Some Studies on Similarity-based Methods for Linear Optimization and Master Production Schedule

A Thesis
Submitted in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy
by

Devanand
(134190003)

Supervisors:
Prof. Ashutosh Mahajan
Prof. N. Hemachandra
and
Tushar Shekhar

Industrial Engineering and Operations Research
Indian Institute of Technology Bombay
Mumbai 400076 (India)

8 May 2023

Approval Sheet

This thesis entitled "Some Studies on Similarity-based Methods for Linear Optimization and Master Production Schedule" by Devanand is approved for the degree of Doctor of Philosophy.

____

____

____

Examiners

____

____

____

Supervisor (s)

____

Chairman

Date: ______________

Place: ______________

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I declare that I have properly and accurately acknowledged all sources used in the production of this report. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Devanand

Date: 8 May 2023 (134190003)

Abstract

Mathematical optimization consists of systematic procedures to model and solve many real-world problems in various disciplines, such as computer science, management science, and economics. Despite the advancement of mathematical programming, backed by heavy computational power, there is no universal method for solving all optimization problems - we have to select an appropriate algorithm for the specific problem. However, the general mathematical procedures used for solving many “difficult” optimization problems can be thought of as solving the related sequence of “easy” mathematical problems. In this thesis, we focus on two procedures that solve a sequence of easy problems, the linear programs (LPs) that we can solve in polynomial time using mathematical programming, to obtain the solution of difficult problems. We exploit the structural relationship among LPs to enhance the underlying strategies.

The first procedure is a “branch-and-bound” algorithm for solving mixed integer programs (MIPs) belonging to the $\mathcal{NP}-$hard class. The branching procedure is an essential step in the branch-and-bound. We detail popular branching procedures and investigate their issues in solving mixed-integer linear programs (MILPs). One of the issues, even in the state-of-the-art branching rule, we primarily focus on is the unnecessary use of “strong branching” calls at nodes in the branching process. The proper use of an expensive strong branching, which solves two LP-relaxations for each branching candidate, motivates us to devise the concept of ‘similarity’ between the current node and the nodes already explored in the tree. Using information from “similar” nodes, we estimate the change in the objective value for each branching candidate, much like reliability branching, to select the variable to branch on. The idea develops into a new branching procedure that effectively exploits the information generated from explored nodes. We call it “SimBranch”. We develop efficient procedures for implementing this scheme, perform computational experiments on benchmark instances, and show the results with the default scheme of an open-source optimization solver (CBC).

The second procedure is a “lexicographic” method for solving hierarchical-multiobjective programs (h-MOLPs), the multiobjective LPs where the order of priorities among the objectives is specified. We study the methods and challenges of two popular lexicographic methods, “constraint-addition” and “variable-fixing”. It includes the derivation of the variable-fixing rule and the theoretical justification of its equivalence with the constraint-addition rule. The study also emphasizes the issue of reoptimizing in their solving process. This further motivates us to introduce a concept of ‘similarity’ between LPs solved in the lexicographic process. The idea of similarity developed into a new lexicographic technique called “ SimLex”. It exploits the structure of the underlying hierarchical model by monitoring the changes in the input parameters and leverages reoptimization - to decide whether to solve the current linear program from scratch or use the available feasible solution obtained from the previous LP solve. We show the computational effectiveness of our approach by comparing it with the standard lexicographic methods available in CPLEX for some hierarchical models chosen from benchmark h-MOLP instances.

Apart from studying the above two procedures for MILPs and h-MOLPs, our contribution to industry problems is to perform a detailed study of one of the main components in master planning in manufacturing industries, known as “master production schedule” (MPS), and some related restrictions associated with it. We mathematically model the MPS as h-MOLP, study popular MPS business objectives, and introduce a toy example called the “potato chip model” to explain the modeling steps. Unlike the lexicographic method, combining objectives using a weighted-sum approach avoids solving several single objective linear programs but faces challenges in obtaining a Pareto optimal solution. We develop customer-specific rules to combine the weighted-sum method with the lexicographic method for computing the underlying MPS. On the computational front, we empirically show the benefit of our idea by implementing and running it for some industry datasets and comparing it with the standard lexicographic method.

Further, we study the challenges in evaluating MPS due to supply chain process restrictions in some industries that produce multiple products from the same assembly line and face a trade-off between inventory and production changeover, known as the “campaign planning” (CP) problem. We study the existing procedure that handles campaign planning restrictions and their limitations. We study the existing procedure that handles campaign planning restrictions and their limitations. We address the issues and observe a significant improvement over two supply chain problems by modeling them as sequential decision problems using the “Cross-entropy” method and providing mathematical models for MPS with CP constraints.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Mathematical Optimization is often used to help a decision-maker choose the 'best' plan of action. Its use has become quite prevalent in industry and business where managers face complex constraints and objectives. Mathematically, an optimization problem may be expressed in many different ways. A form that we consider in this thesis is as follows.

$$\begin{aligned} &\underset{x}{\text{minimize}}\ f(x) \\ &\text{subject to } g_i(x) \leq b_i,\ i = 1, \ldots, m, \\ &\qquad\qquad x_i \in \mathbb{Z},\ i = 1, \ldots, d. \end{aligned} \tag{1.1}$$

Here $x \in \mathbb{R}^n$ is the decision variable, $m,\ n$ and d, where $d \leq n$, are non-negative integers, and $f : \mathbb{R}^n \to \mathbb{R}$, is the objective function consisting of k decision functions. The inequalities $g_i(x) \leq b_i$, where $g_i : \mathbb{R}^n \to \mathbb{R},\ b_i \in \mathbb{R},\ i = 1, \ldots, m$, are called *constraints*. The set $X = \{x \in \mathbb{Z}^d \times \mathbb{R}^{n-d} \mid g_i(x) \leq b_i,\ i = 1, \ldots, m\}$ is called the feasible region of the mathematical optimization problems (1.1). These problems are also sometimes called mathematical programs.

Mathematical programs are classified into different categories based on the decision variables, objective function, and constraints. For problem 1.1, we can categorize it according to the type of decision variable x, the feasible set X, and the input parameters, $m,\ n,\ d$ and, k. Some of the prominent categories are:

1. Constrained and Unconstrained Program: If none of the constraints restrict the problem, i.e., $X = \mathbb{R}^n$, we call it an *unconstrained optimization problem*. Otherwise, it is a *constrained program*.

2. Linear and Nonlinear Program: If any function f or $g_i,\ i = 1, \ldots, m$ is nolinear, we call the problem a *nonlinear program* (NLP). If the objective function and constraints are linear, we call it a *linear program* (LP).

3. Integer, Mixed-Integer and Continuous Program: The problem is said to be an *integer program* (IP) if $d = n$. If $d = 0$, it is called a *continuous program*. If $0 < d < n$, it is called a *mixed-integer program* (MIP). A MIP with linear objective and linear constraints is called a *mixed-integer linear program* (MILP). A MIP is known as a *mixed-binary program* (MBP) if its integer variables are restricted to be binary, i.e., $x_i \in \{0, 1\}$ $i = 1, \ldots, d$.

4. Single Objective and Multiobjective Program: The problem (1.1) is a *single-objective program* (SOP). Sometimes one may have more than one objective function. The output of the objective function is then a solution vector, i. e., $f : \mathbb{R}^n \rightarrow \mathbb{R}^k$. We call it a *multiobjective program* (MOP). An LP, an IP, and, a MIP, with $k > 1$, are generally referred to as a multiobjective linear program (MOLP), a multiobjective integer program (MOIP), and a multiobjective mixed-integer program (MOMIP), respectively. Most of the literature on mathematical optimization focuses on SOPs and refers to them as optimization problems. This thesis uses the term "optimization problems" for the single-objective programs and uses MOLP to refer to multiobjective linear programs.

The categories above are not exhaustive. There are several other important categories that we have not listed as they are not studied in this thesis. An interested reader may refer [2] to for other categories. A point x is *feasible* to optimization problem (1.1) if $x \in X$. We call X a feasible set consisting of all the points feasible to the problem. A point x^* is optimal for the problem if 1) x^* is feasible and 2) the value of the objective function at x^* is not greater than that of any other feasible solution. That is, $f(x^*) \leq f(x)$ for all feasible x. In the case of MOP, it is not easy to compare the objective values at two feasible points. We require a comparison of solution vectors. To obtain a superior solution, instead of optimal solution, we use the concept of dominance for two solutions and the *Pareto solution*. We will discuss it in detail in Section 1.3.

Based on the complexity of the problems, we can divide them into two major classes.

1. "easy" - Problems that we can solve in *polynomial* time using some mathematical programming. Linear programs are considered easy problems as there are available optimization solvers to solve them in polynomial time.

2. "difficult" - Problems that belong to the $\mathcal{NP}-$hard class [3]. As we increase the input size, the time for solving the problem using any known algorithm increases exponentially. MILP comes under this class because no algorithm can guarantee to provide the optimal solution in polynomial time.

Mathematical programming includes developing theories about the form of a solution, constructing algorithms or procedures to seek a solution, formulating problems into mathematical programs, etc. Due to the computational complexities, some of the procedures solve a series of easy optimization problems to get the solution to a difficult problem. This thesis focuses on two strategies that solve a sequence of LPs to get the solution to a difficult problem and exploit the structural relationship among LPs to enhance the performance of the underlying procedures. These strategies are 1) *branch-and-bound* algorithm to solve an MILP and 2) *lexicographic procedure* to solve a *hierarchical*-MOLP. We explain these terms briefly.

Branch-and-bound algorithm

The branch-and-bound (B&B), proposed by Land and Doig [4], is a general framework and a widely-used methodology for producing exact solutions to $\mathcal{NP}-$hard optimization problems. It is a strategy of *divide-and-conquer* where we partition the feasible region into smaller regions and then, if required, further partition the subdivisions. In general, there are a number of ways to divide the feasible region, and as a consequence there are a number of branch-and-bound algorithms. For solving an MILP, the B&B constructs a search tree by successively dividing the problem into subproblems based on the *LP-relaxation* information at a node. LP-relaxation of an MILP is the LP obtained by relaxing the integrality restrictions in the MILP.

A MIP denotes an optimization problem where some decision variables are restricted to integers. MIP appears in many areas, including operations research, power systems, health-care, supply chain industries, etc. There are many applications such as scheduling [5], planning, shortest path-finding [6], and optimizing complex systems such as those arising in transportation, telecommunications, etc., that can be modeled in MIPs. We will use the terms MILP and MIP interchangeably as this thesis only considers optimization programs consisting of linear objective functions and constraints.

We define an MILP as the following optimization problem:

$$\begin{aligned} \textbf{MILP} : \underset{x}{\text{minimize}}\ & c^{\mathrm{T}}x \\ \text{subject to}\ & Y := \{x \in \mathbb{R}^n \mid Ax \le b,\ x_i \in \mathbb{Z},\ i \in I\}, \end{aligned} \tag{1.2}$$

where $c,\ b$ are given rational vectors of size $n,\ m$, respectively, A is a given rational matrix of size $m \times n$ and I is a given index set of variables constrained to be integers. B&B is the most popular method for solving an MILP. Aided by several other enhancements like cutting planes, presolving, heuristic search, etc., B&B provides provable optimal solutions to MILPs. Its main components are the methods for creating and solving relaxations, node selection strategies, and branching schemes. Associated with B&B is a search tree

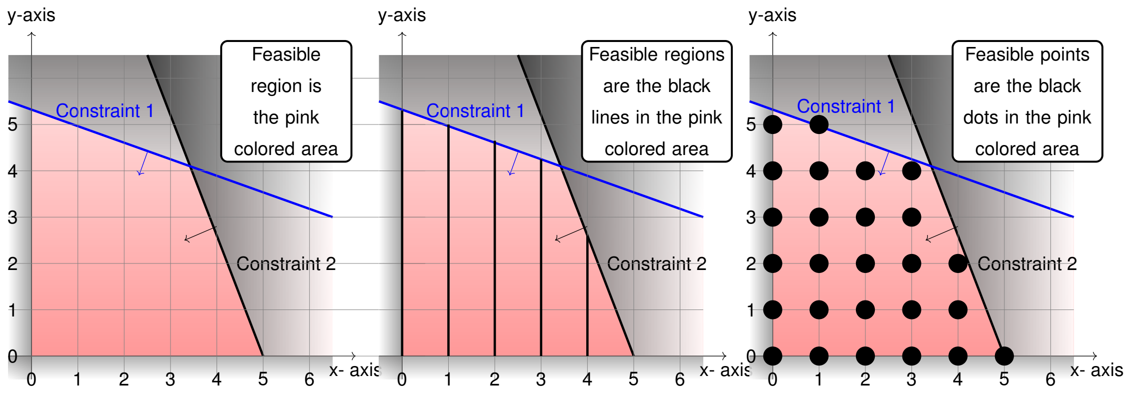


Figure 1.1: A linear, a mixed-integer and a pure integer feasible set (Read from left to right)

consisting of nodes and edges that denote the MILP subproblem and the choice of the branching decision, respectively. The root node corresponds to the original MILP. This B&B tree can grows exponentially in the number of integer variables. Unlike LP, MILP lies in the class of $\mathcal{NP}-$hard problems. As the size of input parameters increases, the difficulty of solving MIPs using current state-of-the-art algorithms can increase at an exponential rate. An LP is an MILP with all the integer variables are relaxed to continuous variables of the following form:

$$\begin{aligned} \textbf{LP} : &\underset{x}{\text{minimize}}\ c^{\mathrm{T}}x \\ &\text{subject to}\ \ Z = \{x \in \mathbb{R}^n \mid Ax \le b,\ x_i \in \mathbb{R}_+,\ i \in [n]\}, \end{aligned} \tag{1.3}$$

Figure 1.1 depicts the feasible regions of linear set, an integer set and a mixed-integer set. Clearly, linear set is the LP-relaxation of the integer set. The image in the middle represents the feasible region of the MIP where continuous vertical lines indicate the variable $x \in \mathbb{Z}$ and the variable $y \in \mathbb{R}$. LP-based B&B method works as follows: First, the integer constraints $x_j \in \mathbb{Z}$ are relaxed to $x_j \in \mathbb{R},\ \forall j \in I$. The relaxed problem is an LP - which is much easier to solve both in theory [7] and practice than the MIP. If the solution of the LP-relaxation, say $\tilde{x}$, is integer feasible, then $\tilde{x}$ is optimal to MIP as well. If the feasible region of the LP-relaxation is empty, then so is the feasible region of the MIP and we can stop. If the LP is unbounded, then the MIP is either unbounded or infeasible and we can again stop. Otherwise, the LP solution value provides a lower bound on the optimal objective value of the MIP (1.2) and $\tilde{x}_j \notin \mathbb{Z}$ for some $j \in I$. In this case, the search space for the optimal solution is divided into two or more parts by branching. Each forms the subproblem and points to a child node in the B&B tree. The node selection rule is the procedure for choosing one node to solve the associated subproblem from the list of unexplored nodes. We study LP-based B&B, popular node selection strategies and branching procedures in B&B in Section 1.2.

Lexicographic method

The second procedure we focus on is the lexicographic method (LM). It is a preemptive priority-based procedure to solve hierarchical multiobjective programs (h-MOP). A multiobjective program (MOP) also named as vector optimization, Pareto optimization, or multicriteria optimization is the area of multi-criteria decision making where more than one objectives are involved and the target is to solve them simultaneously. The credit of introducing the idea of optimization problem with multiple objectives goes to Y. Edgeworth [8] and Vilfredo Pareto [9]. They brought the theory of indifference curve and, for the first time, mentioned the difficulty of obtaining its solution, and developed the basic concept of optimality for MOP in the context of economics, which is now referred to as the "Pareto optimal solution" (we will discuss later in this Section). We refer the readers to [10, 11, 12, 13] for various works and surveys of MOP.

This thesis focuses on the *hierarchical- MOLP* (h-MOLP). It is the MOLP where the order of priorities among the objectives in the problem is specified. We study the following h-MOLP with bounded variables, i.e., the variables that are upper and lower bounded by some finite known values:

$$\begin{aligned} \texttt{lexmin}\, & {c^1}^{\mathrm{T}}x,\ {c^2}^{\mathrm{T}}x, \ldots, {c^t}^{\mathrm{T}}x \\ \text{subject to}\ \ & Ax = b, \\ & l \le x \le u, \end{aligned} \tag{1.4}$$

where $c^k,\ k = 1, \ldots, t$, are cost vectors, b is a vector of size m and A is a rational matrix of size $m \times n$. l and u are lower and upper bound parameter vectors. Here the term "lexmin" denotes lexicographic minimum, also denoted as ${c^1}^{\mathrm{T}}x \gg {c^2}^{\mathrm{T}}x \ldots \gg {c^t}^{\mathrm{T}}x$. It signifies that first objective $({c^1}^{\mathrm{T}}x)$ is much more important than the second objective $({c^2}^{\mathrm{T}}x)$ which is, on its turn, much more important than the third one $({c^3}^{\mathrm{T}}x)$, and so on and, the last objective $({c^t}^{\mathrm{T}}x)$ is of least importance. The additional upper and lower bounds restrictions make it more practical in industry applications. There are broadly two approaches for solving h-MOLP, preemptive method and non-preemptive method [12]. This thesis focuses on the lexicographic method, a preemptive priority-based method for h-MOLP. One of the benefits of preferring the lexicographic method is that it always provides a Pareto optimal solution [14]. We study two popular lexicographic methods of h-MOLPs that solve a sequence of LPs, *constraint-addition rule* and *variable-fixing rule*, in detail in Section 1.3.

Apart from studying various branching strategies for MILPs and lexicographic methods for h-MOLPs, our contribution to industry problems is to perform a detailed study of one of the main components in master planning in manufacturing industries, known as *master production schedule* (MPS), and some related restrictions associated with it. MPS expresses planning for the production of each commodity in specific configurations, quantities, and dates, consisting of many business objectives [15]. We consider MPS as a h-MOLP and enhance the existing popular methods used to solve MPS. We discuss them in Section 1.4.1.

Further, we study the challenges in evaluating MPS due to supply chain process restrictions in some industries, especially process industries. They produce multiple products from the same assembly line and face a trade-off between inventory and production changeover. Switching production from one product to another incurs an overhead in cost and time. Such a problem of planning the production of batches of different products, known as a *campaign planning* (CP) problem, makes the MPS difficult. We study the existing procedures that handle campaign planning restrictions and their limitations. We address one of them by modeling it as a sequential decision problem, solving it using the *Cross-entropy method*, and providing a mathematical model for MPS with CP constraints. We discuss this in detail in Section 1.5.

Before proceeding further, we summarize the main goals of this research as follows:

1. Study branching rules in B&B for solving MILPs, and lexicographic rules for solving hierarchical MOLPs and investigate the challenges these rules face.

2. Define the concept of similarity between a sequence of “easy” LPs while solving “hard” MILPs and MOLPs using B&B and lexicographic methods, respectively.

3. Develop new branching and lexicographic rules based on similarities between LPs for solving MILPs and h-MOLPs, respectively. Study their computational effectiveness over existing methods.

4. Study MPS as h-MOLP and analyze complexities and challenges in its solution computation.

5. Study supply chain campaign planning problems as 1) sequential decision problems, and 2) MIP, and explore the challenges to solve them.

1.1 Notation

Unless otherwise mentioned, this thesis uses the following notation throughout: We use lowercase italics to denote scalars, e.g., α, c, κ. A one-dimensional vector also follows the same notation as a scalar, though the context makes the reader distinguish between them. The set of real numbers will be denoted by $\mathbb{R}$, that of integers by $\mathbb{Z}$, that of natural numbers by $\mathbb{N}$ and that of rational numbers by $\mathbb{Q}$. Upper case italicized letters e.g. A, S represents sets. For a given set A, $A_+ := \{x \in A \mid x \geq 0\}$ represents the set of non-negative elements. So, $\mathbb{Z}_+$ and $\mathbb{R}_+$ represent the set of non-negative integers and linear numbers, respectively. The ceiling and floor of the scalar a are represented by $\lceil a \rceil$ and $\lfloor a \rfloor$, respectively. For a non-negative integer t, we define $[t] := \{1, 2, \ldots, t\}$ if $t > 0$ and $[t] := \emptyset$ if $t = 0$. The dot product of two vectors $u \in \mathbb{R}^n$ and $v \in \mathbb{R}^n$ is denoted by $u^{\mathsf{T}}v$. Similar to sets, matrices are denoted with italic capital letters, e.g. A, S. The context lets the reader distinguish between the matrix and the set. For an $m \times n$ real matrix $M \in \mathbb{R}^{m \times n}$ we use M^T to denote the transpose of M. We use M_i to denote the i^{th} column of a matrix, M.

1.2 Branch and Bound Algorithm

The branch-and-bound algorithm (B&B), a "divide and conquer" method, successively divides the problem into smaller subproblems. The process can be represenetd by a tree called a branch-and-bound tree (B&B tree), where each node corresponds to the subproblem. At any point in time, the subproblem occupies one of the states of the node, 1) the root node - a starting stage where we solve the relaxation of the original optimization problem, 2) the solved node - a node whose child we have already explored, 3) the feasible node - a node that yields a feasible solution to the original problem and might update the incumbent solution and 4) the pruned node - a node that will not be investigated further. We have shown these states in Figure 1.2.

B&B for solving an MILP follows the following steps: We start with the root node that corresponds to the MILP, the original problem which we want to solve. We drop all the integrality constraints in MIP and solve its LP-relaxation. We have defined MILP in the model (1.2) and its LP-relaxation in the model (1.3). The solution of the LP-relaxation is also the solution to MILP if there are no integer violations. In such a case, we stop with this solution. Else, if it is infeasible, so for the MILP and we stop. If it is unbounded, it will be either infeasible or unbounded to the MILP and we stop. If some of the variables which are restricted to be integer are fractional in the solution of LP-relaxation, it will

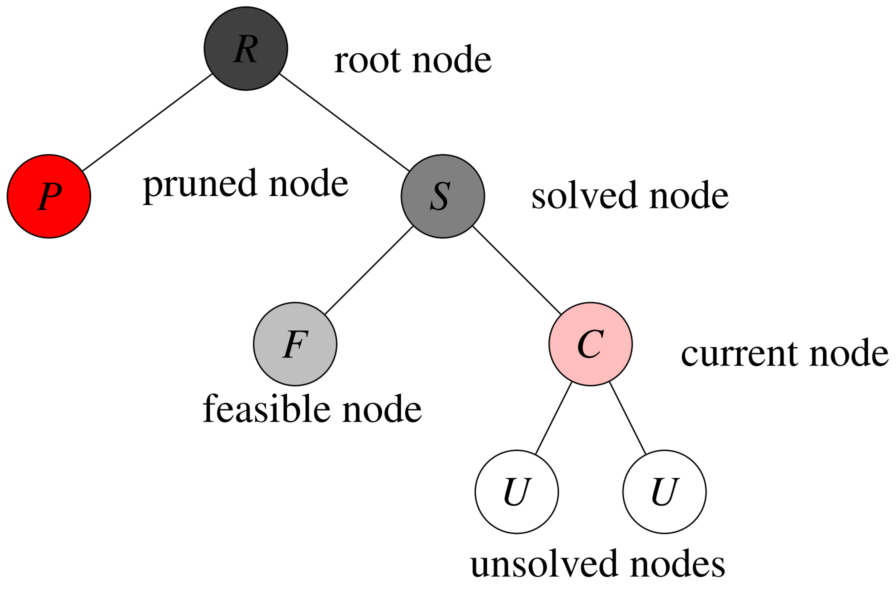


Figure 1.2: Various states of a node in B&B

not be a feasible solution for the MILP. The optimal value of the LP-relaxation gives the lower bound to the optimal value of the MILP. We divide the search space into two or more parts, using a technique called *branching*, each forming an MILP. The union of feasible regions (sets) of subproblems corresponds to the feasible set of the original problem and the intersection may be an empty set. Each subproblem corresponds to the child nodes of the root node. From such unexplored child nodes, we select one using a technique called *node selection*. We follow the same procedure with the selected node as we do with the root node. The optimal solution value of a subproblem must be at least the lower bound obtained from the LP-relaxation of its parent. If a subproblem provides a feasible integer solution, then we get an upper bound on the optimal value of (MILP) of the original problem. Nodes associated with all subproblems that have lower bounds more than this upper bound can be pruned in the B&B tree. We continue until, we process all the nodes in the B&B tree. The Algorithm 1 mentions the steps in a LP-based B&B to solve the MIP (1.2).

The procedure in the B&B points to two important questions one may ask, one is about the "node selection", and the other is about the "branching strategy". This thesis focuses on the second question, the branching procedure, specifically a variable-branching rule. While discussing branching procedures, we assume that the node selection method is known and fixed for any experiments with various branching methods.

Algorithm 1: Branch and Bound

```
Input: $N^0$: root node pointing to the original problem $MLP^0$ = MILP.
Output: $z^*$, $x^*$: optimal value and optimal solution to the MILP.
Initialize: $L = \{N^0\}$, $x^* = \phi$, $\underline{z} = -\infty$, $\overline{z} = \infty$.
 Step 1: if $L$ is empty: no node is available then
    Optimal solution is $x^*$, Optimal value is $z^*$;
    Stop.
else
    Choose a node $N^i$ in $L$. Update $L = L - \{N^i\}$. /* Node Selection */
end
Step 2: Solve LP-relaxation, $LP^i$ of $MIP^i$ of node $N^i$. /* Bound */
if $LP^i$ is infeasible then
    go to Step 1.
else
    Let $x^i$ and $z^i$ be optimal solution and optimal objective value of $LP^i$. Step 3:
    if $z^i \geq \overline{z}$ then
        go to Step 1 ;
    else
        if $x^i$ is feasible to the MILP then
            set $x^* := x^i$ $\overline{z} := z^i$. Delete all nodes $N^k$ from $L$( pointing to those
                problem $MIP^k$) that have optimal value $z^k > \overline{z}$ and go to Step 1.
                /* Prune */
        else
            Step 4: From $MIP^i$, construct $MIP^i_1, \dots, MIP^i_k$ problems, $k \geq 2$,
                with smaller feasible regions (by adding linear inequalities)
                whose union does not contain ($x^i$), but contains all the solutions
                of $MIP^i$ and each solution from $MIP^i_t$, $t \in [k]$ is an integer
                feasible solution to the selected MIP. Add new nodes $N^i_1, \dots, N^i_k$
                to $L$ and go to Step 1. /* Branch */
        end
    end
end
Step 5: Update $\underline{z} = \min\{z^i \mid N^i$ pointing to $MIP^i \in L\}$. If $\underline{z} \geq \overline{z}$, stop. Else, go to
 step 1.
```

1.2.1 Node Selection

There are many node selection schemes, broadly categorized into two parts, *informed* and *un-informed* node selection strategies (also known as *blind search*). An un-informed search is the one where the agent has no information about the number of steps and corresponding possible cost needed to reach from the current state to the goal. *Breadth-first search* (BFS) and *depth-first search* (DFS) are two such popular techniques [16]. On the other hand, an informed search is an information-centric search that takes into account the goal at each search step and provides some additional information. *Best-first search (BestFS)* and *A* search* are the popular informed search strategies. In DFS, we start from the root node and explore the nodes as deep down as possible along each branch before returning (backtracking). Usually, it is preferable over other node selection strategies for the following reasons.

1. In general, most integer feasible solutions lie deep in the tree. DFS can find them faster than the other procedures. For some problem instances, finding an optimal solution is time consuming and difficult. For them finding a feasible solution at an early stage is essential if we want to abort the solver early because it cannot compute an optimal solution in a reasonable time.

2. DFS is beneficial for those problems that do not have an objective function and where the solving process is only to find a feasible solution. It helps in solving pure feasibility problems like SAT.

3. The DFS is a recursively defined function, which simplifies the coding and thus makes the implementation more manageable.

4. As one moves down the enumeration tree, each subproblem refers to the subsequent nodes obtained from the preceding one with few changes in the relaxation. Variable branching simply adds (or updates) an upper or lower bound for a particular variable. These few changes between the parent and child nodes sometimes speed up the solution of the subproblems associated with these child nodes by using the optimal solution of the problem associated with the parent node. This process is called *reoptimization*. We will discuss the concept in Section 1.3.

However, problems may arise if an optimal solution is located near the root node and the DFS prefers a long path (especially in the unbalanced tree) on which no optimal solution is located. Unlike, DFS, BFS explores the nodes near the root before processing

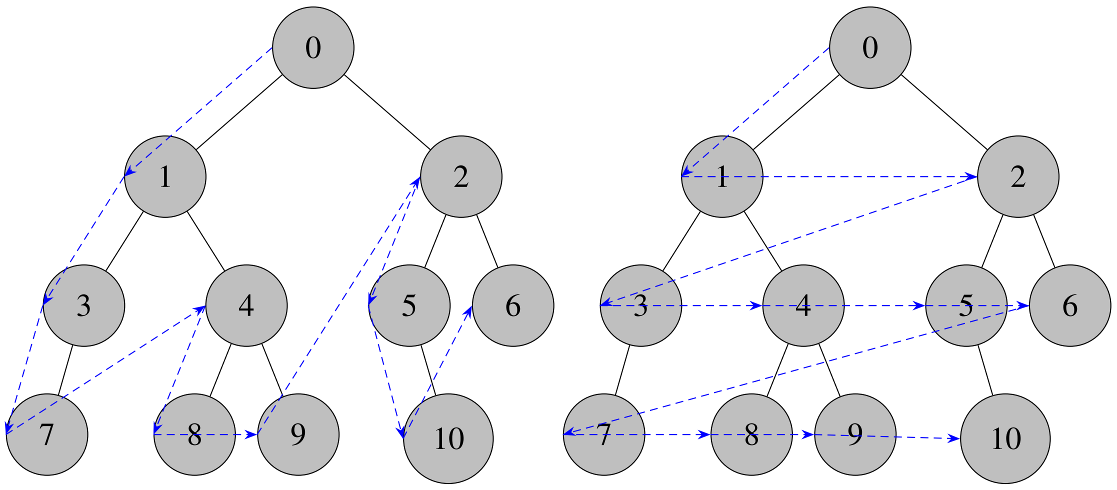


Figure 1.3: The strategy of the depth first search (DFS) and the breadth first search (BFS)

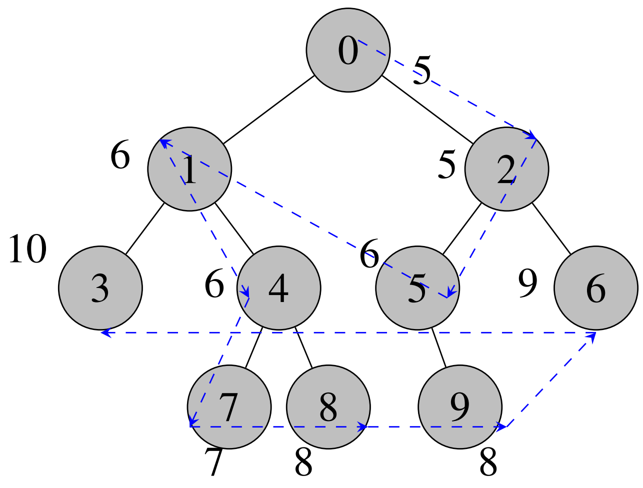


Figure 1.4: Best first search

the subproblems positioned far from the root node. This strategy has the advantage that an optimal solution is always found that is closest to the top node of the tree (especially for unbalanced search trees). In general, however, it is seen that complete solutions are usually in greater depth. The BFS naturally cannot use pruning rules compared to the incumbent solution. It leads to a relatively high memory requirement than the DFS. The is the main reason why we do not prefer it in the B&B context. Figure 1.3 illustrates the order in which BFS and DFS process the nodes in B&B. The problem with the blind search in both BFS and DFS is that it does not use information about the problem structure. It led to spending a significant amount of time exploring a poor search space region. BestFS is an informed node selection strategy that, unlike the blind search, uses node information in its selection decision. In the B&B context, it aims to improve the global lower bound as fast as possible by always selecting a subproblem with the smallest lower bound of all new nodes. The benefit of BFS is that it does not stick to exploring the nodes in one branch before backtracking to another branch. It is one of the reasons that BestFS is often able to find the optimal solution earlier in the search process. Figure 1.4 depicts the order in which BestFS processes the nodes. Nodes 1 and 5 are processed before they meet the optimal node 4. However, when the selection depends on a tie-breaking rule, the BFS may spend a lot of time in the middle regions of the search tree and never find an optimal solution. This situation arises when more than one node points to the subproblem whose LP-relaxation solution consists of an optimal solution. A trade-off among these static selection approaches leads to the development of a variant of DFS and BestFS strategies [17]. A default node selection strategy in SCIP combines all three of these strategies. It starts with DFS and continues with a few consecutive nodes. Then, a node with the best estimate is chosen. At a particular frequency, a node with the smallest dual bound is selected instead of a node with the best estimate [18].

1.2.2 Branching Strategy

The branching scheme has a significant impact on the performance of B&B and is the focus of our study. The importance of selecting a good branching candidate has been recognized early by [19, 20] and is still an active area of research. As we see in Algorithm 1, it is a dividing procedure that divides the problem, associated with the currently exploring node into smaller subproblems. It leads to the formation of child nodes to the current processing nodes, where each node points to the respective smaller subproblem. In our thesis, we focus on a generic branching procedure, known as a *variable-branching* scheme. It is a most natural ways of branching where we select one variable out of the set of “fractional" variables $C = \{i \in I \mid \widetilde{x}_i \notin \mathbb{Z}\}$ and create two subproblems by adding the constraints $x_j \leq \lfloor \widetilde{x}_j \rfloor$ and $x_j \geq \lceil \widetilde{x}_j \rceil$:

$$\begin{aligned} &\text{minimize } c^{\mathrm{T}} x \\ &\text{subject to } \ Ax \leq b, \\ &\qquad x_j \leq \lfloor \widetilde{x}_j \rfloor \\ &\qquad x_i \in \mathbb{Z}, \ i \in I. \end{aligned} \qquad \text{and} \qquad \begin{aligned} &\text{minimize } c^{\mathrm{T}} x \\ &\text{subject to } \ Ax \leq b, \\ &\qquad x_j \geq \lceil \widetilde{x}_j \rceil \\ &\qquad x_i \in \mathbb{Z}, \ i \in I. \end{aligned}$$

The two subproblems are also MILPs. Their optimal solution values must be at least the lower bound obtained from the LP-relaxation of their parent. In the next chapter, Chapter 2, we will study various braching procedures in detail. Let us understand the steps in B&B and the variable-branching procedure with the following example:

$$\begin{aligned} \textbf{IP1}: \ &\text{minimize } \ -17\,x - 12\,y, \\ &\text{subject to } \ S := \{ \text{ Constraint } 1 := 10x + 7y \leq 40, \\ &\qquad\qquad \text{Constraint } 2 := x + y \leq 5, \\ &\qquad\qquad x, \ y \geq 0 \\ &\qquad\qquad x, \ y \text{ are integers } \}. \end{aligned} \tag{1.5}$$

Figure 1.5 shows the division of feasible regions from the variable-branching scheme where a variable for branching is assumed to be selected from the candidate set randomly when solving the problem (1.5). The node selection procedure is random. We start with the LP-relaxation of **IP1**, whose feasible region S is denoted by black coloured dots inside the pink coloured regions. The pink coloured region is the feasible region for the LP-relaxation of **IP1**. The optimal solution of the relaxation, $(\frac{5}{3}, \frac{10}{3})$, indicated by a green dot, does not coincide with block coloured dots - it's not the feasible solution of **IP1**. It invokes a branching procedure. We select x from the candidate set, $\{x, \ y\}$. The branching step divides **IP1** into two subproblems. $S1$ and $S2$ are the feasible reasons of the subproblems, whose feasible regions for the LP-relaxation of the subproblems are the pink coloured regions. The optimal solution to the LP-relaxation of the first subproblem (with feasible region $S1$) is $(1, \ 4)$. It is also feasible to the integer program. We do not explore this subproblem further. The optimal solution for the LP-relaxation of the other subproblem is $(2, \ 2.86)$. Since the solution is not integer feasible, we perform a branching operation on the selected branching candidate y with the solution value 2.86. This results in two subproblems 1) one with the addition of the constraint $y \geq 3$ is infeasible, and 2) one with the addition of the constraint $y \leq 3$ is feasible to LP. The feasible region of the new subproblem is $S3$. The optimal solution of its relaxation (2.6, 2.0) is again integer feasible. This calls the branching procedure. We continue the similar process of creating new subproblems, solving the relaxation, and deciding whether to process further, stop or branch. We found (4, 0) to be the optimal solution for **IP1**. The optimal value obtained is 68.

This B&B tree can be quite large and grows exponentially with the number of integer variables. Unlike an LP, an MILP lies in the class of $\mathcal{NP}-$hard problems [3], so the number of LP-relaxations required to solve an MILP can be exponentially large in the worst case. However, with the advances in LP- and MILP-based solvers, many medium and large-size problems can be solved in reasonable amount of time.

One popular variable branching strategy is to select a variable that leads to the largest improvement of the lower bound and resulting in fewer nodes in the B&B tree [21]. The idea, called *strong branching* (SB) is to simulate the change in the lower bound by solving two LP-relaxations for each candidate and identifying the changes in the bounds explicitly. The candidate that pushes the lower bound the most is selected for branching. SB has been observed to reduce the number of nodes in the tree but requires a large amount of computational time to evaluate all candidates. It is one of the widely used methods that later became the integral component of the other state-of-the-art branching schemes. *Pseudocost branching* [19] tracks changes in the lower bounds every time a new node is processed. The pseudocost score of a variable candidate is, roughly speaking, the average change in the objective function seen by changing the bounds of the variable. It has been observed to be useful only once the tree has become large and sufficient data has been collected. *Reliability branching* [22] tries to collect the SB scores in the early stages of B&B. Once a sufficiently large sample of scores has been collected for a variable, its average score can be used as an estimate for the SB score. While reliability branching has been shown to outperform previously proposed schemes, the evaluation of SB scores is concentrated at the top of the search tree, and hence the estimates may not be accurate. We will discuss the impact of this limitation, do a literature review and, provide a motivation for a new branching procedure that effectively exploits the information generated from the explored nodes in the B&B tree in Chapter 2.

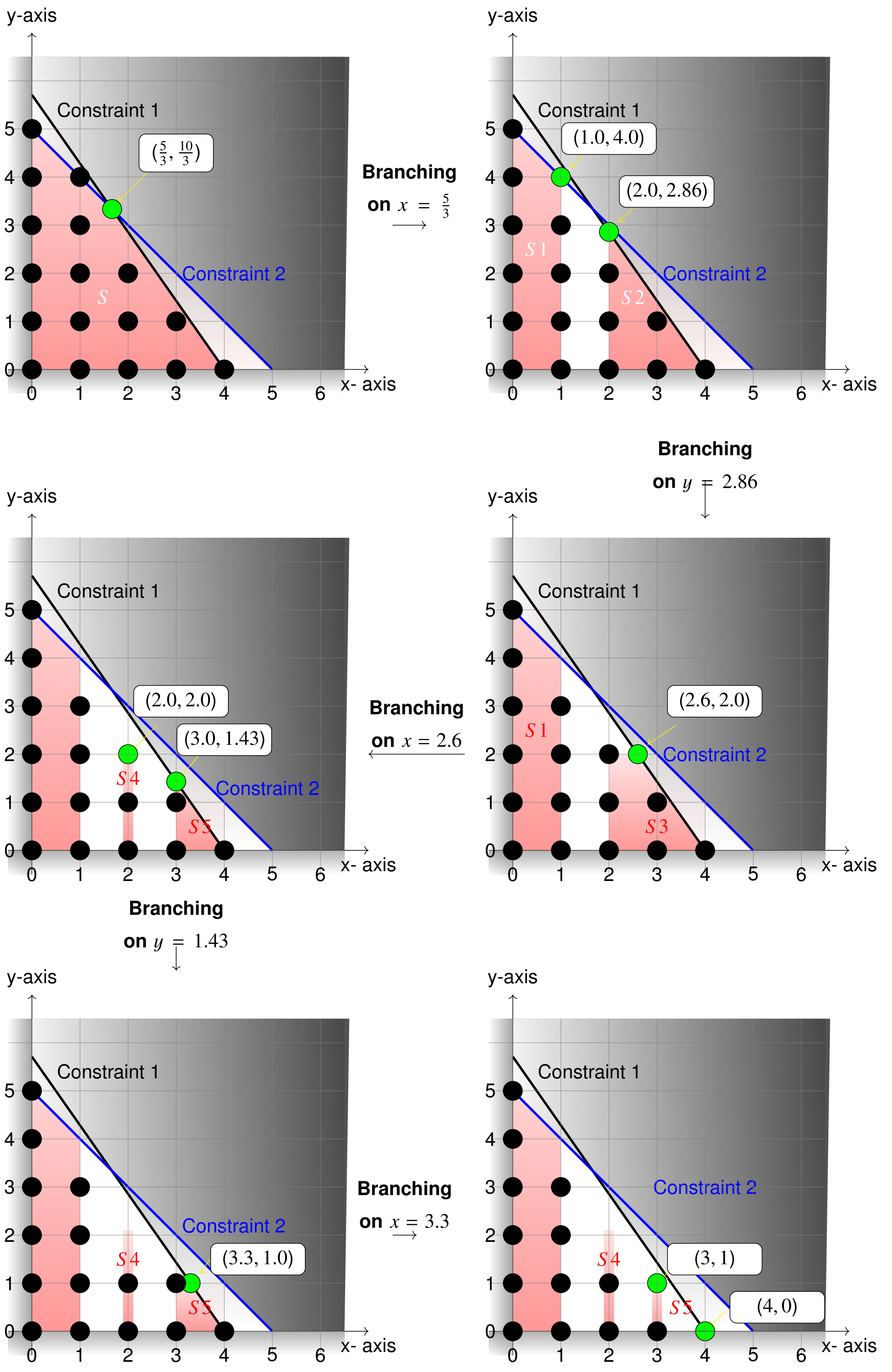


Figure 1.5: Branching steps in B&B for problem IP1

1.3 Multiobjective Linear Program

A Multiobjective linear program (MOLP) is a multiobjective optimization where all the objective functions and constraints are linear. The Mathematical model of an MOLP is expressed as follows:

$$\begin{aligned} \textbf{MOLP} : & \underset{x}{\text{minimize}}\ ({c^1}^{\mathrm{T}}x,\ {c^2}^{\mathrm{T}}x, \ldots,\ {c^t}^{\mathrm{T}}x) \\ & \text{subject to}\ \ X = \{x \in \mathbb{R}^n \mid Ax = b, l \leq x \leq u, x_i \in \mathbb{R}_+,\ i \in [n]\}, \end{aligned} \tag{1.6}$$

where $c^k,\ k = 1, \ldots, t$, are the cost vectors, $t,\ m,\ n \in \mathbb{Z}_+$, b is the rhs vector of size m and A is a rational matrix of size $m \times n$. Input vectors l and u are the lower and the upper bound parameter vectors.

A feasible solution $x^* \in X$ that minimizes all $c^i,\ i = 1, \ldots, t$ simultaneously is called a *ideal-solution*. Clearly, if $x^* \in X$ is the ideal solution then for any $x \in X$, $c^i(x^*) < c^i(x), i = 1, \ldots, t$. Some literature also refer it as *utopia point*. For a non-trivial MOLP where objective functions are conflicting in nature, i.e., increasing objective value of one objective function will lead to a decrease in the other objective functions, no ideal solution exists. If such a solution exists, there is no motivation to consider multiple objectives. We determine the goodness of a solution by a concept called *dominance*. Consider two points, $x_1, x_2 \in X$ in . We say x_1 dominates x_2 (or x_2 is dominated by x_1) if, 1) $c^i(x_1) \leq c^i(x_2)$ for all $i = 1, \ldots, t$ and 2) there exists at least one objective c^j from the list of objectives such that $c^j(x_2) > c^j(x_1)$. The non-dominated set of all feasible space is called “Pareto optimal" solution and the set of Pareto optimal outcomes is often called the *Pareto front*, Pareto frontier, or Pareto boundary. Let us consider an MOLP:

$$\begin{aligned} \textbf{MOLP1} : & \text{minimize}\ (\mathtt{f1} := -5\mathtt{x}_1 + 2\mathtt{x}_2, \mathtt{f2} := \mathtt{x}_1 - 4\mathtt{x}_2) \\ & \text{subject to}\ \ \text{constraint1} : -x_1 + x_2 \leq 3, \\ & \qquad\qquad\ \ \text{constraint2} : x_1 + x_2 \leq 8, \\ & \qquad\qquad 0 \leq x_1 \leq 6, \\ & \qquad\qquad 0 \leq x_2 \leq 4. \end{aligned} \tag{1.7}$$

Figures 1.6 and 1.7 depict its Pareto optimal solution and the Pareto front. Since there are infinitely many Pareto optimal solution exists, selecting one of them asks the

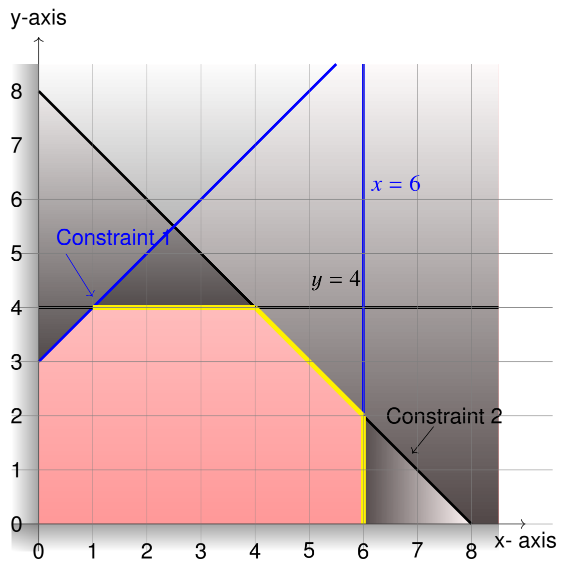


Figure 1.6: Pareto Optimal Set

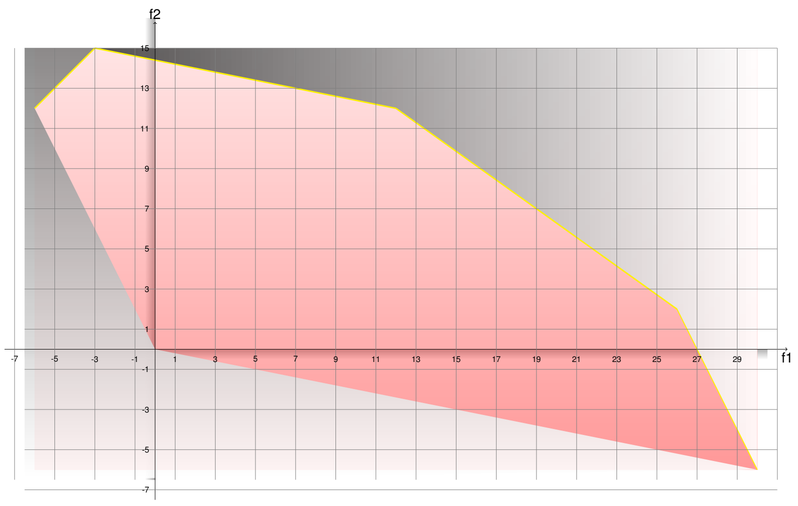


Figure 1.7: A Pareto Front

question of how to solve the MOP and incorporate the preferences of the decision maker. This leads to broadly divide the optimization methods for the MOP into following classes - 1) *apriori* method, where the domain expert (or the decision maker) provides the preference information of the business objectives before solving the MOP, 2) a posteriori method, where the MOP is solved first, to obtain all the Pareto optimal solution, and then the domain expert selects one (or few) of them, and 3) interactive method, where MOP is solved in iterations and at each iteration the domain expert sets the preferences to get the Pareto optimal solution. We focus on the apriori-based class and in that more specifically *lexicographic method*. A lexicographic method solves MOLPs with the available preferences of objectives. Such MOLPs are called hierarchical MOLPs (h-MOLPs). A mathematical definition of a h-MOLP is defined in the model (1.4).

1.3.1 Lexicographic Method

Unlike the common apriori-based MOP method (such as a weighted sum method (WSM)[23]), a lexicographic method (LM) imposes the preferences by ordering the objective functions as per the decision of the domain expert about the significance of these objective functions. Because WSM combines all the objectives, it only needs one LP solver call to obtain the solution of the MOLP. In contrast the WSM, the LM requires many LP solver calls – one call for each objective. To solve the model (1.4) in a hierarchical fashion, we need to solve a sequence of single-objective optimization problems for $k = 1, \dots, t$ as follows:

The objective functions $c^1, c^2, \dots, c^t$, ranked with highest to the lowest order of importance, are available to us. To preserve the hierarchy of the objective functions, we solve as follows:

$$\begin{aligned} \mathrm{LP}^k := \min\ & {c^k}^{\mathrm{T}} x \\ \text{subject to}\ & Ax = b, \\ & c^i x \leq y^i,\ \forall\, i \in [k-1], \\ & l \leq x \leq u, \end{aligned} \tag{1.8}$$

where y^k is the optimal value of the problem $\mathrm{LP^k}$ (it is assumed that the problem LP^k is feasible) for each $k = 1, \dots, t$. We start with computing $y^1 = \{\min {c^1}^{\mathrm{T}} x \mid Ax = b,\ l \leq x \leq u\}$. If the solution is unique, we stop, and the obtained solution is optimal to the h-MOLP. Otherwise, we solve the next immediate linear program, LP^2 with a newly added constraint, ${c^1}^{\mathrm{T}} x \leq y^1$ by preserving the previously obtained solutions. We follow the

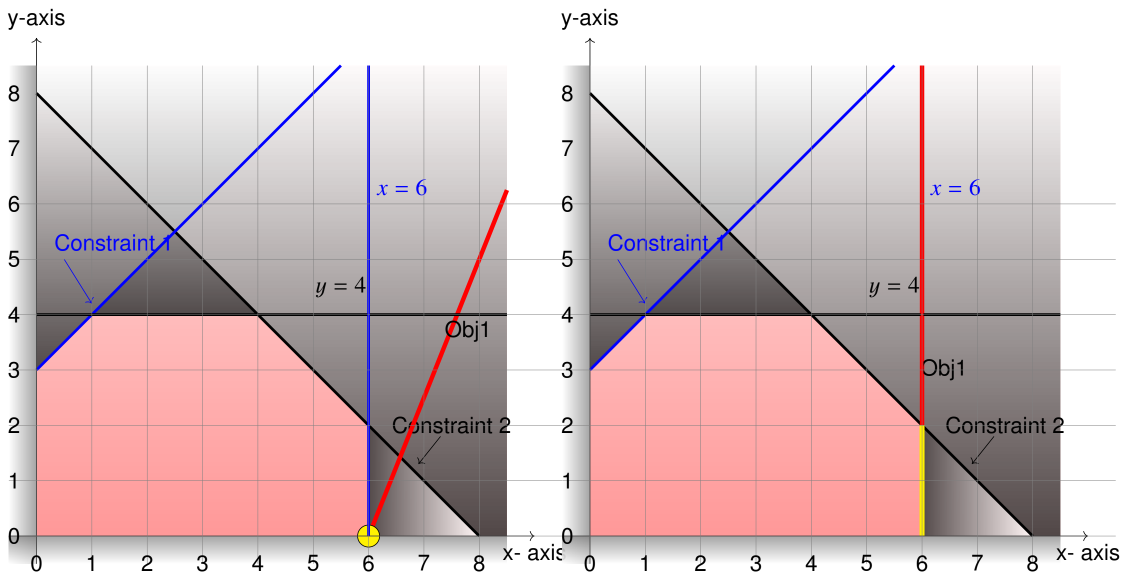


Figure 1.8: Solution set for the first objective for the model (1.9) and the model (1.10) is denoted by the yellow point (6, 0) and line segment (6, 0) – (6, 2)

same procedure until we reach to solve the lowest priority program $\texttt{LP}^t$. We call this *constraint-addition* rule.

After every LP solve in the LM, the uniqueness of the obtained solution will determine whether we should terminate the process or not. The LM and the h-MOLP we solve are interesting only if we have alternate optimal solutions while solving the high-priority objectives. If the current problem has alternate optima, i.e., the solution obtained is not unique, then we continue to solve the next LP. Otherwise, the current solution of LP is also the solution to the h-MOLP. For example, if the preference of the objective functions are known in the model (1.7), we have a reformulation in the h-MOLP as follows:

$$\begin{aligned} \mathbf{h-MOLP1} : & \texttt{lexmin f1} := -5\texttt{x}_1 + 2\texttt{x}_2,\ \texttt{f2} := \texttt{x}_1 - 4\texttt{x}_2 \\ \text{subject to } & \text{Constraint1}: -x_1 + x_2 \leq 3, \\ & \text{Constraint2}: x_1 + x_2 \leq 8, \\ & 0 \leq x_1 \leq 6, \\ & 0 \leq x_2 \leq 4. \end{aligned} \tag{1.9}$$

Now consider the new h-MOLP model which is same as the model (1.9) with only change in the first objective function as follows:

$$\mathbf{h-MOLP2} : \texttt{lexmin}\ -x_1,\ x_1 - 4x_2$$

$$\begin{aligned}\text{subject to }\ & \text{Constraint1: } -x_1 + x_2 \le 3,\\ & \text{Constraint2: } x_1 + x_2 \le 8,\\ & 0 \le x_1 \le 6,\\ & 0 \le x_2 \le 4. \end{aligned} \tag{1.10}$$

Figure 1.8 shows the optimal solution set of the first objective function and the Pareto set of the h-MOLP for models (1.9) and (1.10). There is no requirement of solving the second objective using constraint-addition method for model (1.9) as it has unique optimal solution to the first LP in the iterative procedure of LM. It is denoted by a yellow circle at point (6, 0). Whereas, we denote the optimal solution to the first LP solve for the model (1.10) by the yellow continuous line segment with the two closed end points (6, 0) and (6, 2).

If the solution of an LP is available, it is easy to identify whether it has an alternate optimal solution or only a unique solution exists. For a standard LP defined in the problem (1.3), if x^* is the basic feasible solution and if the reduced cost of every non-basic variables is positive, then x^* is the unique optimal solution [24, Exercise 3.6]. For LPs with bounded variables that we are interested to solve in a sequence for the h-MOLP (1.4), consider x^* as the basic feasible solution for one of the LPs, say LP1. If the reduced cost of every non-basic variables at their upper bound is positive and the reduced cost of every non-basic variables at their lower bound is negative, then x^* is the unique optimal solution [25, Chapter 5]. Given x^* as the optimal solution of the LP^1 in the LM procedure (1.8), if the set of optimal solution $F := \{x \in S := \{x \in \mathbb{R}^n \mid Ax = b, l \le x \le u\} \mid c^1x = c^1x^*\}$ of LP^1 is singleton, then x^* will be the solution to the model (1.4). Otherwise, we find the optimal solution for LP^2, which is a point $y^* \in F$, such that $c^2y^* \le c^2x$ for all $x \in F$. Without additional computational effort many LP solvers provide reduced cost information along with the solution. In practice, it becomes easier to determine the existence of alternate optimal solution using the reduced costs. For LPs with many zero-valued coefficients in their objective functions, the probability of an alternative optimal solution is high. In our contribution to industry challenges, we pick large scale problems posed to h-MOLPs. In most cases, the h-MOLPs for such problems contain many zeros in their objective functions. It motivates us to solve such problems.

A *column-dropping* or *variable-fixing* rule is another type of LM that reduces cost information in the sequence of LP solves. Unlike the constraint addition rule, instead of adding the constraints $c^ix = y^i$, $\forall\ i \in [k-1]$, it equivalently fixes a certain number of variables at one of their upper or lower bounds. For a minimization problem, if the reduced cost of a non-basic variable is positive, we fix its obtained solution value at its lower bound. Similarly, if the reduced cost of a non-basic variable is negative, we fix its obtained solution value at its upper bound. The model (modLP^k) in the method (1.11) mentions the sequence of solves using the variable-fixing rule.

$$\begin{aligned} \text{modLP}^k := \min\ & c^k x \\ \text{s.t.}\ & Ax = b, \\ & x_j = f_j\ \forall j \in J^k \subseteq [n], \\ & l \leq x \leq u. \end{aligned} \tag{1.11}$$

Here J^k denotes the index subset of $\{1, \ldots, n\}$ for which components of the decision variable x_j is fixed at f_j. This procedure requires changes in the bound section, and objective function between two consecutive LP solves. It is different than the constraint-addition rule where changes are realized in the rhs vector, the coefficient matrix, and the objective function vectors. We discuss variable-fixing rule in detail, the benefit of using it over the constraint-addition rule, and its limitations in Chapter 3.

One of the major disadvantages of both the rules is that they can require the solution of many single objective problems to obtain just one solution point. It becomes a challenge for large h-MOLPs, for example, the master production schedule (MPS) in some supply chain manufacturing industries with a large-sized constraint set and many business objectives. To speed up the methods, almost all the optimization solvers provide a feature of using the solution of the high-priority objectives as a starting solution to solve the low-priority objectives. This reuse of the previous solution aims to save computation time. For example, Gurobi [26] uses the advance basis and Cplex [27] uses the advanced indicator flag to allow the user to save and reuse the solution basis. We call it *reoptimization*. It is a technique to solve a new mathematical model by using the available solution of a similar model with slight modification to the new model. We can find a wide range of work of reoptimization in the literature of application of reoptimization, such as scheduling problems[28, 29, 30], Steiner tree [31, 32], covering problems [32], travelling salesman problem [33, 34] and several other applications of reoptimization [35, 36, 37, 38, 39, 40, 41, 42] and warm-start procedures [43, 44, 45, 46] for various optimization problems. Generally, optimization solvers offer two flavors of reoptimization, warm-start - where the solution basis of one of the LPs is saved to solve a new LP and hot-start - where the solution basis is readily available for solving the new LP. A hot-start in B&B for MIPs works well when DFS is chosen as the node selection strategy. The minor change between LPs associated with two adjacent nodes helps in reusing inter-

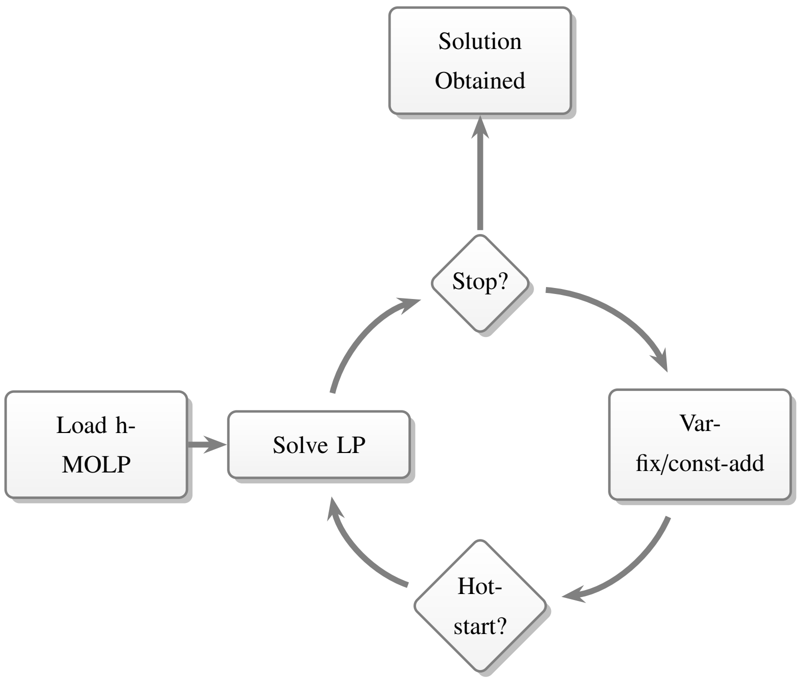


Figure 1.9: Iterative LP solves in LM with hot-start

nal matrix factorization available after the LP solve and results in dramatic performance improvement compared to the situation where we do not consider hot-start.

Reoptimization helps to solve a new model by applying the available solution of a similar model with slight modification in the rhs vector, cost vector, bounds of variables, and coefficient matrix. Figure 1.9 illustrates the flow of LP procedure and its components. LM loads the h-MOLP problem and solves the current LP. It checks the stopping conditions. If it covers all the objectives or the solution of the current LP is unique, the process is terminated. Otherwise, it preserves the preference among objective functions by the variable-fixing or the constraint-addition rule and then it checks if reoptimization is enabled. If so, hot-start (or warm-start) is used to solve the LP with a new updated objective function. The available basis for the new LP, which is always primal feasible, makes hot-start favourable. Because objective functions differ in each LP solve, we can not guarantee dual feasibility. The reason for primal feasibility is that the constraint set of the new LP is the subset of the constraint set of the recently solved LP obtained after adding a constraint or updating the bounds. In Chapter 3, we refer to this subset of constraint-set as the "face" of the constraint set of the LP solved. We will provide some results on it and also give a theoretical guarantee of primal feasibility.

Reoptimization does not always help. In Chapter 3, we will see that there are instances for which it is better to avoid the available starting solution and start afresh. We will also discuss that instead of an ad-hoc decision, a systematic reoptimization in LM can speed up the process. We highlight the challenges in popular lexicographic methods, mainly the limitations in solving a sequence of LPs in their procedures, and devise a new strategy for an adaptive reoptimization that closely look into the fractional changes in LP parameters.

1.4 Master Planning in Supply Chain Planning

At a high level, we consider a supply chain as a network of two or more legally separate organizations linked by material, information, and financial flows. These organizations may be suppliers, manufacturing plants and inventory locations, transportation services, and the ultimate customers. The recurring task of integrating these organizations along a supply chain and coordinating material, information, and financial flows to fulfil the customer demands to improve the competitiveness of a supply chain as a whole is known as supply chain management [47]. Oliver and Webber, in 1982, coined the terms *supply chain* (SC) and *supply chain management* (SCM) and defined SCM as the "process of planning, implementing and controlling the operations of the supply chain with the purpose to satisfy customer requirements as efficiently as possible" [48]. However, SCM lacks a universally accepted definition as the supply chain evolved with time. Other than the firms, external influencing factors redefine the concept of SCM [49]. Initially, firms used to operate locally, and most of the manufacturers owned their own factories - there was no concept of outsourcing. Today, companies are connected to international organizations and are agile enough to handle global impacts, such as Covid 19, global inflation, wars, etc. From managing the flows of materials and information to addressing the international effect, SCM changes its role as it evolves.

As per the "SCOR-Model", a standard for representing, analysing, and configuring supply chain at a high level, SCM consists of five components - planning, sourcing, making, delivering, and returning [50].

1. Planning is the management of balancing resource capacities with demand requirements and the communication of plans across the supply chain. It also covers measuring supply chain performance and managing inventory, assets, and transportation.

2. Sourcing is managing suppliers that procures goods and services to meet demand efficiently and economically.

3. Making is responsible for each action that transforms raw materials into the final product to meet planned and current demand.

4. Delivering is the component that covers all the steps necessary for order management, warehouse management, and reception of products at a customer's location, together with installation. It includes all responsibility to have seamless delivery to consumers, utilizing the freights -road, rail, and air.

5. Returning is the management of post-delivery customer services. It includes returning defective items or excess supply chain products.

Supply Chain Planning (SCP) is an essential aspect of SCM. It is the preparation process for sequencing activities in the supply chain - to answer a question about the next scheduled task on a respective machine, to optimize the delivery of goods, services and information from supplier to customer, and to balance supply and demand. An advanced planning system (APS) is a tool that integrates all the different planning processes in supply chain planning across the other components of the supply chain [51]. It uses solution approaches such as mathematical programming or metaheuristics and associates various planning tasks with supply chain processes. The supply chain planning matrix (SCPM), the underlying structure of APS, categorises planning tasks under two aspects: 1) planning horizon - it classifies the planning into long-term, mid-term, and short-term planning, and 2) supply chain processes - it divides the plan from most upstream to most downstream sectors in the supply chain. Figure 1.10 [52] illustrates various planning tasks along the supply chain. We focus on a "master planning" that deals with medium-term procurement, production, and distribution planning.

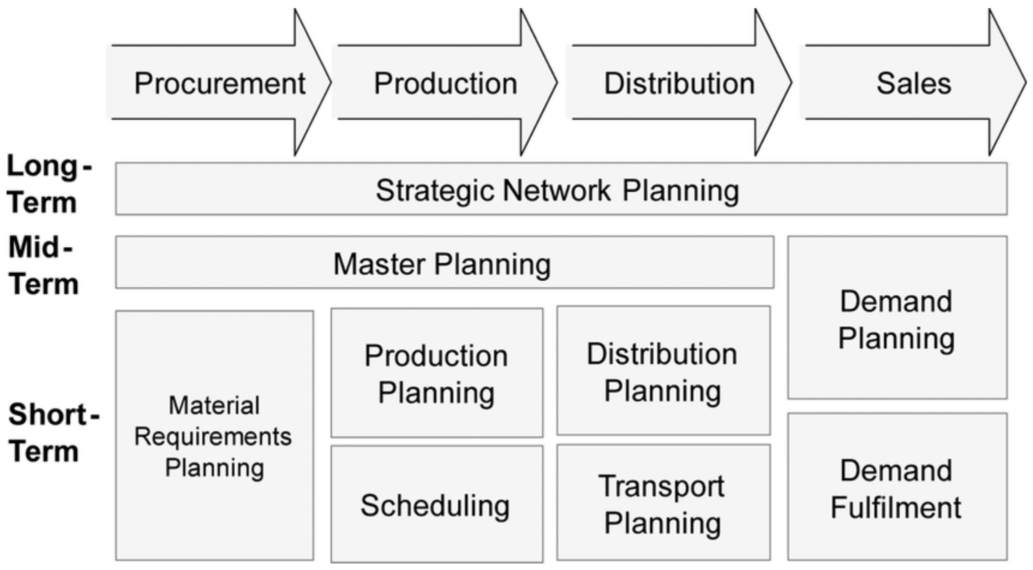


Figure 1.10: Supply Chain Planning Matrix

Master planning looks for the most efficient way to meet demand forecasts and customer requirements. *Association for Supply Chain Management* (APICS)[53] defines master planning as “A group of business processes that includes the following activities: demand management (which includes forecasting and order servicing); production and resource planning; and master scheduling (which includes the master schedule and the rough-cut capacity plan)”. To understand the importance of master planning in the industry, let us think about what will happen if we do not consider master planning. Without a master planner, there will be a lack of coordination between sales and production ends. Suppose production is not aware of the time and amount of bulk orders. In that case, it will produce as per a regular schedule, leading to not meeting the demand or overproduction and thus the inventory overhead. Sudden information about bulk manufacturing of items might not help meet the demand requirement as the raw material would not be readily available for sudden large production. If you cannot meet the customer demand on time, you will lose the customer. Master planning consists of three major segments of supply chain processes 1) demand management, 2) sales and operations plan, and 3) master production schedule. Demand management (DM) manages forecasted and customer demand orders. The previous sales history, order history, and forecasts are scrutinized to the consensus demand forecast. DM is usually a monthly process. If the supply is limited, prioritizing the forecasted consensus demand is also essential. The demand information is then input to sales and operation planning (S&OP) and master production schedule (MPS) segments. S&OP integrates sales, marketing, development, manufacturing, sourcing, and financial plans with the available strategic plan and demand pictures. Its two primary aims are 1) to balance supply and demand through integration between company departments and with suppliers and customers and 2) the alignment of the strategic plan and the operational plan of a company. At a high level, S&OP gives the business the blueprint or a ‘game-plan.’ The responsibility of an MPS is to use this blueprint and the demand details obtained from DM. In this thesis, our contribution towards industry work is on MPS.

1.4.1 Master Production Schedule

A master production schedule (MPS) is a mid-term production plan that lists what the company plans to produce. It expresses the planning for the production of each commodity in specific configurations, quantities, and dates. We refer the reader to [54, 55, 15] for details of the objectives and goals of MPS.

MPS provides a rough-cut capacity plan of what and when needs to be produced based on the required information of quantity and date as input demand list. Other input information includes inventory, production lead time, and resource capacity. The output

MPS obtains the information about the producing items, the quantities available by the due date, the delay in meeting the demands, the resource capacity required, and other information that feeds into a *materials requirements planning* (MRP) schedule. These outputs form many business objectives, such as maximizing demand requirements, minimizing backlog in production, minimizing inventory at various points in the supply chain, minimizing safety stock violations, etc. Some of the industries also cover distribution planning along with production planning. Considering all of them together makes MPS complex, as some conflict with the varied scale of units, and there is a trade-off in optimal value selection. This complexity in MPS has attracted researches to model it to multiobjective optimization problem and solve them with various available methods, such a goal programming [56], multi-objective programming (MOP) [57, 58, 59, 60] and evolutionary algorithms [61, 62, 63, 64]. Some industries set priority among the business objectives considered under MPS in order to make the solving method simpler. A hierarchical-based process, such as the lexicographic procedure, is simple to obtain the Pareto optimal solution for MPS. [59], [60] attempted LM for solving biobjective problem and in multi-objective production planning problem. We focus on explaining a simplistic hierarchical model for a manufacturing firm to understand MPS. LM is used to solve this model.

Mathematical Modeling of Master Production Schedule

For mathematical modeling of MPS in a given supply chain problem, we have the following information:

1. a set of resources with known capacities and a list of operations utilizing the respective resources,
2. a list of on-hand inventory such as raw materials, fixed goods, etc., and the rate of production or consumption of raw material, intermediate and finished goods,
3. a list of customer demand requirements and set of business requirements in the form of cost functions, such as minimizing unmet demand, minimizing inventory, minimizing safety stock requirement violation, etc.,
4. production horizon that consists of discrete-time intervals, each called bucket, and associated parameters such as resource load limit per bucket in the horizon.

With this information set, the planner must optimize each business requirement without any violation of the hierarchy.

Let R and I denote the index sets of resources and inventory items available in the production process. Let O be the index set of operations with subsets $O^k \subseteq O$ that can utilize the resource $r^k \in R$. A known amount 'load_per' is the amount of resource utilized by one unit of operation. Let us consider there is a demand of $\widetilde{d^j_{t_x}}$, where j and t_x denote the corresponding item code and the due date to receive the demand requirement.

Let us define the unknown decision variables $c^r_1, c^r_2, \ldots c^r_{\widetilde{t}}$ to be the amount of resource (associated to each resource $r \in R$) required to process the associated operations at time bucket $t = 1,\ 2,\ 3, \ldots, \widetilde{t}$. Each variable c^r_t is upper bounded by the known amount of resource, maximum capacity($max\_c^r_t$). Similarly, the decision variable $op^i_{j,\ t}$ defines the operation $i \in O^j$ with the resource j utilizes at time bucket t that is needed to produce one unit of product item. Associated to each inventory location $i \in I$ and resource type j, a decision variable $b^i_{j,t}$ defines the amount of inventory carried from time bucket t to $t + 1$. We also define an associated decision variable $xd^j_{t_x}$ that denote the demand (of type j) that could be satisfied over the given due date t_x over the known supply chain settings.

The bucket to bucket planning of the supply chain creates a network structure that helps in posing a network-type mathematical formulation. For a simplistic formulation, assume there is only one resource r that can load three operations $O1, O2$ and $O3$. Here $O^r = \{1,\ 2,\ 3\}$ is the index set of production operations and, $r = 1$ is the resource type. Each operation type consumes raw material (available in infinite amounts) and produces the corresponding finished goods $d1$, $d2$, and $d3$.

We set the planning horizon as a daily bucket window, $t = 1, \ldots, T$ days. We can simplify it by considering load_per, the rate at which an operation consumes a resource, to one. Lead time is set to zero. We set consume_per (produce_per), the rate at which a manufacturing operation consumes (produces) items, to one.

The demand requirements for the finished products are: $\widetilde{d^1_{t1}}$ units of item $d1$ on $t = t1$ day, $\widetilde{d^2_{t2}}$ units of item $d2$ on $t = t2$ day, and $\widetilde{d^3_{t3}}$ units of item $d3$ on $t = t3$ day. The requirement of demands is of equal priority. MPS can be mathematically formulated with demand satisfaction as one of the business requirements as follows:

$$
\begin{aligned}
\text{LP1:}\quad & \text{obj1:= } \min -xd^1_{t1} - xd^2_{t2} - xd^3_{t3} \\
& \text{subject to } \sum_{i \in O^1} op^i_{1,t} - c^1_t \le 0, \quad \text{for all } t = 1,\ 2,\ 3, \ldots, T, \\
& op^i_{1,1} - b^i_{1,1} = 0 \quad \text{for all } i = 1,\ 2,\ 3, \\
& op^i_{1,2} + b^i_{1,1} - b^i_{1,2} = 0 \quad \text{for all } i = 1,\ 2,\ 3, \\
& \qquad \vdots \\
& op^i_{1,ti} + b^i_{1,ti-1} - b^i_{1,ti} - xd^i_{ti} = 0 \quad \text{for all } i = 1,\ 2,\ 3, \\
\text{bound: } & 0 \le xd^i_{ti} \le \widetilde{d^i_{ti}} \quad \text{for all demand item } i = 1,\ 2,\ 3, \\
& 0 \le c^1_t \le max\_c^1_t \quad \text{for all bucket } t = 1,\ 2,\ 3, \ldots, T,
\end{aligned}
$$

$$op^i_{1,t}, b^i_{1,t} \geq 0 \quad \text{for all } i = 1,\ 2,\ 3, \text{ and } t = 1,\ 2,\ 3 \ldots, T. \tag{1.12}$$

LP1 consists of inventory balance constraints that balance the total inflow, total outflow, and inventory carryover of materials at a location and a particular time bucket, and resource load constraints that consider the capacity utilization of resources. The objective function used in LP1 is to minimize unmet demand (equivalent to maximizing demand satisfaction). We also need to consider other key performance indicators (KPIs) required for MPS. Optimizing them simultaneously over the given supply chain constraints is not possible. We solve them in a hierarchy by associating each business requirement with a priority value by modeling it to a h-MOLP. The preference of objectives depends upon the decision of the planner. Generally, minimizing unmet demand is set as the highest priority demand. It is followed by the objective of minimizing *backlog*. The backlog refers to the quantity of those unfulfilled demand orders delivered to the customer after the due date. The objective is to minimize such delays in meeting demands. Decision-makers also consider other objectives for the MPS process, such as minimizing the operation earliness (i.e., reducing early production), minimizing time-based and amount-based safety-stock requirements violations, inventory, and other critical business objectives. The selection of these objective functions and their priorities are industry-specific. Moreover, for large-scale industries, the mid-term period sometimes spans one year, involving hundreds of business objectives. It leads to a h-MOLP formulation with millions of constraints and variables requiring many LP solver calls. In Chapter 4, we discuss MPS with a potato chip industry, a hypothetical model, and devise methods to solve them faster than the existing conventional methods. We also provide the detail of implementing a new similarity-based lexicographic process for large-scale MPS.

1.5 Manufacturing Campaign Planning

Every manufacturing industry has limitations that may restrict products manufactured on demand. Such limitations are resource constraints, financial limitations, inventory-related issues, etc. Here, we deal with a manufacturing system facing resource constraints where specific resources require setups to support multiple operations. Some process industries with heavy set-up times and additional sequence-dependent constraints that produce various products from the same assembly line face a trade-off between inventory and production changeover. Figure 1.11 shows a process industry, for example, a beverage industry.

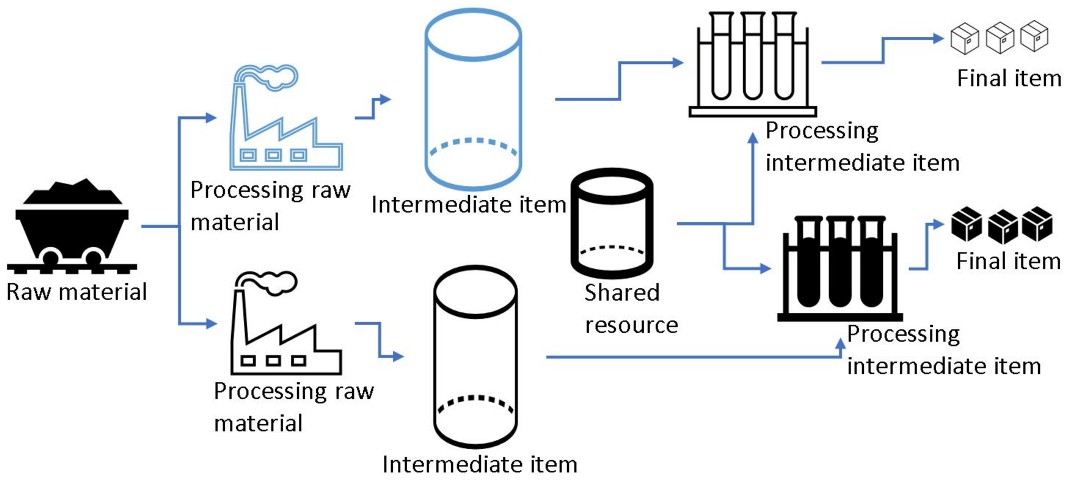


Figure 1.11: Constrained Resource in a Simplistic Supply Chain Model

Two operations that consume intermediate items share a tank, known as a shared resource. Such situations mainly occur in manufacturing systems that produce similar products with minor changes. In the beverage industry, making soft drinks with different flavors needs to clean the tank to remove the previous flavors and add a new flavor. Also, some constrained resources are blending tanks, which mix different types of intermediate products, and can be used only to make one beverage category, for example, a flavor of soft drink. The Setup-change (cleaning) of such tanks from one kind of production to another requires a significant amount of time. Similarly, in the glass manufacturing industry, where the oven can only be used to produce one color of glass at a time, for example, clear, green, or brown, and there can be a significant amount of time required to change from one color to another. In chemical industry, to produce one chemical item for another requires cleaning up the chemical containers. Switching from one type of product to another requires an overhead. We call it *changeover time*. It is time-consuming and costly. It is necessary to manufacture on time to meet the customer requirement of various types of product items timely. However, deciding when to switch the production from one type of product to another is difficult. A frequent switch will incur high changeover time and lead to increased set-up costs, and less frequent will impact customer satisfaction if demand is unmet. Also, with the seldom changeover, producing every type of product beforehand leads to an inventory problem. Such manufacturing problems are known as *campaign problems*, and the computed manufacturing orders based on time or quantity by applying campaign planning to the production process are called the *campaigns*. We need a planning strategy that avoids the campaign problem, an MPS that provides the campaign for production such that:

1. there should not be a degradation in productivity

2. customer demand requirements should not be unsatisfied

3. there should not be an excess inventory

It is essential to consider campaign constraints during planning. In most industries, shared resources that load a similar group of operations consist of complicated changeovers that are sequence-dependent. If we do not consider it during planning, the resulting plan becomes infeasible during scheduling.

Research on campaign planning has always been active and challenging. Many studies related to campaign planning problems could be found in the literature on setup minimization, scheduling in process industries, manufacturing campaign planning, and case studies that discuss the campaign planning challenges in some specific plants [65, 66, 67, 68, 69]. However, we limit ourselves to literature focusing on the mathematical formulation for MPS with campaign planning restrictions. The MILP-based campaign scheduling in a chemical plant is studied in [70]. In this, scheduling is done on short or mid-term production planning in the continuous time frame.

This problem involves making discrete decisions (discussed in the later section) that require a MIP formulation. In most campaign planners, LP can be used as a guide to determine the campaign plan for sets of resources. Advanced production scheduling for batch plants in process industries is the work done by Neumann *et al.* [71] that formulates an MINLP to do the production schedule. Some literature discusses campaign planning with case studies such as food processing [72] and chemical plant [70], which use the heuristic method. C. Suerie describes the model based on a standard lot-sizing of PLSP (proportional lot-sizing and scheduling problem)[73, 74]. Unlike the above models that explicitly use the campaign as a discrete variable and minimize setup time and holding cost separately, [75] by NB Kamath, *et al.* includes campaign planning with MPS heuristically by imposing campaign constraints locally. It proceeds in the following steps: First, it does the production planning, considering all business objectives hierarchically without looking into any violation of campaign planning restriction. This computed planning helps to evaluate the *weighted consumption profile* (WCP), a measure used to set the priority values for each operation. Then, planning violations are avoided by inspecting each bucket by turning assembly operations off or on as per their priorities. We refer to this as the 'heuristic method'. We observe that the heuristic method imposes the campaign constraints at each campaign bucket and then resolves the MPS. Thus for every bucket, there is an MPS run which is computationally expensive. Further, the plan quality obtained is sensitive and relies on weights chosen for the WCP computation, leading to a suboptimal plan.

We propose two methods to address these issues:

1. Improve the ‘heuristic method’ by formulating the campaign planning problem as a sequential decision problem and finding the ideal parameter values using the Cross-entropy method [76]. We call it ‘improved-heuristic’. and,

2. Reformulate the MPS model by incorporating campaign constraints. We call it the “exact-method”. The complexity is that the supply chain constraint set changes from continuous to an integer. The benefits are 1) the model returns a globally optimal solution by a single MILP solver call, and 2) it computes other important KPIs without violating the campaign constraints and avoiding additional modeling effort.

The details of the challenges and the proposed methods are discussed in Chapter 5. A mathematical program with integrality constraints is modeled for the exact method. This mathematical model includes all the campaign planning restrictions of the whole MPS planning horizon. Now we explain a sequential decision problem and the Cross-entropy method used for an improved-heuristic for the MPS with the CP.

1.5.1 Sequential Decision Making

Sequential decision making is a situation where a decision maker’s objective depends on the sequence of decisions. We also call this decision objective a utility or a long-term reward. The sequential decision-making problem (SDP) is the problem of selecting a sequence of actions from a set of sequences of actions to obtain the best possible outcome. Such a sequence is known as a policy. The decision maker’s objective is to compute the best policy. SDP under a certain domain (i.e., with the certainty of actions and rewards) can be solved using some search algorithms. The solution in such a case will be the sequence of steps that leads to an optimal state - for example, solving mixed-integer linear programs using the branch-and-bound method. Here the series of branching decisions leads to the node whose associated LP-relaxation provides the optimal solution. On the other hand, SDP with uncertain domains requires a set of action-state pairs in a sequence that leads to the optimal state. For example, the Markov decision problem (MDP), a stochastic sequential decision model with a *memorylessness* property, is solved using some methods based on reinforcement learning (RL). In our context, we use the term “SDP” for SDP under uncertainty, unless otherwise mentioned. In engineering field, we find them in the optimal control area. In operations research area, it is available as dynamic programming. In neuroscience, we can study SDP as reward systems. In day-to-day life, a chess game is an example of sequential decision making where the decision to

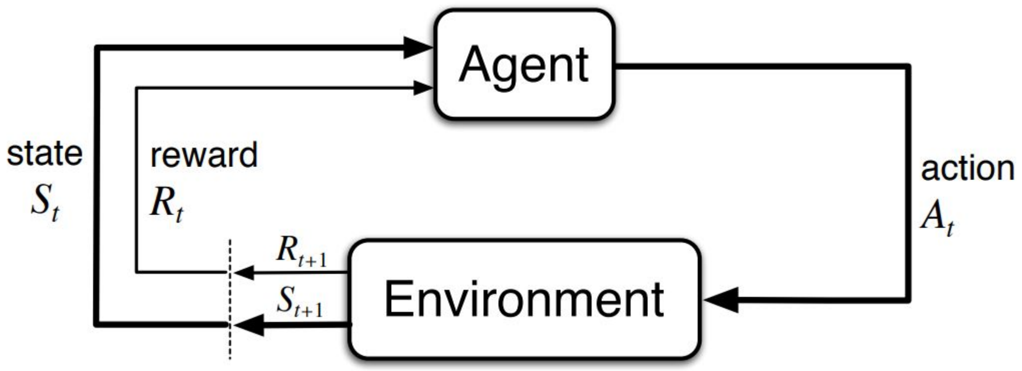


Figure 1.12: Update in a state when an agent interacts with the environment through action (Image taken from [1])

win a match can not be decided by a single action but by a sequence of moves. A player needs to come up with a set of moves that leads him to the win state without sacrificing their more number of pieces. Here, the game's moving from one state to another is associated with probability. Formally, SDP consists of the following components:

1. S – a set of all possible states of the system.

2. A – a set of all possible actions when a system is occupied with a given state.

3. M – a transition probability matrix, consists of the probability of moving to one state from the current state and action taken. In the case of a situation where transition probability is only conditioned over the current state and independent from the current action taken, we can make such transition as state transition probability. We assume that the decision maker's information of M is known.

4. T – a set, the decision time horizon.

5. $R(state, action)$ – a reward function representing an expected reward obtained when the system is currently in a given state, *state*, and taking the action, *action*.

MDP, an SDP model generally studied in RL due to it memorylessness property which says that the current state captures all the information from the past that is relevant in determining the future states and rewards, i.e., $Prob(S_{t+1}|S_t) = Prob(S_{t+1}|S_1, \ldots, S_t)$.

Figure 1.12 depicts the interaction of an agent in the environment. The current state S_t updated to S_{t+1} after action, A_t is taken. This action results in the current reward, R_t. The agent's objective is to maximize a specific reward function (utility) with some pre-decided standard reward criteria such as

1. the utility over the finite horizon - It is the sum of all the expected rewards obtained from updating the state from starting to the state at the last finite known horizon. We can write it as a utility given the initial state m and the final period T, $V(m) = \max_\pi E_\pi \sum_{t=0}^{T} R_t$.

2. the discounted reward function over the infinite horizon - Here, the reward function $= \max_\pi E_\pi \sum_{t=0}^{\infty} R_t \alpha^t$, where α is the discount factor.

3. reward is average, i.e., reward function $= \max_\pi \liminf_{T\to\infty} \frac{1}{T} E_\pi \sum_{t=0}^{T-1} R_t$. Here E_π denotes the expectation over some probability measure induced by a policy π.

If the reward function, R_t, and transition probability matrix, M, are known to an agent, several strategies exist to find the policy that maximizes the return. On the other hand, if an agent is unaware of the model or assumes that information of R_t and M are not (entirely) available, we need to have a learning scheme to obtain the optimal policy that leads to maximum return. Here in our study, we focus on the Cross-entropy method, an evolutionary algorithm to learn the model and obtain the optimal policy.

1.5.2 Cross-Entropy Method

From a biological perspective, the Cross-entropy method (CE) is an evolutionary algorithm where some well-fit individual survives from the population and governs the future generation. From a mathematical context, it is a derivative-free optimization, where the problem/model is considered a black box, i.e., we do not have any information about the model. We know that the black box takes some input and provides output without detailed mathematical processing.

CE has a wide range of applications. The literature survey on the application of CE range from the application of machine learning (ML) and the heuristics based on combinatorial optimization, such as policy learning [77, 76], black box optimization for TSP, Maxcut problems, etc., to the industries, such as facility layout [78], buffer allocation[79], manufacturing planning[80, 81], etc. The CE involves an iterative procedure. Each iteration comprises two phases - generating a random data sample using some distribution rule and updating the distribution rule's parameters to produce a "better" sample in the next iteration. Steps followed in the CE in mentioned in Algorithm 2.

At any time step t, given a state S_t and a set of possible actions A_{S_t}, the controller has to take action $a_t^* \in A_{S_t}$ such that the total reward i.e., total score achieved at the termination stage be maximized. In other words, we must have a "good" policy to achieve this total reward. Given a state S_t and an action set A_{S_t} as an input, for each time step t, from start

Algorithm 2: Cross-entropy Method

```
Initialize: Choose initial parameters - mean $\mu_{0i}$ and standard deviation $\sigma_{0i}$, for
  individuals $w_i$, corresponding to the weight vector of size $s$:
  $W_0 = (w_{01}, \dots, w_{0s})$. Set iteration number $k = 0$ and the total number of
  iterations = maxItr.
  Step 1: Generate $n$ random sample vectors $X_j = (X_{j1}, \dots, X_{js}),\ j = 1, \dots, n$
  using normal sample distribution with parameter vectors $(\mu_{k1}, \dots, \mu_{ks})$ and
  $(\sigma_{k1}, \dots, \sigma_{ks})$.
  for every elements in a vector using normal sample distribution with
  parameter vectors $(\mu_{k1}, \mu_{k2}, \dots, \mu_{ks})$ and $(\sigma_{k1}, \sigma_{k2}, \dots, \sigma_{ks})$.
  Step 2: For each generated sample as an input weight vector, use some policy
  defined by the decision maker, based on evaluation function, and compute the
  utility value $Out_j\ \forall j = 1, \dots, n$.
  Step 3: Sort sample vectors by generated output values (in descending order).
  Assign the top output value as $OutTop_k$.
  Step 4: if $k > itrn$ or output value starts converging then
    Exit with $W_k = OutTop_k$ and corresponding input weight vector.
else
    $k = k + 1$.
end
Step 5: Choose top $m$ sample vectors and evaluate mean and standard deviation:

  $$\mu_{ki} = \frac{1}{m}\sum_{j=1}^{m} X_{ji} \text{ and } \sigma_{ki} = \sqrt{\frac{1}{m}\sum_{j=0}^{m}(\mu_{ki} - X_{ji})^2}\ \forall i = 1, \dots, s.$$

  and return to Step 1.
```

till the termination, we obtain the following action:

$$a_t^* = \arg\max_{a_i \in A_{S_t}} \{\mathrm{Exp}(\mathrm{Reward}(S_t,\ a_i))\}.$$

Here $(S_t,\ a_i)$ is a state obtained by taking an action a_i on state S_t.

a_t^* can be extended as,

$$a_t^* = \arg\max_{a_i \in A_{S_t}} \sum_{\left\{S_j \in \text{set of possible states after } (S_t,\ a_i)\right\}} Prob((S_t,\ a_i),\ S_j) \times \mathrm{Reward}(S_j).$$

where $Prob((S_t,\ a_i),\ S_j)$ is the transition probability from $(S_t,\ a_i)$ to S_j and $\mathrm{Reward}(S_j)$ is maximum evaluation-value of the possible next state, i.e.,

$$\mathrm{Reward}(S_j) = \max_{a_q \in A_{S_j}} \{Eval(S_j,\ a_q)\}.$$

Evaluation function: The Evaluation function has the form :

$$Eval = w_1 \times f_1 + w_2 \times f_2 + w_3 \times f_3 \ldots + w_n \times f_n,$$

where $Eval$, gives evaluation-values of the given configuration of the states. It is a linear combinations of the features $(f_1, f_2, f_3, \ldots, f_n)$ weighted by the coefficients $(w_1, w_2, w_3, \ldots, w_n)$. Note that each feature defines "goodness" of the configuration of a state and corresponding weights.

The 'heuristic method' generates and solves a sequence of linear models. We can formulate it as an SDP. The planner (the agent), at each time bucket (the decision epoch), takes an effective campaign decision (the action). The decision to stop some running operations and start the idle process changes the configuration of the campaign from one state to another. A set of key performance indicators (KPIs) can form an element of a suitable utility function for a good campaign decision criteria. A MIP and SDP formulations of the MPS with campaign planning constraints using the CE method are discussed in detail in Chapter 5. We find the features and corresponding weights required in CE for policy learning for the CP problem. Further, we discuss a case study focusing on the industrial outlook and challenges of a tire manufacturing industry.

1.6 Outline of Thesis and Contributions

We start with the literature survey of some popular variable-branching procedures and highlight the challenges in reliability branching, the state-of-the-art branching rule, in a branch-and-bound algorithm, in Chapter 2. We exploit a concept known as "similarity between nodes" that addresses the significant challenges that popular branching rules face when using strong branching, and introduce a new branching procedure. Its connection with other branching rules is also discussed. Further, computational results on benchmark instances are presented and compared with the default branching scheme of a solver, Coin-or branch-and-cut (CBC). In Chapter 3, we obtain some theoretical developments related to lexicographic methods. We highlight the challenges and present a new lexicographic method, a variant of the variable-fixing rule, that exploits the structure of the underlying hierarchical model, a h-MOLP. Further, we provide some computational experiments and compare them with other methods. Chapters 4 and 5 provide our contribution to the industry wherein the master production schedule posed to the h-MOLP model. In Chapter 4, along with modeling the MPS, we investigate the procedure to solve this model efficiently by improving the existing methods and using the new lexicographic discussed in Chapter 3. In Chapter 5, we discuss the extension of work on MPS, discussed in Chapter 4, where we address the challenges of enhancing the mathematical modeling of MPS that also considers the campaign planning problem, which we generally encounter in some process industries. We do a case study on the importance of campaign planning in one of the tire manufacturing industries. We summarize the salient contributions of this thesis:

1. We show the limitations of reliability Branching, the state-of-the-art branching procedure in a branch-and-bound algorithm. We define a concept, "similarity between nodes", to find the fraction of similarity among a sequence of 'easy' subproblems solved to obtain the solution of the 'difficult' MIP problem using a branch-and-bound algorithm.

2. Using the concept of "similarity", we develop a new branching rule, "SimBranch", that addresses one of the significant issues in reliability branching. We also obtain its connection with other popular branching schemes. Detailed computational results are obtained on benchmark MIP instances by implementing SimBranch on CBC, an open-source MIP solver, and results are compared with the default branching rule in CBC.

3. We show the theoretical results on the equivalence between the variable-fixing rule and the constraint-addition, two popular LMs for a h-MOLP. We define a concept, "similarity between LPs" to find the fraction of similarity among 'easy' LP problems solved to obtain the solution of the 'difficult' h-MOLP with LMs. With this concept, we develop a new LM, "SimLex", that effectively exploits reoptimization between LPs for faster solution of the h-MOLP and show its computational effectiveness over existing LMs on some available benchmark problems.

4. We formulate MPS to h-MOLP and obtain some techniques to speed up the existing LMs for its solution. The supply chain of some consumer products and goods industries is studied to obtain the MPS and find the impact of preferring 'SimLex' over other LMS.

5. An MPS that respects campaign planning constraints are mapped as a sequential decision-making problem. An evolutionary method, the Cross-entropy method, is used to solve it.

6. We provide a mathematical model for MPS that respects campaign planning constraints for large-scale industries. The MPS of the tire manufacturing industries are obtained using this model and compared with the existing method.

7. We perform a detailed case study of the importance of campaign planning in one of the tire manufacturing industries and highlight managers' and planners' viewpoints on it.

Chapter 2

Similarity-based Branching for Integer Optimization

2.1 Introduction

We have introduced a branching scheme in the branch-and-bound algorithm (B&B), with an example in the previous chapter. In this chapter, we detail some branching procedures and investigate the issues reliability branching, one of the variable branching schemes, faces in solving mixed-integer linear programs (MILP). One of the issues we mainly focus on is the unnecessary use of strong branching calls at nodes in the branching process. The strong branching simulates the change in the lower bound by solving two LP-relaxations for each branching candidate that result in fewer nodes in the branch-and-bound tree (B&B tree). Achterberg, in his research [82], finds that it results in 65% fewer search tree nodes on average, compared to the state-of-the-art hybrid branching strategy, but with the expense of an increase of up to 44% in computation time. He introduces the concept of a *reliable* candidate and develops a new hybrid rule, known as reliability branching, by combining pseudocosts branching and strong branching techniques. However, reliability branching also suffers from the unnecessary use of strong branching calls, which we will discuss in Section 2.3.

The proper use of strong branching calls to improve branching schemes motivates us to devise the concept of 'similarity' between the current node and the nodes already explored in the tree. It defines a similarity measure between nodes computed using relevant features of the relaxation like bounds on variables. Using the information from *similar* nodes, we estimate the change in the objective value for each branching candidate, much like reliability branching, to select the variable to branch on. The idea develops into a new branching procedure that effectively exploits the information generated from explored nodes. We call it *SimBranch*. It tries only a few strong branching calls placed strategically in a few search tree nodes and only uses the collected strong branching information when relevant. SimBranch is generic - we can relate it with other popular branching procedures, such as strong branching, reliability branching, and pseudocost branching. Later in this chapter, we show that this similarity-based strategic decision of using relevant strong branching calls can improve the speed of solving various benchmark MILPs. We recall problem (1.2), an MILP defined in Chapter 1:

$$\begin{aligned} \textbf{MILP} : & \underset{x}{\text{minimize}}\ c^{\mathrm{T}}x \\ & \text{subject to}\ \ Y := \{x \in \mathbb{R}^n \mid Ax \leq b,\ x_i \in \mathbb{Z},\ i \in I\}. \end{aligned} \tag{2.1}$$

For the sake of explanation of the SimBranch procedure, we assume that all the integer decision variables in the MILP are binary, i.e., $x_i \in \{0,1\},\ i \in I$. It forms the following mixed binary program (MBP):

$$\begin{aligned} \textbf{MBP} : & \underset{x}{\text{minimize}}\ c^{\mathrm{T}}x \\ & \text{subject to}\ \ B := \{x \in \mathbb{R}^n \mid Ax \leq b,\ x_i \in \{0,1\},\ i \in I\}, \end{aligned} \tag{2.2}$$

Here B is the feasible set to the MBP. Other parameters used are the same as in problem (2.1). The idea of similarity can be extended for general integer cases and possibly other programs, like MINLP and CSP.

The remainder of the chapter is organized as follows. We provide a literature survey of some popular branching procedures in Section 2.2. Section 2.3 discusses the issues in reliability branching. The main idea of SimBranch is explained in Section 2.4. Formal notation, the algorithm and parameter selection guidelines are presented in Section 2.5, and the computational results on benchmark instances and summary of the work in Section 2.6.

2.2 Branching Schemes

The objective of an ideal branching mechanism is to minimize the number of subproblems (nodes) in the B&B that need to be evaluated. Also, the method should not be costly as there might be a situation where the time to solve a problem is more to minimize the evaluation of the total node. Its primary purpose should be to divide the feasible regions Y of the MIP (2.1) into $k,\ k \geq 2$ subregions, $S_1,\ S_2,\ldots,S_k$ such that

$$Y = \cup_{i=1}^{k} S_i. \tag{2.3}$$

The subregions will further create k MIPs, each with the same objective function of the problem (2.1) but with different set of feasible points $S_1, S_2, \ldots, S_k$. In this thesis, we focus on a variable branching scheme that forms two subproblems after dividing the feasible region by branching on the fractional variable. Branching on a variable is a simple way to divide the feasible area that satisfies equation (2.3). In some cases, dividing the problem into more than two subproblems is effective [83, 84], but most procedures employ splitting into only two subproblems by variable-based branching.

Let $\widetilde{x}$ and z^k be the optimal solution and optimal value to the current LP-relaxation of a subproblem MIP^k, an MILP associated with the node N^k at iteration k in the B&B for solving the program (2.1). Consider C^k be the subset of I, an index set of variables constrained to be integers, defined as:

$$C^k = \{i \in I \mid \widetilde{x}_i \notin \mathbb{Z}\}.$$

A variable branching scheme that branches on $x_i \in C^k$ generates subproblems, MIP_i^{k-} and MIP_i^{k+}, by adding two trivial inequalities, $x_i \leq \lfloor \widetilde{x}_i \rfloor$ and $x_i \geq \lceil \widetilde{x}_i \rceil$ respectively, in the constraint set of MIP^k (1.2.2). We represent subproblems MIP_i^{k-} and MIP_i^{k+} by the nodes N_i^{k-} and N_i^{k+}, respectively, the left and right-side children of N^k in the tree. Let z_i^{k-} and z_i^{k+} be the optimal objective values of the LP-relaxation of MIP_i^{k-} and MIP_i^{k+}. This information at N^k, will be used to calculate the branching score needed to find an ideal variable to branch on.

The notations we referred to for components and parameters in the variable branching process above will later help explain some well-known branching methods. In addition, we will also use the following terminology:

- An MILP is assumed to be of the form (2.1) and the mixed binary program (MBP) will be of the form (2.2).

- Candidate set: Set C^k, the set of fractional variables that participates in the branching variable selection, we call it a *candidate set* at node N^k.

- Up-fractionality and Down-fractionality: For a candidate $i \in C^k$, *up-franctionality*, and *down-fractionality* are defined as:

$$f_i^+ := \lceil \widetilde{x}_i \rceil - \widetilde{x}_i \text{ and } f_i^- := \lfloor \widetilde{x}_i \rfloor - \widetilde{x}_i. \tag{2.4}$$

- Down-child and Up-child of N^k: N_i^{k-} and N_i^{k+}, the left and right child created after branching on $x_i \in C^k$ at the node N^k, are called *down-child* and *up-child*.

- Relaxed-objective value at N^k: z^k, the optimal objective value of the relaxation of MIP^k will be referred as *relaxed-objective value* at node N^k.
- Relaxed-down objective and Relaxed-up objective values at N^k: z_i^{k-} and z_i^{k+}, the optimal objective values of the relaxation of MIP_i^{k-} and MIP_i^{k+} will be called as *relaxed-down* and *relaxed-up* objective values at N^k.
- Branching node: A node N^k in the tree which calls branching operation.
- Branching variable or Ideal candidate at N^k: A candidate x_i chosen for branching from the candidate set C^k, using a given branching scheme at a node N^k, is called a *branching variable* or an *ideal candidate* of that node.

2.2.1 Schemes based on Pseudocosts

The popular and highly used measure in selecting the branching variable is the estimation of an average objective gain, known as *pseudocosts*. This section defines pseudocosts and describes the branch schemes based on it.

Pseudocost Branching

Pseudocosts branching is the scheme based on the scores that estimate the average objective gain of branching candidates. We call it a *pseudocost* score. Consider we are solving an MILP using B&B and, at iteration k, we have processed the node N^k. Assume the candidate set and the relaxed-objective value at N^k are C^k and z^k, respectively. Also, given the branching variable x_i, the relaxed-down and relaxed-up objective values are z_i^{k-} and z_i^{k+}, respectively.

Using the above information at N^k, we define the unit objective gain for down-child and up-child as follows:

$$\Delta_i^{k-} = \frac{z_i^{k-} - z^k}{f_i^-} \text{ and } \Delta_i^{k+} = \frac{z_i^{k+} - z^k}{f_i^+}. \tag{2.5}$$

At node $N^t, t > k$ with available candidate set C^t, we compute pseudocosts, the *down score* and the *up score*, for each branching candidate $x_i,\ i \in C^t$ as follows:

$$\Psi_i^{t-} = \frac{\sum_{k\in\{j|x_i\in C^j\}} \Delta_i^{k-}}{\eta_i^{t-}} \text{ and } \Psi_i^{t+} = \frac{\sum_{k\in\{j|x_i\in C^j\}} \Delta_i^{k+}}{\eta_i^{t+}}. \tag{2.6}$$

Here η_i^{t-} is the number of nodes processed, up to iteration t, whose 1) selected branching candidate was x_i, and 2) down-child provided a feasible solution. Analogously we define

η_i^{t+} where up-child provided a feasible solution. C^j is the candidate set at branching node $N^j, 1 \leq j < t$.

Pseudocost branching uses Ψ_i^{t-} and Ψ_i^{t+}, pseudocosts scores to select the branching variable as follows:

$$i^* = \arg\max_{i \in C^t}\{s_i := W(\Psi_i^{t-}, f_i^-, \Psi_i^{t+}, f_i^+)\}. \tag{2.7}$$

Here s_i is a score associated with the candidate x_i, defined as a function that takes pseudocosts and fractionality scores as input. Beninchou *et al.* [19] provided the following score function to choose the branching variable x_{i^*}:

$$s_i = \min\{\Psi_i^{t-} \cdot f_i^-, \Psi_i^{t+} \cdot f_i^+\}. \tag{2.8}$$

Later Linderoth *et al.* [85] suggested another selection rule, a linear approach with the use of the parameter $\alpha \in [0, 1]$ as

$$s_i = \alpha \cdot \min\{\Psi_i^{t-} \cdot f_i^-, \Psi_i^{t+} \cdot f_i^+\} + (1 - \alpha) \cdot \max\{\Psi_i^{t-} \cdot f_i^-, \Psi_i^{t+} \cdot f_i^+\} \tag{2.9}$$

Achterberg [82] proposed the following product-based rule with $\epsilon \in [0, 1]$:

$$s_i = \max\{\Psi_i^{t-}, \epsilon\} \cdot \max\{\Psi_i^{t+}, \epsilon\} \tag{2.10}$$

This rule, with the setting of $\epsilon = 0.6$ outperforms the linear approach by 14% even with the best possible tunning of α.

One of the difficulties with pseudocost branching is that it does not have any information about the past branching decisions at the beginning of the algorithm. The pseudocosts associated to candidates must be initialized at the early nodes before using it to compute the branching variable using equation (2.7) in somewhere later nodes for the branching decision. One basic setting rule is to initialize $\Psi_i^{t-} = 1$, if all the down scores are uninitialized, i.e., there is no x_i such that $\eta_i^{t-} > 0$. If some of them are initialized with some scores, the uninitialized Ψ_i^{t-} will be equal to the average of that scores. We similarly initialize Ψ_i^{t+}.

The branching decisions at the early nodes in the B&B tree have enormous impacts on the tree's structure and the underlying subproblems. Gauthier *et al.* [19] initially studied it. The mechanism of initializing variables with uninitialized pseudocosts using strong branching was later developed by Linderoth *et al.* [85]. It is a hybrid approach that uses strong branching calls at the top nodes, at some fixed depth, and then to switches to pseudocost branching. Achterberg improved this idea and introduced a term 'reliability' that used strong branching with pseudocost branching in more dynamic way.

Strong Branching

Applegate *et al.*[86] developed the idea of *strong branching* for solving the traveling salesman problem. (We refer [87] to the reader for the problems and methods on traveling salesman problem). In the process of selecting an ideal candidate to branch on, a strong branching method first evaluates the LP-relaxations of the subproblems to the child nodes to the current processing node with an MILP associated with it. The splitting of the current problem into subproblems is done on some selected candidates from candidates set by variable branching procedure.(see Section 1.2.2). The computed LP-relaxations with respect to the branching candidate evaluates the improvement in dual bounds. A candidate that returns the most improvement is selected as the ideal branching candidate. Suppose, at iteration k, we have processed node N^k. Assume the candidate set and the relaxed-objective value at N^k are C^k and z^k, respectively. For a candidate x_i, we compute relaxed-down and relaxed-up objective values, z_i^{k-} and z_i^{k+}, respectively, by solving LP-relaxations on the child nodes of N^k. Using objective gain, defined in equation (2.5), we compute SB_i, a *strong branching score*, as follows:

$$SB_i = \max\{\Delta_i^-, \epsilon\} \cdot \max\{\Delta_i^+, \epsilon\}. \tag{2.11}$$

The candidate x_{i^*} is selected as a branching variable with maximum strong branching score, that is, $i^* = \arg\max_i\{SB_i\}$.

If the strong branching score is computed for all the candidates in candidate set C^k, given the score computation function, we can obtain the best branching variable locally.This strategy, known as *full strong branching*, is computationally costly as strong branching explicitly solves two (up and down child) LP-relaxations for all the candidates in C^k.

We can speed up the full strong branching by limiting the number of simplex iterations in the LP-relaxation solves. Another way to improve the computation overhead is by selecting a subset of C^k as branching candidates. Most solvers [27, 88, 89, 26] provide the parameters to tune the simplex iterations, and number of candidates to participate for strong branching.

Reliability Branching

Achterberg *et al.* [22] improved the idea of [85] by dynamically using strong branching with pseudocost branching and developed a scheme known as *reliability branching*. It is considered to be the state-of-the-art branching rule for almost all MIP/MINLP solvers. Reliability branching combines strong branching with pseudocost branching by using strong branching for the variables with uninitialized pseudocosts and those set to *unreliable*. A variable is *reliable* if it calls more than a *fixed* number of strong branching. Otherwise, it is unreliable. This fixed number is called a *threshold value*.

Consider the relaxed-objective value z^k and the candidate set C^k at N^k are available to us. Suppose N_i^{k-} and N_i^{k+} are down and up child nodes of N^k, generated by branching on the candidate $x_i,\ i \in C^k$. The corresponding relaxed-objective values are z_i^{k-} and z_i^{k+} obtained after solving LP-relaxation with a maximum α number of dual simplex iterations. We first compute score s_i, for each $x_i, i \in C^k$ and pick the γ number of most scoring candidates from C^k for the branching decision. The parameter γ controls the maximum strong branching calls per node. For a candidate x_i, we check $\min(\eta_i^-,\ \eta_i^+) < \lambda$, a condition that ensures this strategy does not call strong branching on reliable candidate. Down and up objective gains are computed by solving two LP-relaxation, one at each child node. Besides γ and λ, a parameter α controls the computational overhead by limiting the dual simplex iterations in solving of LP-relaxation of the MIPs associated with child nodes. Using down and up gains, we compute the pseudocost score of all the chosen candidates and pick the candidate with the best pseudocost score. One more parameter, β, a lookahead value, is used to stop the evaluation if no new best candidate appears for the beta number of successful candidates. Algorithm 3 mentions the steps in reliability branching at node N^k.

We provide more insights on reliability branching in Section 2.3.

Lookahead Branching

Lookahead Branching [90] selects the branching variable by predicting the lower bounds of grandchild nodes. It studies the impact of the current branching decision on the bounds of grandchild nodes of the current node. Given a node N^t at iteration t with the relaxed-objective value z and candidate set C^t. Let us consider N_i^{t-} and N_i^{t+} are its down and up child nodes with the relaxed-objective values, z_i^- and z_i^+, respectively. Further assume that N_{ij}^{t--} and N_{ij}^{t-+} are the down-child and up-child with relaxed-objective values, z_{ij}^{--} and z_{ij}^{-+} respectively, evaluated by branching the node N_i^{t-} at $x_j, j \in C_i^{t-}$. Similarly we have objective values, z_{ik}^{+-} and z_{ik}^{++} by branching the node N_i^{t+} at $x_k, k \in C_i^{t+}$. Here C_i^{t-} and C_i^{t+} are the candidate sets at down-child and down-child of N^t, respectively. We depicts this two level deep branching with notation in Figure 2.1.

Along with z_{ij}^{--}, z_{ij}^{-+}, associated with N_i^{t-}, we use indicator parameters $\rho_{ij}^{--}, \rho_{ij}^{-+}$. They are set to 1 if the corresponding nodes, N_{ij}^{t--} and N_{ij}^{t-+}, would be pruned. Similarly we have

Algorithm 3: Reliability Branching

```
Initialize: $i = 0,\ l = 1$, and $s^*_{prev} = -\infty$
 Step I: Calculate pseudocost score, $s_j$ (using 2.7) for each $j \in C^k$.
 Step II: $F$ := a list consists of candidates from the sorted $C^k$ in non-increasing
 order of their pseudocost scores. Assume $len(F)$ is the size of $F$.
 while $i \leq \min(\gamma,\ len(F))$ and $\min(\eta^-_j,\ \eta^+_j) < \lambda$ do
     1. Compute $\Delta^{k-}_{F(i)}$ and $\Delta^{k+}_{F(i)}$, down and up objective gains (using the
        expression 2.5). /* $F(i)$ is the $i^{th}$ element in $F$ */
     2. Update $\Psi^{t-}_{F(i)}$ and $\Psi^{t+}_{F(i)}$, down and up pseudocosts (2.6) using the down
       and up gain scores ;
     3. Update pseudocost score $s_{F(i)}$. Compute $s^* = \max_{t \in F} s_t$.
       if $s^*_{prev} < s^*$ then
           $s^*_{prev} = s^*$ and $l = 1$.
     else
         if $l < \beta$ then
             Set $l = l + 1$.
         else
             Go to Step III.
         end
     end
end
Step III: Return with $F(i)$.
```

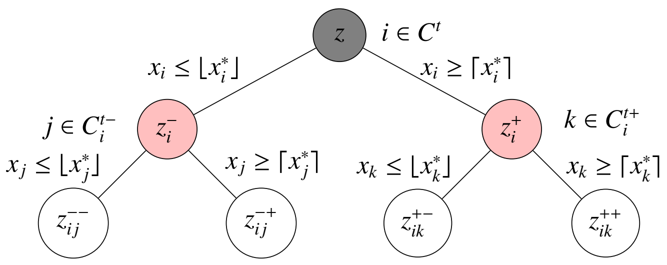

z
$i \in C^t$
$x_i \leq \lfloor x_i^* \rfloor$
$x_i \geq \lceil x_i^* \rceil$
$j \in C_i^{t-}$
z_i^-
z_i^+
$k \in C_i^{t+}$
$x_j \leq \lfloor x_j^* \rfloor$
$x_j \geq \lceil x_j^* \rceil$
$x_k \leq \lfloor x_k^* \rfloor$
$x_k \geq \lceil x_k^* \rceil$
z_{ij}^{--}
z_{ij}^{-+}
z_{ik}^{+-}
z_{ik}^{++}

Figure 2.1: Child and grandchild nodes of a node in B&B tree

$\rho_{ik}^{+-}, \rho_{ik}^{++}$ for the nodes N_{ik}^{t--} and N_{ik}^{t-+}. A weighting function,

$$s_i = W(z, z_{ij}^{--},\ z_{ij}^{-+},\ z_{ik}^{+-},\ z_{ik}^{-+},\ \rho_{ij}^{--},\ \rho_{ij}^{-+},\ \rho_{ik}^{+-},\ \rho_{ik}^{++})$$

uses the computed objective values and the indicator parameter values and returns a score s_i. The x_i with maximum score is chosen to be an ideal candidate to branch on. We refer [90] for the W function used for the score computation where the lookahead score computed from the weighting function considers the objectives of minimizing the number of grandchild nodes that are created and decrease the LP-relaxation bounds at the grandchild nodes as much as possible.

Lookahead branching is even more expensive than strong branching as instead of solving two LP-relaxations associated to the down and up child nodes, it requires solving many LP-relaxations, each associated with one grandchild. The computational result shows that the idea can often significantly reduce the number of nodes in the search tree. However, calling strong branching information at grandchild nodes leads to computational overhead.

Hybrid Branching

Hybrid branching is a general branching rule that combines different branching schemes to form an efficient mechanism, resulting in lesser node evaluation and improved computation time in the B&B tree. Reliability branching is one of the examples of hybrid branching that combines pseudocosts branching and strong branching using *reliability* that dynamically use strong branching calls. Another hybrid scheme combines inference branching (which will discuss later in this section) with reliability branching. The recent hybrid scheme [91] is a default branching rule in the MILP solver, SCIP [89]. It combines three branching techniques:

1. Inference branching for solving constraint satisfaction problems (CSP)
2. Variable state independent decaying sum [92]used for solving satisfiability problems (SAT)

3. Reliability branching for solving MIP

Variance-based Branching

Gregor Hendel [93] highlights the issue of sample variance in reliable scores in reliability branching. He introduces a variance-based reliability scheme that updates the pseudocost scores by using the sample variance of the past observation of each candidate. A subset of variables with high variance is considered unreliable. These are the potential candidate for "strong-branching." A subset of variables with low variance is considered unreliable. They will use the pseudocosts score in the computation of the branching variable. The method implemented in SCIP shows a promising result for effectively reducing the size of branch-and-bound trees compared to the reliability branching, especially for large trees.

2.2.2 Some More Branching Rules

Random Branching

Random branching is the most basic branching scheme that does not use any information from the LP-relaxation. It randomly selects a branching candidate from the candidate set and does branching by creating the explicit bounds on it.

Most Infeasible Branching and Least Infeasible Branching

Most infeasible branching chooses a variable from the branching candidate whose fractional part is close to 0.5. In contrast, the least infeasible branching determines the variable x, whose fractional portion is closed to 0. Both the rules are rare in use - like random branching, they yield an inferior performance.

Inference Branching

This branching rule, applied to MIP, is mainly inspired by the branching mechanism used in the solvers (for example, SATZ [94]) that solve SAT/CSP instances where no objective function is available. Branching decisions based on LP-relaxation of the objective value is useless for such cases. For such a situation, one idea is to select a branching variable that produces a more significant number of deductions in other variables after fixing a value to it. Let us consider an instance with the following set of constraints:

$$x_1 + x_2 = 1,$$
$$x_1 + x_3 + x_4, \leq 1$$
$$-x_1 + z \geq 3.$$

If we set $x_1 = 0$, then the domain deduction on other variables is inferred to $x_2 = 1,\ z \geq 3$. So the number of deductions is 2. Similarly, if $x_1 = 1$, it implies $x_2 = x_3 = x_4 = 0$ and $z \geq 4$. In that case, the number of inference deductions is 4. Similar to pseudocosts, an inference value to a given variable is evaluated using the (previous and current) information of the number of domain deductions on other variables defined (in [95]) as:

For up and down score on variable $x_i,\ i \in C$, we have:

$$\Phi_i^+ = \frac{\varphi_i^+}{\nu_i^+} \text{ and } \Phi_i^- = \frac{\varphi_i^-}{\nu_i^-}.$$

Where φ_i^+ and ν_i^+ are the total number of inference deductions and the number of corresponding subproblems on which domain propagation has already applied with respect to variable x_i. Domain propagation refers to the task of tightening the domains of variables by inspecting the constraints and the current domains of other variables at a local subproblem in the search tree. Similar to pseudocost branching, inference branching also suffers from the problem of initializing the up and down scores. Like pseudocosts branching uses strong branching for initializing variables with no initial pseudocost score, inference branching uses presolving techniques such as *probing* on constraints. For presolving in MIP, we refer to [96] and [82, Chapter 10].

Orbital Branching

Orbital branching is a branching method in B&B to solve the MIPs containing a great deal of symmetry. Instead of using a single variable to branch on, it finds a group of variables called *orbits*. In their work on orbital branching, J. Ostrowski, *et al.* developed the idea of using orbit to solve covering and packing problems and implemented it in the solver MINTO [97]. The method is limited to deal with the problem containing structured symmetric groups. Consider an example where O be an index set containing indices of the group (orbit) of variables. We can have the branching: $\sum_{i \in O} x_i \geq 1$ or $\sum_{i \in O} x_i \leq 0$. Now, if at least one of the variables is set to one and all are equivalent, we can pick any ($t \in O$) variable arbitrary, i.e., $x_t = 1$ or $\sum_{i \in O} x_i = 0$. In other way, if x_t is chosen as a branching variable, the nodes corresponding to $x_j, j \in O - \{t\}$ will be prune nodes.

Backdoor Branching

In the context of branching mechanism, a *backdoor* is a set of branching variables whose integrality is enough to guarantee the optimal solution value to the MIP. Backdoor branching [98] iteratively uses the idea of a backdoor. It makes a sequence of short enumeration runs in the “sampling mode”. (See sampling mode in [99]). At each sampling run, it solves a set covering model and collects backdoor, a small cardinality set of branching variables “covering” all fractional solutions in the current list. A MIP solver is then called to solve the model by choosing the covered variables as the highest priority variable for branching. It will provide a low-cost fraction solution. After the specific iteration, we solve the final run considering MIP solver as a black box where, without looking into any criteria, we set branching priority 1 to all the variables in the solution of the last set-covering problem and 0 to the remaining ones. Backdoor branching is compared with the default procedure in the state-of-art MIP solver IBM ILOG Cplex 12.2 [100]. The result shows that backdoor branching performs better on geometric mean average (if we do not consider the sampling time) over on some specific instances taken from MIPLIB2003 [101], MIPLIB2010 [102] and COR@L [103], libraries of benchmark instances.

Nonchimerical Branching

Nonchimerical branching [104] improves strong branching by selectively using branching candidate set. It only keeps the variables with *nonchimerical* fractionalities. It iteratively removes the variables with chimerical fractionalities (the fractionality whose impact is low in the LP solution) and ends up providing the candidate set with nonchimerical fractionalities that significantly impact the objective function by rounding them up or down. The strategy is implemented in the solver IBM ILOG Cplex 12.2 using callback functions and compared with full strong branching and hybrid branching, with a specific number of strong branching, on the benchmark instances considered in [22, 99] and chosen from the MIPLIB2010. In specific settings, this strategy shows a good improvement.

Cloud Branching

Cloud branching [105] introduces a novel method to use the cloud of the LP-relaxation. The cloud is a multiple alternative optimal (relaxation) solution. The branching method exploits the information such as dual degeneracy from the cloud that can enhance branching rules, such as strong branching and pseudocost branching. The cloud branching, implemented with full-strong branching, showed an encouraging reduction of the mean run time than the default full strong branching on the standard MIP test sets on SCIP solver [106] and opens the opportunity for other cloud-based branching rules.

Abstract Model-based Branching

Pierre Le Bodic and George Nemhauser present the first theoretical model for selecting branching variables in their work on branching [107]. It introduces a polynomially solvable decision problem, called *single variable branching* (SVP), to study the model that defines a simplistic B&B. The existing score functions which evaluate the pseudocosts score for the branching candidates are discussed and verified with SVB that the score functions used for pseudocosts computation are imperfect. Further, a Multiple Variable Branching problem (MVP), the extension of the SVB problem, is introduced. These problems help analyze the model, present new scoring functions that analytically estimate the dual bound improvement, and select the variable for branching. The abstract-based model proves its efficiency in both simulated experiments and MIP instances.

ML-based branching

Some branching schemes apply machine learning-based concept. An approach by Elias Khalil *et al.* [108] provides the machine learning framework for the branching variable selection in B&B for MIPs. It learns surrogate functions to mimic strong branching by solving the learn-to-rank problem [109]. The model uses three broad features: atomic features, interaction features, and features equivalent to the degree-2 polynomial kernel. These consist of 1) static information containing various information extracted from the MIP problem and 2) the dynamic features extracted from the current and the solution history. Experiments on MIP benchmark instances by implementing it on IBM cplex 12.6.1 produces a significantly smaller search tree than existing heuristics finding it competitive with a state-of-the-art commercial solver.

Similar work on a learning-based approximation to estimate the strong branching is mentioned in [110]. It broadly collects two types of features, static and dynamic. Static features are from input parameters in the MIP. Dynamic features are chosen from 1) solution of the problem at a current node, such as Driebeek penalties [111], up and down-fractionality in the solution, and 2) state of the optimization solution, such as a change in objective value when a variable is chosen for branching. Once the features are collected, an *extremely randomized trees* [112], a modified random forest algorithm, is used for learning the branching heuristic function. The experiment of applying the strategy showed promising results on some chosen instances from MIPLIB libraries.

Branching on Multi-aggregated Variables

Multi-aggeration is a presolving method to reduce the number of variables. It replaces variables with an affine linear sum of other variables. Though it reduces the size of the problem, it restricts the degree of freedom in variable-based branching rules. Gerald Gamrath, *et al.*, in their work on branching [113], presented a scheme for considering both general disjunctions defined by multi-aggregated variables and the standard disjunctions based on single variables for branching. It leads to a hybrid method that uses variable and constraints-based branching rules. The idea is implemented in SCIP and incorporated into a strong branching rule, that reduces the number of nodes on a general test set of publicly available benchmark instances. The computational result shows that method is effective for a specific class of problems.

2.3 Issues in Reliability Branching

2.3.1 Reliability Requirements - Same for Every Branching Candidates

In a reliability branching scheme, whether the given branching information of the branching candidate is reliable is decided by a positive parameter. We generally call this the threshold value (λ). Note that we perform λ number of strong branching calls on a candidate until it contains reliable branching information. Once it has reliable information, we stop calling strong branching and start estimating the branching information by using the weighted average of previously calculated explicit information obtained from strong branching calls. The disadvantage of this rule is that the threshold value for all the variables is the same. However, structurally different variables inside a model will have other reliability requirements. Gregor Hendel [93] in his work on a sample variance-based pseudocosts variable branching, emphasizes this issue. He develops a variance-based reliability scheme that updates the pseudocost scores (to modify the reliability requirement) by using the sample variance of the past observation of each candidate. A subset of variables with high variance is considered unreliable candidates and thus calls strong branching, and those with low variance are considered reliable.

2.3.2 Limited Information in Branching Selection Score

Reliability branching only uses objective gain information in the pseudocosts evaluation. One can use more information besides the objective value, such as dual information, a sensitivity range of the objective function coefficient of decision variables, the impact of the active constraints in the relaxation of the subprogram to the parent node [114], Driebeek penalties [111]. A work on machine learning-based variable branching [110] uses such information to approximate the strong branching.

2.3.3 Threshold Value is Invariant over Problem Instances

The same fixed threshold value over various problem instances might not scale with the changes in the problem sizes. The growth of the B&B tree is unpredictable while solving the problem instances. If it were, we would have an ideal branching mechanism to judge the threshold value before solving a model.

2.3.4 Branching Decision is Short-Sighted in Nature

Including reliability branching, almost all current branching mechanisms are myopic. That is, they exploit (select) the branching variable with the highest objective gain, but do not spend time choosing other variables. The decision to locally select the best candidates causes it suffers from the problem of not finding optimal local solutions.

2.3.5 Uneven Calls of Strong Branching

The decision of what should be an ideal threshold value and when to perform strong branching calls in reliability branching is crucial. In Achterberg's reliability branching, strong branching is called more often on the top nodes in the tree. It is because every integer variable which appears as a branching candidate (for the first time) is initialized as unreliable and set to reliable only if the number of strong branching calls on it reaches the threshold value (λ). This leads to the problem of uneven calls of strong branching in reliability branching. The resulting B&B tree contains nodes with strong branching calls on the top nodes and nodes with no strong branch calls after a certain level of depth in the tree. This may result in sharing of branching information between two *structurally different* nodes and performing strong branching on two *structurally similar* nodes. We use the term structurally different and similar nodes to denote the extent to which the subproblems associated with the nodes differ in bounds on variables. Two nodes are similar if fractional changes in bounds on variables are much less. Else, we call them different nodes. We can avoid this by calling the strong branching not only the top node but throughout the tree. One of the rules of thumb is to provide a fixed gap (say g) between two consecutive strong branching calls. With this, we can avoid calling strong branching on a candidate on both the nodes, which are not more than g hops apart. It can help spread the total strong branching calls throughout the tree. However, the problem with such a rule is the unnecessary calls of strong branching even if one of the nodes, which is "structurally similar" to the current node, has already called the strong branching on the corresponding variables. We mean the similar term nodes here to denote the nodes with

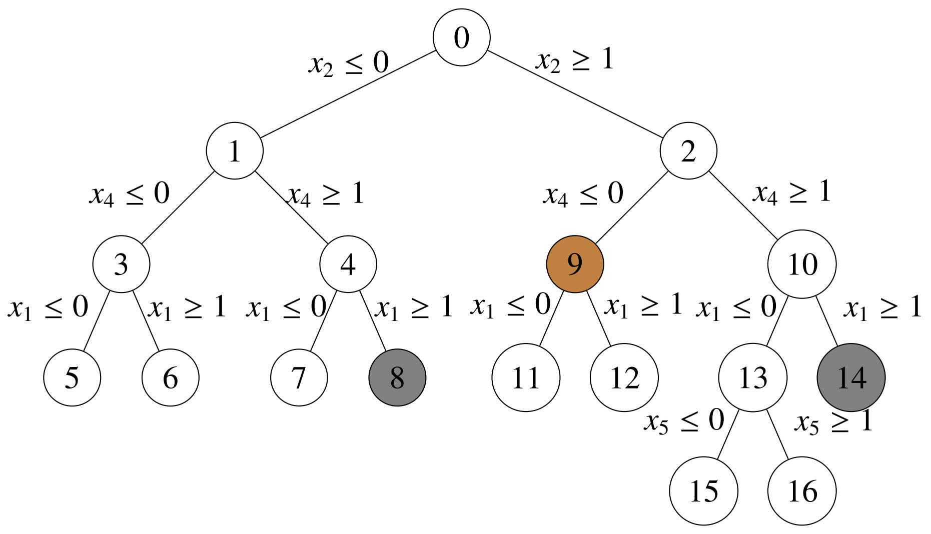


Figure 2.2: Node8 and Node14 are only differ by x_2. Whereas, Node8 and Node9 have bound differences in x_1, x_2 and x_4

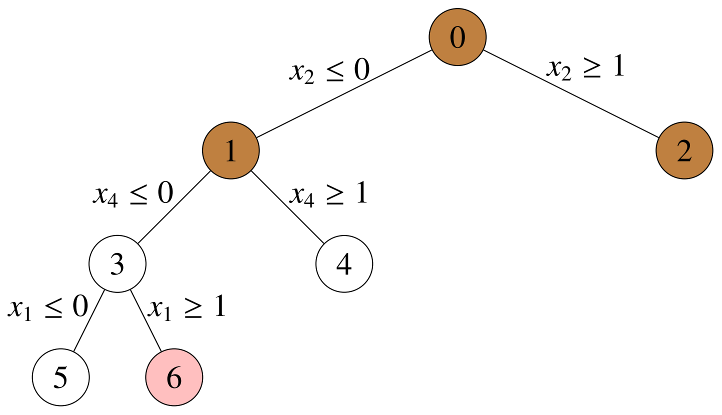


Figure 2.3: Node 2 is differ by one, two and four bounds from the node 0, node 1 and node 6, respectively.

nearly equal change in the objective function. We define it formally in the next section. Let us consider two examples:

Figure 2.2 represents a typical branch-and-bound tree to understand this issue. In this figure, we highlight the position of different nodes, which perform strong branching calls on the potential candidates. If we measure the similarity between nodes by the number of *hops* between them, Node 8 and 14 are farther apart–the distance between nodes 8 and 14 is 5 hops. If the fixed gap (g) is set to 4, we will call strong branching on both the nodes even though the subproblems associated with those nodes only differ in one bound, which is x_2. Assuming such a slight change in the problems result in a similar behavior of the branch-and-bound algorithm, the branching information obtained at Node 8 can be helpful to Node 14. Similarly, if we call strong branching on 8, we will not call strong branching at Node 9 as $g = 4$. However, structurally subproblems associated with 8 and 9 are different by 4 variable bounds. Figure 2.3 also mentions the case where strong branching is called at the current processing node 2, just after the Node 6 was processed. There might be the case where strong branching is called on variable x_4 at Node 2. We can observe that Node 1 has the strong branching information for variable x_4, and it (structurally) differs by only one bound. Node 1 could have shared the strong branching information with Node 2.

2.4 The Notion of Similarity-based Branching - SimBranch

We discussed in the previous section the importance of strong branching in most variable branching schemes. It is mostly helpful in reducing the number of search nodes in the tree, but it requires much effort to evaluate the branching candidates. To overcome the issues of effective use of strong branching, we try a two-pronged strategy. First, we perform strong branching at nodes that are well 'spread-out'. Second, at any given node, say Node A, we use the strong branching information collected from only those nodes that are 'similar' to A.

We keep track of nodes where strong branching was deployed for any candidate. At an active node where we want to select a branching candidate, we first find all 'similar' nodes and collect strong branching information from these nodes only. If a branching candidate was evaluated in any of nodes 'similar' to the current node, then the information available from similar nodes is used to estimate the candidate's score. If none of the 'similar' nodes evaluated the candidate, we perform strong branching to get the score. This strong branching information is then stored for future use. We start by explaining the notion of 'similarity' and then describe the algorithm. The details of how the strong branching information is stored and retrieved are presented later.

2.4.1 Similarity of Nodes

We would ideally like to call two nodes of a B&B tree 'similar' if branching on any given candidate in the two nodes results in a nearly equal change in objective function values. Predicting whether two nodes are similar without actually evaluating the change is difficult, so we resort to a different and easier criterion. Since the LP-relaxation of

Algorithm 4: SimBranch

```
Input: $N^i$: Current processing node; $C^i$: List of branching candidates
        associated to $N^i$
Output: A suitable candidate, $j^*$ for the branching
Find list of processing nodes, $S^i$ similar to $N^i$
for $j$ in $C^i$ do
    if branching score of $j$ is not available in any node in $S^i$ then
        call strong branching to compute a score $s^j$ on $j$;
    else
        collect the average branching score available from the score of $j$ from
          the nodes $S^i$;
    end
end
Return candidate $j^*$ with maximum score $s^j$ for $j \in C^i$.
```

a node differs from that of another only in bounds on the integer variables, our notion of similarity is also based on the bounds of integer variables. A more general notion of similarity can include more features of the subproblems besides the bounds on variables and can be explored in the future. We define the similarity between two nodes A and B, as follows:

Definition 2.1. *Suppose $F^A \in \mathbb{R}^d$ and $F^B \in \mathbb{R}^d$ are feature vectors associated with nodes A and B respectively. We say A and B are similar if $||F^A - F^B||_1 \leq \theta_F$ where $\theta_F > 0$ is a given parameter. We refer to the distance $||F^A - F^B||_1$ as 'feature distance' between nodes A and B.*

To better explain the notion of similarity among nodes, we take the help of a simplistic B&B tree illustrated in Figure 2.4. We assume that the MILP has only binary variables, n in number, and that we branch on variables only. Each branching thus creates two new (child) subproblems by fixing the suitable branching candidate to zero or one. Further, we assume that branching variables are chosen lexicographically, i.e., variable x_1 is used for branching in node-0, variable x_2 in nodes $1, 2$, x_3 in $3, 4, 17, 18$ etc. The feature vector for any node in this example would have $2|I|$ binary elements. For example, F^0, F^1 and F^2 at the corresponding nodes $0, 1$ and 2, with $|I| = 6$ binary variables, are as follows: $F^0 :=$ [0 1 0 1 0 1 0 1 0 1 0 1], $F^1 :=$ [0 0 0 1 0 1 0 1 0 1 0 1] and $F^2 :=$ [1 1 0 1 0 1 0 1 0 1 0 1]. Here each element in the list with odd index $j \in O := \{1, 3, 5, 7, 9, 11\}$ contains the lower bound of a variable $x_{\frac{j+1}{2}}$. And even index $j \in [12] - O$ contains the upper bound of a variable $x_{\frac{j}{2}}$. The first and the second elements in F^0, F^1 and F^2, are 0 1, 0 0 and 1 1. It

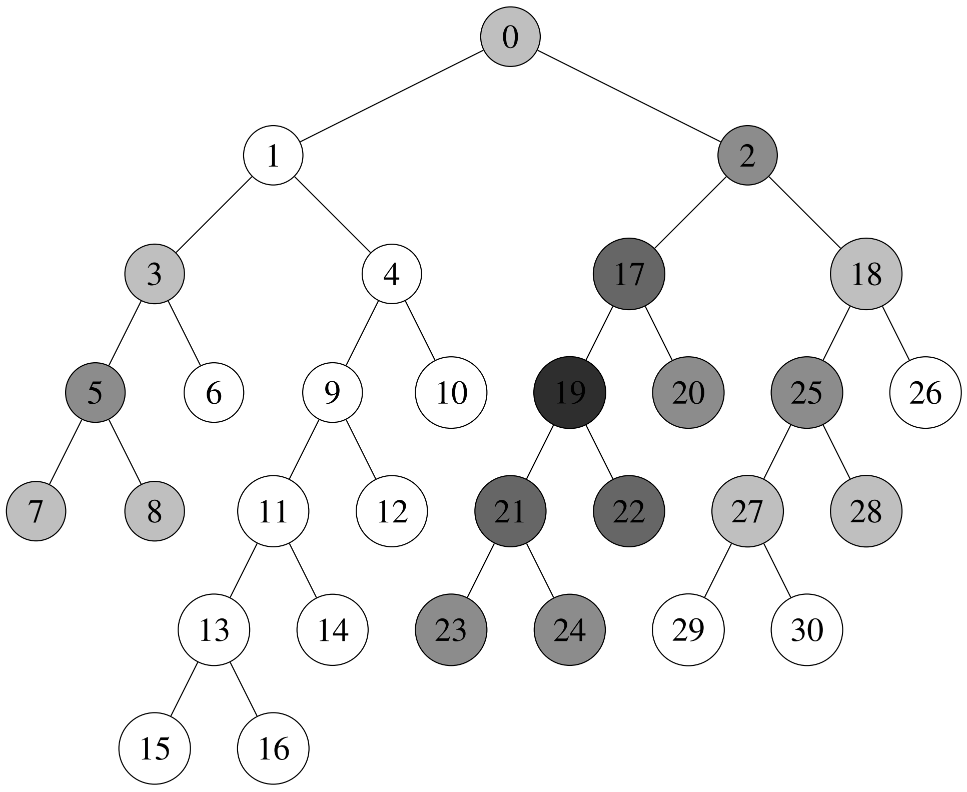


Figure 2.4: Similarity in B&B tree: Shaded nodes are at a distance three or less from Node19. Darker nodes are more similar to 19 than others.

indicates that x_1 is fixed as $x_1 = 0$ and $x_1 = 1$ in the subproblem at node N^1 and N^2, respectively. By Definition 3.7 $\|F^{19} - F^5\| = \|F^{19} - F^{23}\| = 2$, that is Node 5 is quite similar to Node 19, even though the two seem to appear far apart in the tree.

If suppose $\theta_F = 2$ in the above example, then any information collected from strong branching at nodes 2, 5, 17, 20, 25 will be used to estimate scores of candidates at Node 19. If there is no information for a candidate in any of these nodes, then strong branching is performed for that candidate. The steps in SimBranch for a branching variable selection are described in Algorithm 2.4. As θ_F is increased, the distribution of nodes similar to a given node becomes more complicated, and explicitly storing the feature vectors and strong branching information becomes too cumbersome. So we need a systematic approach to manage this difficulty. We describe it after pointing out some connections between SimBranch and other strong branching related methods.

2.4.2 Connections with Other Methods

The similarity parameter θ_F is important and can be used to control the method. A low value of θ_F allows few nodes to be considered similar and leads to more strong branching calls. When $\theta_F = 0$, strong branching will be called on every branching candidate at every node.

SimBranch can also be viewed as a variant of reliability branching [91] with some key differences. Borrowing the notion of a reliable candidate, we can say that a candidate in SimBranch is considered reliable if we have information about it from a node that has characteristics similar to the current node. Reliability branching has a threshold parameter, λ, to decide whether a branching score of a candidate is reliable. If strong branching has been performed on a candidate λ times, anywhere in the tree, it is considered reliable and its branching score is evaluated on the basis of previously collected scores.

SimBranch with a very high value of θ_F is similar to reliability branching with $\lambda = 1$ – both methods will evaluate each candidate only once before switching to estimated values. Selecting an extremely high value of λ in Reliability Branching or setting $\theta_F = 0$ in SimBranch makes them both equivalent to the traditional strong branching. When θ_F is not set to any of its extreme values, SimBranch may behave quite differently from reliability branching. SimBranch will re-evaluate candidates only when the tree size becomes large and the nodes become dissimilar. The variance-based enhancements [93] of reliability branching does something similar by looking at the variance in the output i.e. the variance in the observed values of changes in objective values. We, in contrast, look at the variance in the inputs (the features or characteristics of nodes).

2.5 Implementing SimBranch

In order to implement SimBranch efficiently, two key concerns need to be addressed: (a) how to store the information collected at various nodes and (b) how to find 'similar' nodes and compute the scores. The first concern arises because we need the information about the nodes in addition to the strong branching scores. Before addressing this concern in Section (2.5.1), we describe hashing functions for storing node features. We use some additional notations to describe SimBranch. We have mentioned them in Table 2.1. For the sake of explanation, we will assume the problem to be an MBP (2.2). We also assume that the B&B method always branches on a binary variable. These assumptions are not too restrictive and the method can be extended for cases where they do not hold. A feature vector that defines a node under these assumptions is just a binary vector of size $2|I|$ containing lower and upper bounds of binary variables. If F^N is a feature vector, one of its representations would be

$$[lb^N_{I(1)},\ ub^N_{I(1)},\ lb^N_{I(2)},\ ub^N_{I(2)}, \cdots, lb^N_{I(|I|)},\ ub^N_{I(|I|)}].$$

Table 2.1: Notation for SimBranch

N^i	: node i of branch-and-bound tree
κ	: number of hashes computed for any node
h^i	: vector (of size κ) of hash values of node i
C^i	: index set of candidate branching variables at node i
S^i	: the set of nodes which are similar according to our measure to N^i
$\Delta_j^{i+}, \Delta_j^{i-}$	: Up and down strong branching scores of variable j at N^i
δ_j^i	: Indicator variable that is one if strong branching is done on variable j at N^i, and zero otherwise
ψ_j^{i+}, ψ_j^{i-}	: up and down pseudocosts of variable j computed up to node i
$\sigma_j^{i+}, \sigma_j^{i-}$	: up and down similarity scores of variable j at node i
τ_j^i	: total SimBranch score of variable j based on pseudocosts and similarity score at node i
α, β	: parameters in [0, 1] used to compute τ_j^i from $\psi_j^{i+}, \psi_j^{i-}, \sigma_j^{i+}, \sigma_j^{i-}$
lb^i	: a vector of lower bounds on variables at node i
ub^i	: a vector of upper bounds on variables at node i
L	: a list containing strong branching information collected at nodes
θ	: threshold value for similarity measure.

Where $I(i)$ is the i^{th} element in index set of binary vectors I and, $lb^N_{I(i)}$ and $ub^N_{I(i)}$ are the lower and upper bounds of the of the binary variable x_i.

One of the significant computational challenges in implementing SimBranch is finding the similarity between the nodes (say node A and node B) using to Definition 2.1. There are many procedures to explicitly measure the degree to which feature vectors, F^A and F^B, are similar, such as Jaccard coefficient, Cosine similarity, Hamming distance, Euclidean distance, and Minkowski distance [115]. These procedures are computationally expensive for a large data set. As the size of the B&B tree grows, finding nodes similar to the current processing node amongst all the explored nodes by explicitly comparing the features is computationally inefficient. Also, subtrees that have been explored are usually deleted to reduce the memory requirements, thereby making those nodes unavailable for evaluating their features. To overcome these challenges, we use a hashing scheme to represent the nodes and the information we want to exploit to overcome these challenges.

For a fixed parameter κ, vectors $H_1, \dots, H_\kappa$ each having $2|I|$ random numbers drawn from uniform distribution are created. κ hash values $h^i_1, \dots, h^i_\kappa$ are created for node i using the relation

$$h^i_j = H^T_j F^i, \quad j = 1, \dots, \kappa. \tag{2.12}$$

Instead of storing and comparing large feature vectors for every node, we propose to store hash values h^i_j, $j = 1, \dots, \kappa$ for each node. As we shall see, $\kappa = 5$ works reasonably well, even when $|I|$ can be in thousands, so our storage requirements come down. Since we store only the hash values, then Definition 2.1 for 'similarity' needs to be suitably modified.

Definition 2.2. *Suppose $F^A \in \mathbb{R}^d$ and $F^B \in \mathbb{R}^d$ are feature vectors (each containing d features) associated to node A and B respectively. We say A and B are similar if $|h^A_j - h^B_j| \leq \theta$ for each $j = 1, \dots, k$, where $\theta \geq 0$ is a given parameter.*

If two nodes are similar in features, then their hash values would also be similar. The converse is not always true, but by choosing a sufficiently large κ, we can ensure that the converse is true with a high probability. These hash values can therefore be used to find similar nodes faster and with less storage. In return for this gain, we may have to sacrifice of some accuracy.

2.5.1 Storing Branching Information

All similarity-based branching information is stored in a single data-structure as follows. We store node specific information for only those nodes where we perform strong branching on at least one variable. This information consists of:

1. An array of κ hash values of the node
2. An array of indices of variables on which we performed strong branching at this node
3. Two arrays, one for storing the up-scores of strong branching variables and the other for their down-scores

The vectors $H_1, \dots, H_k$ are created before the start of B&B and stored. Suppose we are processing a node N^i with bounds, lb^i and ub^i on its variables. If the node relaxation is infeasible or if the relaxation solution is integer feasible or if node is pruned because of its lower bound, then no extra branching information is required to be stored. If branching is required, then a vector F^i is first created, by concatenating lb^i and ub^i of all binary variables of the problem associated with Node i, and all the κ hash values are computed. Nodes similar to Node i are searched for information collected.

If strong branching is required at the node, then the information generated from the branching is stored using the above mentioned data-structure. A linked list L is used to store data-structures for different nodes. The number of objects in this list L is usually much smaller than the number of nodes in the tree as only those nodes where strong branching is performed enter this list.

2.5.2 Selecting a Branching Candidate

Given a node N^i, a solution to the relaxation, $\hat{x}$, and bounds, lb^i and ub^i, on the variables of the subproblem, the following procedure can be used to select a branching candidate. SimBranch first finds the set S^i of nodes "similar" to N^i. The set S^i is initially empty. The hash values $h^i_j, j = 1, \dots, \kappa$ are computed. Each element of the linked-list L is considered one-by-one. Suppose for notational convenience, an element of L corresponds to Node-N^k of B&B tree. N^k is added to S^i if they are similar, i.e., if

$$\left|h^i_j - h^k_j\right| \leq \theta, \text{ for each } j = 1, \dots, \kappa$$

Next, a list C^i of branching candidates is created. It is comprised of all integer variables that have a fractional value in $\hat{x}$. For every variable $x_j \in C^i$, we collect all available strong branching scores of x_j from each node in S^i and aggregate them into the 'similarity scores'

$$\sigma^{i+}_j = \alpha \cdot \frac{\sum_{k \in S^i}(\Delta^{k+}_j)}{\sum_{k \in S^i} \delta^k_j} + (1-\alpha)\psi^{i+}_j \text{ and } \sigma^{i-}_j = \alpha \cdot \frac{\sum_{k \in S^i}(\Delta^{k-}_j)}{\sum_{k \in S^i} \delta^k_j} + (1-\alpha)\psi^{i-}_j, \tag{2.13}$$

if any similar nodes are found. Here $\alpha \in [0,1]$ is a fixed parameter that defines relative weight of Pseudocost Score and SimBranch Score. Δ_j^{k+}, Δ_j^{k-} are strong branching scores obtained from the node N^k. δ_j^{k+}, δ_j^{k-} are one if strong branching information for variable x_j is available at N^k and zero if not.

The above scores are defined only if one or more nodes in S^i have Strong Branching information about x_j. In such a case, the total score of x_j is evaluated as:

$$\tau_j^i = \beta \cdot \max\{\sigma_j^{i+}, \sigma_j^{i-}\} + (1-\beta)\min\{\sigma_j^{i-}, \sigma_j^{i+}\}, \tag{2.14}$$

where β is a fixed parameter in $[0,1]$. Parameter β is also used in pseudocost branching similarly. [22] also suggested the product rule to combine the up and down scores

$$\tau_j^i = \max\{\Delta_j^{i-}, \beta\} \cdot \max\{\Delta_j^{i+}, \beta\}. \tag{2.15}$$

In case none of the nodes in S^i have any strong branching information about x_j, then we perform strong branching on x_j. The strong branching up and down scores are evaluated as

$$\Delta_i^+ = \frac{f_{N_i^{k+}}(\hat{x}) - f_{N^k}(\hat{x})}{f_i^+} \text{ and } \Delta_i^- = \frac{f_{N_i^{k-}}(\hat{x}) - f_{N^k}(\hat{x})}{f_i^-}. \tag{2.16}$$

Where $f_{N_i^{k+}}(\hat{x})$ and $f_{N_i^{k-}}(\hat{x})$ are the objective value of the relaxations of subproblems related to up and down child nodes.

Δ_j^{i+} and Δ_j^{i-} are added to the data-structure for future use. The score of x_j at N^i is determined as:

$$\tau_j^i = \beta \cdot \max\{\Delta_j^{i+}, \Delta_j^{i-}\} + (1-\beta)\min\{\Delta_j^{i+}, \Delta_j^{i-}\}$$

Every time we perform strong branching we also update the pseudocosts ψ_j^{i+} and ψ_j^{i-}. Finally, the variable with maximum score is selected as the branching candidate.

$$j^* = \arg\max_{j \in C^i}\{\tau_j^i\}. \tag{2.17}$$

2.5.3 Parameters in SimBranch

Performance of SimBranch relies on the values of its parameters. Among these, the parameters θ_F, θ and κ are crucial to SimBranch. In spite of its clear and natural interpretation, parameter θ_F is not directly used in the algorithm. We propose the following formula for setting for θ from θ_F and κ. Assume θ_F is a positive integer. For each hash vector H_i, define θ_i as

$$\theta^i = \sum_{k=1}^{\theta_F} \omega_i^k, \quad i = 1,2,3,\ldots,\kappa,$$

where ω_i^k is the kth largest element in H_i. Thus, θ_i is the sum of θ_F largest elements of H_i. Then θ can be set to

$$\theta = \max_{i=1,2,\cdots,\kappa} \theta^i$$

The motivation for setting θ in this manner is as follows. We would like a value of θ so that we are as accurate as possible in determining similarity. There can be two types of errors when trying to ascertain similarity from θ in place of θ_F.

1. T1 error (false positive): $\|F^A - F^B\|_1 \leq \theta_F$. But $\left|h^A - h^B\right|_\infty > \theta$.

2. T2 error (false negative): $\|F^A - F^B\|_1 > \theta_F$. But $\left|h^A - h^B\right|_\infty \leq \theta$.

The proposed scheme ensures that T1 error is zero.

Proposition 2.3. *Given κ vectors $H_1, H_2, \ldots, H_\kappa$ each having d random numbers drawn from a uniform distribution* $[0, 1]$*, and the threshold integer parameter θ^F, let*

$$\theta^i = \sum_{k=1}^{\theta_F} \omega_i^k, \ \forall i = 1, 2, 3, \cdots \kappa,$$

where ω_i^k is the k^{th} largest element in H_i. If, for any two binary vectors F^1 and F^2 of size d, we have $\|F^1 - F^2\|_1 \leq \theta_F$, then $|H_i^T F^1 - H_i^T F^2| \leq \theta^i$, $\forall i = 1, 2, \ldots, \kappa$.

Proof. For feature vectors F^1 and F^2, suppose $\|F^1 - F^2\|_1 \leq \theta_F$. Let I be an index set such that $F^1(i) \neq F^2(i)$ for each $i \in I$. Clearly, the number of elements in I, len$(I) \leq \theta_F$. Define two sets $I^+ = \{i \in I \mid F^1(i) = 1\}$ and $I^- = \{i \in I \mid F^2(i) = 0\}$. Clearly $I^+ \cup I^- = I$, $I^+ \cap I^- = \emptyset$. For all $i = 1, 2, \ldots, \kappa$, we have

$$\begin{aligned} |H_i^T F^1 - H_i^T F^2| &= |1 \sum_{j \in I^+} H_i[j] - 0 H_i[j] + 0 \sum_{k \in I^-} H_i[k] - 1 H_i[k]|. \\ &= | \sum_{j \in I^+} |H_i[j]|, \\ &\leq \sum_{j \in I^+} |\omega_i^j|, \\ &\leq \sum_{k=1}^{\theta_F} \omega_i^k. \end{aligned}$$

□

The above mentioned choice of θ ensures that we never perform strong branching on a candidate if the required information is available in a node whose real feature distance is within θ_F. On the other hand, we may still have T2 error: we may assume two nodes are similar because of their close hash values, when in fact, they may be far apart feature wise.

Next we consider the choice of θ_F. To gain some insight, we perform an experiment in which we mimic a very specific B&B tree (T_{lex}): a full binary tree where branching decisions are taken in lexicographical order (similar to the example (4)). The goal of the experiment is to study the effect of θ_F on number of times we have to perform strong branching. We further assume that at all nodes in the B&B tree, all binary variables that have not been fixed by earlier branching take fractional values. Thus at any node, we either perform strong branching for all unfixed variables or none. For the branching score initialization, we call strong branching at the root node. We calculate the total number of strong branching calls needed to evaluate the score of the last (lexicographic) variable. Since it is the last variable, we never branch on it (except at the leaf nodes), but strong branching is performed on it every time we visit a node that is dissimilar to those where strong branching was done earlier.

Table 2.2 tells us the maximum number of times we have to perform strong branching for a variable for three different tree sizes. We see that the number of calls of strong branching reduce quite fast (nearly exponentially) as θ_F increases, and low values of θ_F may suffice for most trees. In our experiments described in Section 2.6 we use two different values: 10 and 30.

Lastly, a note about choice of κ. The selection of a good kappa value leads to embedding a set of points in a high-dimensional space into a much lower dimension so that distances between the points are nearly preserved. A high kappa value means more projection operations, ensuring low T2 errors at the cost of extra computing. A low kappa value will neglect some points in high dimensions to be preserved in low dimensions. We found κ values in the range 3-7 to be quite reasonable. We use the value 5 in our experiments described next.

2.6 Computational Results and Summary

In this section we provide empirical evidence of the effectiveness of SimBranch by implementing it in CBC (Coin-OR Branch and Cut) [88], an open-source mixed-integer linear programming solver. We compare the performance of our implementation (SimBranch) with that of the default branching scheme in CBC (Default-Cbc). For both solvers, we turned off primal heuristics and provided the best known solution value as an input. This change was made to neutralize the unpredictable behaviour of the solvers on account of 'accidently' finding feasible solutions either through primal heuristics or during the B&B tree search. All other settings including those of presolve, cuts etc., were left undisturbed. Like CBC, our subroutines are written in C++ and compiled with GCC-6.3.0.

The hardware used for the computation is a 64−bit Intel(R) Xeon (R) E5-2670 v2 at 2.50GHz CPUs with 20 cores and 128GB RAM. To avoid multiple processes to share

Table 2.2: Number of strong branching nodes in the tree for the given values of θ_F and number of binaries

Depth of the tree	9	10	11
Total nodes	1023	2047	4095
θ_F	Strong branching nodes		
0	1023	2047	4095
1	341	1365	1365
2	253	529	1013
3	121	441	441
4	46	84	156
5	17	64	64
6	17	17	49
7	17	17	17
8	5	10	14
9	1	9	9

common resources, we run one job at a time. CBC by default does not use multiple CPUs in parallel.

Test problems for our comparison of different approaches consist of instances chosen from MIPLIB 2010 [102] and MIPLIB 2017 [116], libraries of MILP benchmark problems. MIPLIB 2010 and MIPLIB 2017 comprise 87 and 240 benchmark instances that respectively contain 84 and 221 pure and mixed binary instances. We ran both SimBranch and Default-Cbc solvers on these 305 pure and mixed binary instances and shortlisted those instances for which at least one of the two solvers took more than 100 nodes to solve. There were 222 such instances in all.

We set the computational time limit to 7200 seconds for each test instance. The settings for SimBranch procedure implementation are as follows. Parameter κ is set to 5. Based on the experiment in Section 2.5.3, we set θ_F to 30 if the number of binary variables in an instance is more than 300. Otherwise, we set it to 10. Also, branching parameters 'maxstrCand' and 'maxitrPerStr', which define the maximum number of strong branching candidates per node and maximum number of (dual simplex) iterations in each strong branching operation, are set to 10 and 100 respectively. We set 'maxstrCand' to 20 and 'maxitrPerStr' to 80 for the instances with less than 300 binary variables. The parameter α is set to 1 so as to not have any effect of Pseudocost Scores in candidate selection.

The rule used to compute the branching score for each candidate in SimBranch is same as in Default-Cbc. The score-factor β is set to a constant value 6E-7 in both solvers.

Table 3.4 compares the overall performance of our branching procedure to CBC. We use shifted geometric mean (SGM) to summarize the solving time. SGM, similar to geometric mean of n elements, is the n-th root of their product. However, each element is added with a positive integer before computing n-th root [117]. This added positive value is called shift. The computed n-th root is then subtracted with shift value. Thus, for a shift $s \in \mathbb{R}_+$, SGM of n numbers, $n_1, n_2, \cdots, n_n$ is calculated as, SGM = $\left(\prod_{i=1}^{n}(n_i + s)\right)^{1/n} - s$. The benefit of using SGM is that it avoids the effects of large outliers, which we see in case of arithmetic and geometric means [118]. In our experiment we use a shift of 10 seconds to report the mean solving time.

We have additionally reported SGM for instances that take more time to solve in order to check whether the SimBranch is effective for only 'easy' to solve instances. Based on the time taken to solve by the two solvers, we classify the instances into four categories: 1) all instances that were solved by at least one solver, 2) instances where at least one of the solvers took more than 500 seconds and at least one solver solved it, 3) instances where at least one of the solvers took more than 1000 seconds (and at least one solver solved it), and 4) instances where at least one solver took more than 2000 seconds (and at least one solver solved it). In each of the categories we report the number of instances solved by each solver and also the number of instances solved by both. The SGM values of time and nodes is computed over the number of instances solved by both.

SGM of total number of nodes explored in B&B is reported with a shift of 50. Also, SGM of optimality gaps ('gap') of those instances that could not be solved by both the solvers is reported with a shift of 1 (percentage) in the 'Time Limit' category. We also report the mean of SimBranch relative to the mean of Default-Cbc (the column % in the table). A value below 100 states an improvement over the default.

We observed that SimBranch could solve 79 out 222 instances and hit the time limit on rest of the instances. Default-Cbc could solve 77 instances in comparison. 71 out of 222 instances were solved by both the procedures, and SimBranch was 19.28% faster than Default-Cbc on these. The reduction in nodes processed in case of SimBranch was observed to be 30.38 %. For 'harder' instances the speedup in SimBranch is slightly higher. Further, the category 'Time Limit' reports that the gap closed by SimBranch is marginally worse (by 0.07%) on instances that hit time limit on both solvers.

Details of the running times, the number of nodes processed and the number of strong branching iterations required in strong branching operations for the instances are available in Table A.1 in the Appendix. It lists performance of SimBranch and Default-

Cbc procedures over all 222 benchmark instances. Table A.2 details node processed, strong branching iterations and percentage gap of those instances which could not be solved by both the procedures within 7200 seconds.

In Table 2.4, we list fourteen instances that were solved by one solver and not the other. We observe that some instances can be easily solved by SimBranch but difficult for Default-Cbc. Similarly for some instances Default-Cbc outperforms SimBranch. The column #strong_itrn reports number of strong branching iterations performed for a given instance under each setting. Rows for 'danoint', 'sp150x300d' and 'neos5' in the Table indicate that fewer and carefully chosen strong branching calls are effective, but it is opposite for instances 'trento1' and 'neos-916792'.

To highlight the difference in the use of strong branching calls in the branching procedures we collect information from top few nodes in the B&B for a specific instance, gmu_35_40. This information is reported for Default-Cbc and SimBranch in Table 2.5 and Table 2.6 respectively. They contain columns 'brCand' and 'strCand' that list indices of branching candidates and strong branching candidates respectively. For each node we show 'brCand' and 'strCand'. We report the information of the top seven nodes of the B&B tree. Each node is represented by node id and depth of the current processing node. For instance, take the third column in Table 2.5. It depicts depth = 1 and node id = node1002. It states that the number of nodes processed so far is 1002 and the current node is positioned at depth 1. Though the number of processed nodes between the current node and the adjacent node at the same depth (with node id = node1) is more than 1000, they are structurally close to each other. Default-Cbc does not capture this and thus, there are repetitions of strong branching calls. This information is captured and other candidates are evaluated by SimBranch. Highlighted entries in Table 2.5 list repeated strong branching calls. We give a summary of repetition of strong branching calls at the bottom of the Tables. #totalStrong and #repeatedStrong denote number of strong branching candidates per node and number of those strong branching candidates whose information can be used from the explored nodes respectively. We can see in the case of Default-Cbc branching (Table 2.5) there are 45 repeated strong branching calls out of total 99. Whereas SimBranch (Table 2.6) does not have such repetition.

For all those instances which could be solved by both the branching procedures, the number of strong branching iterations, the dual simplex iterations required in strong branching operations in branching decisions in B&B is collected. The ratio of total number of strong branching iterations in SimBranch (SimBranchStrong) to Default-Cbc (cbcStrong) is computed and shown as a scatter plot in Figure 2.5. A point denotes the ratio of SimBranchStrong to cbcStrong and a horizontal dotted line (at 1.0) represents a reference line which divides bubbles into two sections. 68% of the bubbles which are below reference line indicates that SimBranch on average requires less number of strong branching iteration. For these instances, geometric mean values of strong branching iterations required by SimBranch and Default-Cbc are 367759.5 and 1531628.9 respectively. Thus SimBranch performed about 75% fewer strong branching evaluations.

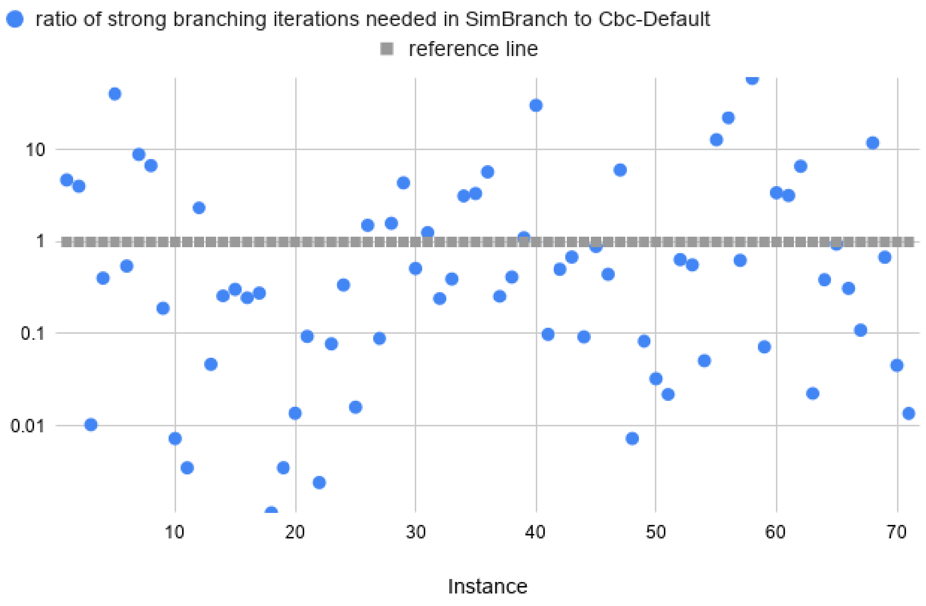


Figure 2.5: A scatter plot of ratio of number of strong branching iterations (SimBranch-Strong) when SimBranch is chosen as branching candidate to that of Default-Cbc on 71 solved benchmark instances: 68% of points are below the reference line.

In Figure 2.6, we have summarised the total time spent in computing hash values in solving a given problem instance. A reference line indicates geometric mean of hash computation time. We observe that SGM (with a shift of one second) of time taken in computing hash values is 3.51 seconds over 222 instances which is illustrated in the scatter plot. This computation time is insignificant against total time taken in solving the problem instance. However, there are instances in which computing hash values go up to more than 100 seconds. These instances had a huge number of nodes and hit the time limit.

To have a more reliable picture of the effectiveness of SimBranch in Default-Cbc, we have done similar experiments with three different random seeds on 71 instances. These are instances that are solved by both the settings (SimBranch and Default-Cbc) within the given time limit of 7200 seconds. In Figure 2.7 we report a computational summary of SimBranch over different three seed values. We see a marginal difference in the scatter plot corresponding to each seed. Geometric means of solving time under three different

Table 2.3: Computational summary of performance of SimBranch compared to Default-Cbc.

Category	#instances solved by both	solved		SGM of solving time				SGM of nodes processed				SGM of %age gap	
		#instances solved by Default-Cbc	#instances solved by SimBranch	Default-Cbc		SimBranch		Default-Cbc		SimBranch		Default-Cbc	SimBranch
				t (sec)	%	t (sec)	%	n	%	n	%		
0 – 7200	71	77	79	450.83	100	**363.95**	80.72	14055.4	100	**9785.98**	69.62	-	-
500 – 7200	44	44	44	1344.11	100	**950.67**	70.73	24691.64	100	**15732.64**	63.72	-	-
1000 – 7200	28	28	28	1984.86	100	**1663.45**	83.81	28943.86	100	**20908.37**	72.24	-	-
2000 – 7200	15	15	15	3733.58	100	**2362.59**	63.28	51318.65	100	**30366.76**	59.17	-	-
‘Time Limit’	-	-	-	-	-	-	-	-	-	-	-	**10.21**	10.28

settings each with different seed values are 326.92, 330.80 and 339.20. The variance in mean seems lower than the difference between SimBranch and Default-Cbc.

To conclude, SimBranch, a new variable branching scheme, looks more closely at the information collected at different nodes and tries to use them selectively. Effectively calling strong branching speeds up the LP-based B&B for MILP by 20% and results in a 30% node reduction. Carefully tuning the algorithm and the data structures should lead us to more improvements. The scheme is readily extendable to general integer cases and possibly other classes of problems like MINLP and CSP. The idea of similarity can also be tried in the node selection strategy.

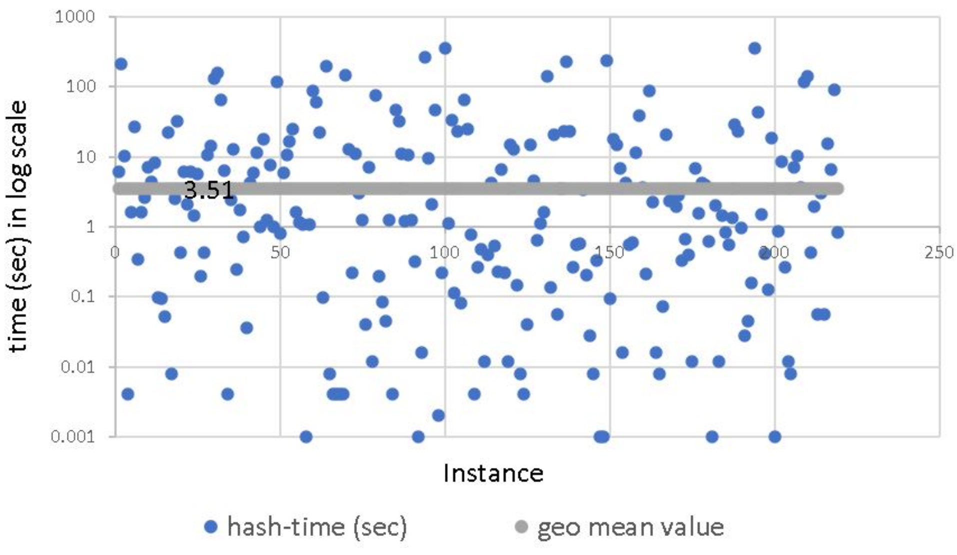


Figure 2.6: A scatter plot of time taken in computing hash values in SimBranch procedure: Shifted geometric mean with a shift of 1 sec is 3.51 sec (horizontal line)

Table 2.4: Comparison of SimBranch and Default-Cbc on instances where one solve hit time limit

	SimBranch				Default-Cbc			
Instance	t(sec)	node	#strong_itrn	gap	t(sec)	node	#strong_itrn	gap
app1-2	7200	12862	3704174	12.12	**4140.93**	13876	1633646	-
neos-1109824	7200	78468	5175	6.22	**4941.3**	37686	3192172	-
neos-916792	7200	315484	27282	12.33	**1944.75**	55134	2845557	-
n3div36	7200	77745	12779475	3.38	**7189.3**	263950	2997734	-
satellites1-25	7200	26777	18050837	2.91	**1189.34**	23446	299649	-
satellites2-60-fs	7200	2729	2388503	57.89	**3908.15**	3526	137085	-
biella1	**628.05**	1350	1181269	-	7200	83754	1308635	0.09
blp-ic98	**4006.47**	47156	9096366	-	7200	128681	6411946	1.53
csched010	7007.23	764318	1330629	-	7200	1228224	12031128	8.33
danoint	**4997.59**	502308	65470	-	7200	687526	17675634	4.02
neos5	**1667.73**	3502835	18805	-	7200	1596440	8518782	6.67
rocII-4-11	**5367.99**	45122	2723384	-	7200	32753	1788990	47.5
sp150x300d	**208.4**	46452	48981	-	7200	2730562	608276	8.69
trento1	**1713.32**	5682	3493318	-	7200	77721	954160	0.045

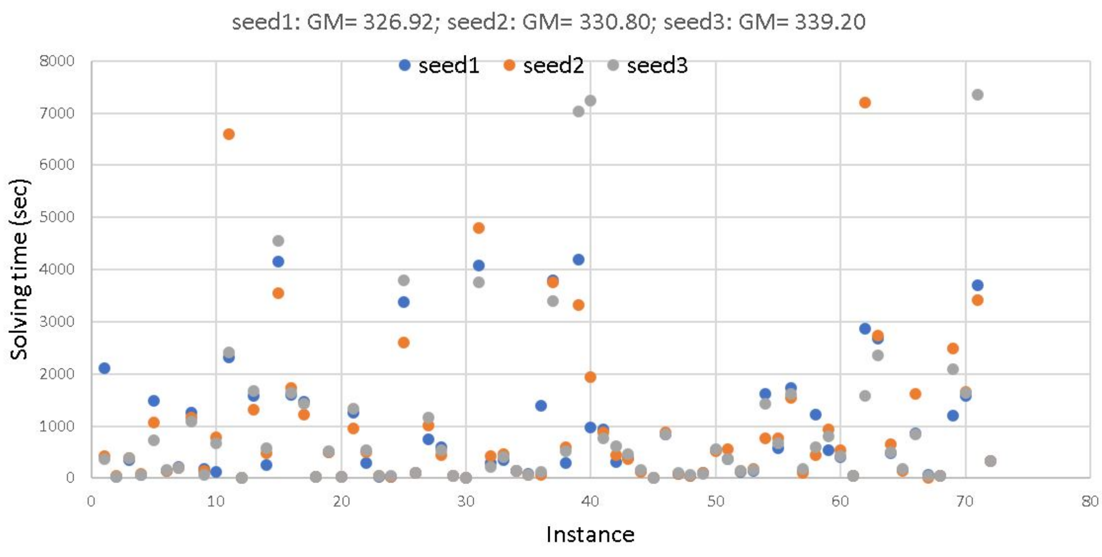


Figure 2.7: Scatter plot comparing solving time of 71 'solved' instances by SimBranch with three different random seeds

Table 2.5: Branching candidates (brCand) and strong branching candidates (strCand) upto depth 2 in the B&B for the instance gmu_35_40 solved by Default-Cbc

	depth = 0		depth = 1		depth = 1		depth = 2		depth = 2		depth = 2		depth = 2		
	node0		node1		node1002		node3		node1003		node1442		node5002		
Sl no.	brCand	strCand	brCand	strCand	brCand	strCand	brCand	strCand	brCand	strCand	brCand	strCand	brCand	strCand	
1	0	0	0	**12**	19	42	9	**9**	0	3	9	**103**	0		
2	2	2	2	**41**	20	98	12	**19**	2	**42**	12	**276**	2		
3	9	9	9	**70**	37	**103**	19	**20**	3	**98**	19		9		
4	12	12	12	**97**	39	253	20	**70**	4	**253**	20		12		
5	19	19	19	**103**	41	**254**	36	**93**	9	280	37		110		
6	20	20	20	**119**	42	276	39	**97**	12		39		114		
7	36	36	36	**127**	69		69	118	19		42		119		
8	39	39	39	**130**	72		70	122	20		69		121		
9	41	41	41	**156**	80		93	124	37		70		127		
10	69	69	69	**170**	82		97	129	39		80		130		
11	70	70	70	**178**	92		110	**130**	42		82		156		
12	93	93	93	**180**	93		114	**152**	98		93		158		
13	97	97	97	**190**	97		118	**156**	110		97		175		
14	103	103	103	**191**	98		119	**158**	121		103		176		
15	110	110	110	**194**	103		121	**170**	127		110		178		
16	114	114	114	**196**	110		122	**192**	130		121		180		
17	119	119	119	**254**	121		124	**196**	152		127		190		
18	121	121	121	**262**	127		127	**198**	153		130		191		
19	127	127	127	**275**	130		129	**232**	156		153		192		
20	130	130	130	**293**	152		130	**275**	158		156		195		
21	152	152	152	**321**	153		152	**321**	169		158		196		
22	153	153	153		156		156		172		172		203		
23	156	156	156		158		158		192		175		204		
24	157	157	157		169		170		193		176		232		
25	158	158	158		172		191		203		178				
26	162	162	162		175		192		204		180				
27	163	163	163		176		194		232		182				
28	165	165	167		178		196		253		253				
29	167	167	168		180		198		262		262				
30	168	168	170		192		203		280		275				
31	170	170	178		193		204		293		276				
32	191	191	180		253		232		321		293				
33	192	192	190		254		275				321				
34	194	194	191		262		321								
35	196	196	192		276										
36	198	198	194		293										
37	203	203	196		321										
38	204	204	198												
39	232	232	254												
40	254	254	262												
41	262	262	275												
42	275	275	293												
43	293	293	321												
44	321	321													Total
fractional-repetition		0		1.00		0.33		0.81		0.60		1.00		0	0.45
#totalStrong		44		21		6		21		5		2		0	99
#repeated-Strong		0		21		2		17		3		2		0	45

Table 2.6: Branching candidates (brCand) and strong branching candidates (strCand) upto depth 2 in the B&B for the instance gmu_35_40 solved by SimBranch

	depth =0		depth=1		depth=1		depth=2		depth=2		depth=2		depth=2		
	Node0		Node1		Node1002		Node2		Node1003		Node5002		Node11001		
Sl no.	brCand	strCand	brCand	strCand	brCand	strCand	brCand	strCand	brCand	strCand	brCand	strCand	brCand	strCand	
1	0	293	0	191	9	70	0	150	0	39	0	155	0	263	
2	2	321	2	19	12	176	2	158	2	181	2	2	2	279	
3	9	170	9	20	37	175	19	172	19	159	9	0	9		
4	12	275	12	262	39	118	20	119	20		12	195	12		
5	19	93	19	41	69	190	69	121	37		110	12	19		
6	20	97	20	103	70	37	70	156	39		114		20		
7	36	194	36	254	110	165	110	180	110		119		69		
8	39	130	39	196	114	69	114	178	114		121		70		
9	41	127	41	192	118	168	119		119		127		82		
10	69	9	69		119	124	121		121		130		83		
11	70		70		121		127		127		156		97		
12	93		103		122		130		130		158		103		
13	97		110		124		150		150		175		110		
14	103		114		125		156		153		176		121		
15	110		119		127		157		155		178		153		
16	114		121		129		158		156		180		156		
17	119		127		130		162		157		190		158		
18	121		130		156		163		159		191		172		
19	127		152		157		167		162		192		175		
20	130		156		158		168		163		195		176		
21	152		158		162		169		167		196		178		
22	153		165		163		172		168		203		180		
23	156		168		165		178		170		204		191		
24	157		170		167		180		172		232		192		
25	158		191		168		191		175				193		
26	162		192		169		192		176				194		
27	163		194		175		194		178				196		
28	165		196		176		196		180				198		
29	167		198		178		203		181				203		
30	168		203		180		204		191				204		
31	170		204		190		232		192				232		
32	191		232		191		321		194				262		
33	192		254		192				196				263		
34	194		262		196				203				279		
35	196		293		198				204				293		
36	198		321		203				232				307		
37	203				204				293				321		
38	204				232				321						
39	232				318										
40	254				321										
41	262														
42	275														
43	293														
44	321														Total
fractional repetition		0		0.0		0.0		0.0		0.0		0.0		0.0	0.00
#totalStrong		10		9		10		8		3		5		2	47
#repeated-Strong		0		0		0		0		0		0		0	0

Chapter 3

Similarity-based Method for Hierarchical Multiobjective Linear Program

3.1 Introduction

A hierarchical multiobjective linear program (h-MOLP) is a linear problem (LP) with more than one objective function, and the order of priorities among those objectives is known to the decision-maker. We recall problem 1.4, a h-MOLP we are interested in solving:

$$\begin{aligned} \texttt{lexmin}\; & {c^1}^{\mathsf{T}}x,\ {c^2}^{\mathsf{T}}x, \ldots, {c^t}^{\mathsf{T}}x \\ \text{subject to}\; & Ax = b, \\ & l \leq x \leq u. \end{aligned} \tag{3.1}$$

As discussed previously, the term lexmin denotes lexicographic minimum. It signifies that first objective (${c^1}^{\mathsf{T}}x$) is much more important than the second objective (${c^2}^{\mathsf{T}}x$) which is, on its turn, much more important than the third one (${c^3}^{\mathsf{T}}x$), and so on and, the last objective (${c^t}^{\mathsf{T}}x$) is of least importance. We also recall two popular methods described in Chapter 1 that solve a sequence of LPs to obtain the solution of the h-MOLP. They are defined as follows:

$$\begin{aligned} \mathrm{LP}^k := \min\ & c^k x \\ \text{s.t.}\ & Ax = b, \\ & c^i x = y^i,\ \forall\, i \in [k-1], \\ & l \le x \le u. \end{aligned} \tag{3.2}$$

$$\begin{aligned} \mathrm{modLP}^k := \min\ & c^k x \\ \text{s.t.}\ & Ax = b, \\ & x_j = f_j\ \forall j \in J^k \subseteq [n], \\ & l \le x \le u. \end{aligned} \tag{3.3}$$

We first obtain the solution of the highest ordered objective in both methods. If it is unique, we stop, and the obtained solution is optimal to the h-MOLP. Otherwise, we pick the next highest important objective and compute its solution without deteriorating the previously obtained optimal objective value. If we find the solution unique, we stop. Otherwise, we continue solving the next highest objective as above. We assume that the h-MOLP (3.1) is nontrivial - we obtain alternative optimal solutions while solving the underlying LPs hierarchically, and objectives to those LPs conflict with each other.

In this chapter, we study the challenges in methods 3.2 and 3.3. We describe the reasons for preferring method 3.3, the variable-fixing rule over method 3.2, and the constraint-addition rule in Section 3.2. In Section 3.3, we discuss some results on fixing of variables and provide a theoretical justification of equivalence of these lexicographic rules in their solving process. In Section 3.4, we highlight the issue of using reoptimization in both the lexicographic rules in their sequence of LP solves. Further, to address this, we introduce a concept of 'similarity' between LPs. This idea of similarity develops into a new lexicographic technique, which we call *SimLex*. We discuss the algorithm and the implementation procedures in Section 3.5. We conclude the chapter with some computational results and the summary in Section 3.6.

3.2 Issues with Constraint-addition Rule

The constraint-addition rule, defined in method 3.2, a popular lexicographic method, follows the following steps: We start with computing the optimal value of the first LP, $y^1 = \{\min c^1 x \mid Ax = b,\ l \le x \le u\}$. To preserve the solution of the previously obtained solution, we solve the next immediate linear program with a newly added constraint, $c^1 x = y^1$. Each new problem adds one new constraint as iteration k goes from 1 to t. t is the last index of the LP, $\mathtt{LP}^{\mathtt{t}}$, we solve. We solve them in sequence until 1) we get a unique solution to the LP or 2) we solve LP with the last objective.

One major problem with this method is that it can require the solution of many linear programs to obtain just one optimal solution to the h-MOLP. There are industry problems with more than hundreds of business objectives. In such cases, it is computationally expensive. Another disadvantage is in the underlying sequence of LPs it solves. After every

LP solve, it requires modification in the problem by imposing an additional constraint to the constraint set. It led to modification in both the rhs and the interaction matrix. Since successive solutions of LPs in the lexicographic method update the interaction matrix, rhs vector, and cost vector, a careful measure is required to exploit reoptimization between the LPs. Many solvers allow us to save the solution basis to use it for providing a starting solution for solving the other similar problem. Hot-start, one of the reoptimization (discussed in Chapter 1), may not always help due to such modifications in the explicit constraint-set between LPs. It makes the method overall computationally expensive and numerically sensitive than the variable-fixing rule. In the variable-fixing rule, changes between LPs are not the explicit constraint-set but bounds on variables. We ran some experiments and noticed this issue. We will describe the experiments and report the result in Section 3.6.

Now we provide an example to show that adding constraints generally makes the problem more sensitive and ill-posed than updating the variable bounds.Let us consider a toy example (3.4) with two objective vectors, $c^1 := (-0.333333, -0.666667)$ and $c^2 := (1, 1)$, with c^1 being more important than c^2.

$$\begin{aligned} &\texttt{lexmin}\ -0.333333\,x_1 - 0.666667\,x_2,\ x_1 +\ x_2 \\ &\qquad c1 : x_1 + 2x_2 = 3, \\ &\text{Bounds} \\ &\qquad 0 \le x_1 \le 2, \\ &\qquad 0 \le x_2 \le 2. \end{aligned} \tag{3.4}$$

For comparison purposes, we solve the problem using constraint-addition and variable-fixing methods. For both the methods, we start with solving $\mathtt{LP}^1$, an LP with objective vector c^1. The objective function value of $\mathtt{LP}^1$ we get is - 1.000000006. The constraint-addition rule then solve the next LP, $\mathtt{LP}^2$, after adding $-0.333333\ x_1 - 0.666667\ x_2 = -1.000000006$ to $\mathtt{LP}^1$.

$$\begin{aligned} \mathrm{LP}^1 :=\ & \min\,(-0.333333\ x_1 - 0.666667\ x_2) \\ & \text{Subject to} \\ & \quad c1 : x_1 + 2x_2 = 3, \\ & \text{Bounds} \\ & \quad 0 \le x_1 \le 2 \\ & \quad 0 \le x_2 \le 2 \end{aligned}$$

Similarly, variable-fixing rule checks the solution of $\mathtt{LP}^1$ and update the bounds of the variables with nonzero reduced cost. We find a positive reduced cost with value 0 of the variable x_1. The next LP we solve after setting x_1 to 0 is `modLP`^{2}. The optimal objective value for both the `LP`^{2} and `modLP`^{2} we obtained is 1.5.

$$\begin{aligned} \text{LP}^2 := \ & \min(x_1 + x_2) \\ & \text{Subject to} \\ & c1 : x_1 + 2x_2 = 3 \\ & c2 : -0.333333\, x_1 - 0.666667\, x_2 = -1.000000006 \\ & \text{Bounds} \\ & 0 \le x_1 \le 2, \\ & 0 \le x_2 \le 2. \end{aligned}$$

$$\begin{aligned} \text{modLP}^2 := \ & \min(x_1 + x_2) \\ & \text{Subject to} \\ & c1 : x_1 + 2x_2 = 3, \\ & \text{Bounds} \\ & 0 \le x_1 \le 0, \\ & 0 \le x_2 \le 2. \end{aligned}$$

In `LP`^{2}, changing the rhs of constraint $c2$ from -1.000000006 to -1.0 leads to a change in the optimal objective value from 1.5 to 2. It indicates that a slight change in the input can result in a significant change in the computed solution of the model. The degree to which an LP is ill-posed is generally decided by *condition number*. Renegar derived the expression for the condition number and introduced the term *ill-posedness*[119, 120]. An LP is ill-posed if it can be made both feasible and infeasible by arbitrarily small changes to its data of a linear program. In our case, the CPLEX solver returns the condition numbers (also called kappa value) 6.7e+8 and 1.0e+0 for `LP`^{2} and `modLP`^{2}, respectively. This high kappa value can cause numerical issues in the quality of the solution, such as 1) inconsistent results when presolving and input parameters are tuned 2) inaccuracy in the computed solution that contradicts the constraints in the model. [121].

The following two main reasons that motivated us to analyze the variable fixing rule further are: 1) The ill-posed behavior in the constraint-addition rule and a low kappa value in the example of the variable-fixing rule. 2) In general, no change in the basis matrix in two consecutive LPs in the variable-fixing rule. In the next section, we study linear programs with bounded variables and analyze some results for optimality using reduced cost information. The results help further to prove the equivalence between the variable-fixing rule and constraint-addition rule.

3.3 Linear Program with Bounded Variables and Fixing of Variables

Let us consider the following linear programming problem such that each decision variable is bounded below by a finite number:

$$LPP := \min_{x\in\mathbb{R}^n}\{c^T x \ : \ x \in S\}, \tag{3.5}$$

where

$$S := \{x \in \mathbb{R}^n \ : \ Ax = b, \ l \le x \le u\}. \tag{3.6}$$

Here, we assume that the real matrix $A \in \mathbb{R}^{m\times n}$ is of rank m. Moreover, $l_i < u_i$ for each $i \in [n]$. Now we define the basic solution and basic feasible solution of S.

Definition 3.1. *Let $S := \{x \in \mathbb{R}^n \ : \ Ax = b, \ l \le x \le u\}$ be a polyhedron as described above, and let $x^* \in \mathbb{R}^n$.*

(a) x^* is a basic solution if:

(i) all equality constraints are active

(ii) out of the constraints that are active at x^*, there are n of them that are linearly independent.

(b) if a basic solution satisfies all of the constraints, then it is called a basic feasible solution

Notice that $x^* \in S$ is a basic feasible solution if and only if it is an extreme point of S. Now we discuss the necessary condition for a basic solution.

Theorem 3.2. *Let $S := \{x \in \mathbb{R}^n \ : \ Ax = b, \ l \le x \le u\}$ be a polyhedron as described in equation (3.6). $x^* \in \mathbb{R}^n$ is a basic solution if and only if $Ax^* = b$, and there exist an index set $I_B \subseteq [n]$ of cardinality m such that:*

(a) The columns A_i, $i \in I_B$ are linearly independent.

(b) if $i \notin I_B$, then either $x_i^* = l_i$ or $x_i^* = u_i$.

Proof. Follows from the [24, Definition 2.9 and Exercise 3.25]. □

If x^* is a basic solution of S, the variables x_i^*, $i \in I_B$ are called the basic variables, and the remaining variables are called the nonbasic variables. The columns A_i, $i \in I_B$ are called the basic columns, and the remaining columns are called the nonbasic columns.

The m basic columns written adjacent to each other, form a matrix, and is called the basis matrix B. Let $I_{N_1} := \{i \in [n] \ : \ x_i^* = l_i\}$ be the index set associated with the nonbasic variables at their lower bounds, and $I_{N_2} := \{i \in [n] \ : \ x_i^* = u_i\}$ be the index set associated with the nonbasic variables at their upper bounds. The columns $A_i, \ i \in I_{N_1}$ are the nonbasic columns associated with the index set I_{N_1}, and the columns $A_i, \ i \in I_{N_2}$ are the nonbasic columns associated with the index set I_{N_2}. The matrix associated with the nonbasic columns $A_i, \ i \in I_{N_1}$ is denoted by N_1, and the matrix associated with the nonbasic columns $A_i, \ i \in I_{N_2}$ is denoted by N_2.

Let x^* be a basic solution of S, and let B be an associated basis matrix. By representing the index set $[n]$ as $I_B \cup I_{N_1} \cup I_{N_2}$, one can partition the matrix A into $[B, N_1, N_2]$, the decision variable x^T into $[x_B^T, \ x_{N_1}^T, \ x_{N_2}^T]$, the basic solution x^{*^T} into $[x_B^{*^T}, \ x_{N_1}^{*^T}, \ x_{N_2}^{*^T}]$, and the cost vector c^T into $[c_B^T, \ c_{N_1}^T, \ c_{N_2}^T]$.

Now, we provide a result on conditions for x_i^* to be optimal solution of a given objective function over S. The condition requires reduced cost information of variables. Let us define the reduced cost in the case of bounded LP.

Definition 3.3. *Let x^* be a basic solution of S. Let B be an associated basis matrix, and let c_B be the vector of costs associated with the basic variables. The reduced cost $\bar{c}_i$ for each $i \in [n]$ is defined as:*

$$\bar{c}_i := c_i - c_B^T B^{-1} A_i$$

Theorem 3.4. *Consider the linear programming problem as presented in equation (3.5). Let x^* be a basic feasible solution of S. Let B be an associated basis matrix, and let $\bar{c}$ be the associated vector of reduced costs. Assume that $\bar{c}_i \geq 0$ for all $i \in N_1$ and $\bar{c}_i \leq 0$ for all $i \in N_2$. Then, x^* is an optimal solution.*

Proof. We will establish that $c^T x^* \leq c^T y$ for all $y \in S$. Let $y \in S$, and let $d := y - x^*$. From $Ax^* = Ay = b$ we have $Ad = 0$. As $Ad = Bd_B + \sum_{i \in N_1 \cup N_2} A_i d_i$, we have

$$d_B = - \sum_{i \in N_1 \cup N_2} B^{-1} A_i d_i.$$

Now,

$$
\begin{aligned}
& c^T d \\
&= c_B^T d_B + \sum_{i \in N_1 \cup N_2} c_i d_i \\
&= c_B^T \Big(- \sum_{i \in N_1 \cup N_2} B^{-1} A_i d_i\Big) + \sum_{i \in N_1 \cup N_2} c_i d_i
\end{aligned}
$$

$$= \sum_{i \in N_1 \cup N_2} (c_i - c_B^T B^{-1} A_i) d_i$$

$$= \sum_{i \in N_1} \bar{c}_i d_i + \sum_{i \in N_2} \bar{c}_i d_i$$

For $i \in N_1$, we have $d_i = y_i - x_i^* = y_i - l_i \geq 0$. This implies that $\bar{c}_i d_i \geq 0$. For $i \in N_2$, we have $d_i = y_i - x_i^* = y_i - u_i \leq 0$, which implies that $\bar{c}_i d_i \geq 0$. As a result, $c^T d \geq 0$, completing the proof. □

3.3.1 Fixing of Variables Using Reduced Costs

We recall the steps in the variable-fixing rule for solving the h-MOLP. The maximum number of LP we solve is the number of objective vectors provided in the model - one LP associated with each objective. After solving each LP, we collect the solution basis status and associated reduced cost for every variable. Considering $y^1 = \min\{{c^1}^T x \mid Ax = b, l \leq x \leq u\}$, solution of highest ordered linear program (modLP^1) in model. For any variable $x_j \in x$, with the available bounds information $l_j \leq x_j \leq u_j$. Suppose we have its basis status information and reduced cost, $\overline{c_j}$ available in the solution. If we find that x_j is nonbasic and at its lower bound and, its reduced cost is positive then we fix x_j with the value l_j by making the modification of the upper bound of $x_j \leq u_j$ to $x_j \leq l_j$ in the solution of the LP with second highest ordered objective (LP^2). Similarly, if we find that x_j is nonbasic and at its upper bound with a negative reduced cost, we fix x_j with the value u_j by making a modification in the lower bound of $l_j \leq x_j$ to $u_j \leq x_j$ in modLP^2. We follow a similar procedure in solving a sequence of LPs.

The above steps follow the result discussed below. It also gives a theoretical justification of variable fixing and the equivalent rule of constraint-addition.

Definition 3.5. *A basis matrix is said to be optimal if $\bar{c}_i \geq 0$ for all $i \in N_1$ and $\bar{c}_i \leq 0$ for all $i \in N_2$.*

Theorem 3.6. *Let $LP := \min_{x \in \mathbb{R}^n} \{c^T x \ : \ Ax = b, \ l \leq x \leq u\}$ be a linear programming problem, where $A \in \mathbb{R}^{m \times n}$ is a real matrix of rank m, and $l_i < u_i$ for each $i \in [n]$. We assume that the feasible set $S := \{x \in \mathbb{R}^n \ : \ Ax = b, \ l \leq x \leq u\}$ is non-empty. Let B be an optimal basis for problem LP. Let x^* and $\bar{c} := c_B^T B^{-1} A$ be the optimal solution and the reduced cost vector associated with B. Then the following are true.*

(i) $F := \{x \in S \ : \ c^T x = c^T x^*\}$, the set of optimal solutions of LP, is a face of S.

(ii) F can be represented as

$$S \cap \{x \in \mathbb{R}^n \ : \ x_i = l_i \ \ \forall i \in [n] \ : \ \bar{c}_i > 0\} \cap \{x \in \mathbb{R}^n \ : \ x_i = u_i \ \ \forall i \in [n] \ : \ \bar{c}_i < 0\}$$

(iii) In particular, $F = \{x \in \mathbb{R}^n \; : \; Ax = b, \; \tilde{l} \le x \le \tilde{u}\}$,

where $\tilde{l}_i, \; \tilde{u}_i$ *for* $i \in [n]$ *is defined as:*

$$\tilde{l}_i := \begin{cases} u_i, & \text{if } \bar{c}_i < 0 \\ l_i, & \text{otherwise,} \end{cases}$$

and

$$\tilde{u}_i := \begin{cases} l_i, & \text{if } \bar{c}_i > 0 \\ u_i, & \text{otherwise.} \end{cases}$$

Proof. (i) Note that x^* is an optimal solution of LP. This means that

$$F = \text{ arg min }_{x \in S} \; c' x$$

In other words, F is the set of optimal solutions of LP. Moreover, it is straightforward to verify that F is a face of S.

(ii) As B is an optimal basis for problem LP and x^* is the optimal solution of LP associated with B, we have $x^*_{N_1} = l_{N_1}, \; x^*_{N_2} = u_{N_2}$, and $x^*_B = B^{-1}b - B^{-1}N_1 l_{N_1} - B^{-1}N_2 u_{N_2}$. Now,

$$c^T x^* = c_B^T x_B^* + c_{N_1}^T x_{N_1}^* + c_{N_2}^T x_{N_2}^*,$$

$$\Rightarrow c^T x^* = c_B^T \Big(B^{-1}b - B^{-1}N_1 l_{N_1} - B^{-1}N_2 u_{N_2}\Big) + c_{N_1}^T l_{N_1} + c_{N_2}^T u_{N_2}$$

$$\Rightarrow c' x^* = c_B^T B^{-1} b + \bar{c}_{N_1}^T l_{N_1} + \bar{c}_{N_2}^T u_{N_2}$$

Let

$$R := \{x \in \mathbb{R}^n \; : \; l \le x \le u\}$$

$$E := \{x \in \mathbb{R}^n \; : \; Ax = b, \; c^T x = cx^*\}.$$

So,

$$F = E \cap R$$

By applying a suitable row operation on E we have

$$E = \{x \in \mathbb{R}^n \; : \; Ax = b, \; (c^T - c_B^T B^{-1} A)x = c^T x^* - c_B^T B^{-1} b\}.$$

Substituting the value of $c^T x^*$ we have

$$E = \{x \in \mathbb{R}^n \ : \ Ax = b, \ \bar{c}^T x = \bar{c}^T_{N_1} l_{N_1} + \bar{c}^T_{N_2} u_{N_2}\}.$$

Recall that $\bar{c}_B = 0_{m\times 1}$. Hence,

$$E = \{x \in \mathbb{R}^n \ : \ Ax = b, \ \bar{c}^T_{N_1} x_{N_1} + \bar{c}^T_{N_2} x_{N_2} = \bar{c}^T_{N_1} l_{N_1} + \bar{c}^T_{N_2} u_{N_2}\}.$$

As $F = E \cap R$, we have

$$F = \{x \in S \ : \ \bar{c}^T_{N_1} x_{N_1} + \bar{c}^T_{N_2} x_{N_2} = \bar{c}^T_{N_1} l_{N_1} + \bar{c}^T_{N_2} u_{N_2}\} \tag{3.7}$$

Now, we will complete Theorem 3.6(ii) by using six logical equivalent steps. Let $y \in F$. The first equivalence follows from equation (3.7). Recall that $\bar{c}_i \geq 0$ for all $i \in I_{N_1}$ and $\bar{c}_i \leq 0$ for all $i \in I_{N_2}$. This implies the second equivalence. As $\bar{c}_i = 0$ for $i \in I_B$, we have $\{i \in N_1 \ : \ \bar{c}_i > 0\} = \{i \in [n] \ : \ \bar{c}_i > 0\}$ and $\{i \in N_2 \ : \ \bar{c}_i < 0\} = \{i \in [n] \ : \ \bar{c}_i < 0\}$. As a result, the third equivalence follows. Let $\alpha_i := y_i - l_i$ for all $i \in [n] \ : \ \bar{c}_i > 0$, and $\beta_i := y_i - u_i$ for all $i \in [n] \ : \ \bar{c}_i > 0$. The fourth equivalence follows from the definition of α_i and β_i. After simplification, we obtain the fifth equivalence. Recall that $l_i \leq y_i \leq u_i$ for all $i \in [n]$. This means that $\alpha_i \geq 0$ for all $i \in [n] \ : \ \bar{c}_i > 0$, and $\beta_i \leq 0$ for all $i \in [n] \ : \ \bar{c}_i < 0$.

$$y \in F$$

$$\Leftrightarrow y \in S \text{ and } \sum_{i \in I_{N_1}} \bar{c}_i y_i + \sum_{i \in I_{N_2}} \bar{c}_i y_i = \sum_{i \in I_{N_1}} \bar{c}_i l_i + \sum_{i \in I_{N_2}} \bar{c}_i u_i$$

$$\Leftrightarrow y \in S \text{ and } \sum_{i \in I_{N_1} \ : \ \bar{c}_i > 0} \bar{c}_i y_i + \sum_{i \in I_{N_2} \ : \ \bar{c}_i < 0} \bar{c}_i y_i = \sum_{i \in I_{N_1} \ : \ \bar{c}_i > 0} \bar{c}_i l_i + \sum_{i \in I_{N_2} \ : \ \bar{c}_i < 0} \bar{c}_i u_i$$

$$\Leftrightarrow y \in S \text{ and } \sum_{i \in [n] \ : \ \bar{c}_i > 0} \bar{c}_i y_i + \sum_{i \in [n] \ : \ \bar{c}_i < 0} \bar{c}_i y_i = \sum_{i \in [n] \ : \ \bar{c}_i > 0} \bar{c}_i l_i + \sum_{i \in [n] \ : \ \bar{c}_i < 0} \bar{c}_i u_i$$

$$\Leftrightarrow y \in S \text{ and } \sum_{i \in [n] \ : \ \bar{c}_i > 0} \bar{c}_i (l_i + \alpha_i) + \sum_{i \in [n] \ : \ \bar{c}_i < 0} \bar{c}_i (u_i + \beta_i) = \sum_{i \in [n] \ : \ \bar{c}_i > 0} \bar{c}_i l_i + \sum_{i \in [n] \ : \ \bar{c}_i < 0} \bar{c}_i u_i$$

$$\Leftrightarrow y \in S \text{ and } \sum_{i \in [n] \ : \ \bar{c}_i > 0} \bar{c}_i \alpha_i + \sum_{i \in [n] \ : \ \bar{c}_i < 0} \bar{c}_i \beta_i = 0$$

$$\Leftrightarrow y \in S \cap \{x \in \mathbb{R}^n \ : \ x_i = l_i \ \forall i \in [n] \ : \ \bar{c}_i > 0\} \cap \{x \in \mathbb{R}^n \ : \ x_i = u_i \ \forall i \in [n] \ : \ \bar{c}_i < 0\}$$

(iii) From the definition of $\tilde{l}$ and $\tilde{u}$ Theorem 3.6(iii) is true.

□

The constraint-addition rule always offers Pareto optimal solutions [14]. From the equivalence result in Theorem 3.6, we can infer that the variable-fixing rule will also provide Pareto optimal solution. However, likewise constraint-addition rule, variable-fixing rule can also require the solution of many single objective problems to obtain just one solution point. For relatively large h-MOLPs, for example, the master production schedule (MPS) in the manufacturing industries with a large-sized constraint set and many business objectives, it becomes a challenge. To speed it up, many solvers [27, 122] provide a feature of using the solution of high priority objectives as a starting solution to solve the low priority objectives. We call it reoptimization [46]. We have discussed reoptimization and Pareto optimality in Chapter 1. It is used to solve a new mathematical model by applying the available solution of a similar model with slight modification to the new model. This modification can be in rhs vector, cost vector, bounds of variables or coefficient matrix. Consider any two consecutive linear problems solved in the variable-fixing method (3.3):

$$\begin{aligned} \min\ & {c^p}^{\mathrm{T}} x \\ \text{s.t.}\ & Ax = b, \\ & x_j = f_j\ \forall j \in J^p \subseteq [n], \\ & l \leq x \leq u. \end{aligned} \quad (\texttt{modLP}^{\texttt{p}}) \qquad \text{and} \qquad \begin{aligned} \min\ & {c^q}^{\mathrm{T}} x \\ \text{s.t.}\ & Ax = b, \\ & x_j = f_j\ \forall j \in J^q \subseteq [n], \\ & l \leq x \leq u. \end{aligned} \quad (\texttt{modLP}^{\texttt{q}})$$

Where $x_j = f_j$ indicate the variable is fixed with some value using the variable-fixing rule (See Theorem 3.6). Also, $J^p \subset J^q$ and optimal solution of the problem ($\texttt{modLP}^{\texttt{p}}$) is a basic feasible to the problem ($\texttt{modLP}^{\texttt{q}}$) (see Theorem 3.6(i)). Though we can use optimal basis of problem ($\texttt{modLP}^{\texttt{p}}$) as starting feasible basis for solving the problem ($\texttt{modLP}^{\texttt{q}}$), sometimes it is better to avoid this available starting solution and start afresh. We can provide a motivating example in the next section and introduce the modification in the variable-fixing method by using the concept of similarity. This similarity will help us in selectively calling the reoptimization.

3.4 Notion of Similarity and SimLex

Reoptimization does not always help. There are instances where it is better to avoid the available starting solution and start afresh. As a motivation, we provide the following example:

$$\begin{aligned} \mathbf{LP_1} := \min_{x\in\mathbb{R}^2}\ & x_2 \\ \text{subject to} \\ 3 \le x_1 + x_2 &\le 9, \\ -3 \le x_1 - x_2 &\le 3, \\ x_1 + 2x_2 &\le 13, \\ x_1 - 2x_2 &\ge -7. \end{aligned} \qquad \text{and} \qquad \begin{aligned} \mathbf{LP_2} := \min_{x\in\mathbb{R}^2}\ & -x_2 \\ \text{subject to} \\ 3 \le x_1 + x_2 &\le 9, \\ -3 \le x_1 - x_2 &\le 3, \\ x_1 + 2x_2 &\le 13, \\ x_1 - 2x_2 &\ge -7. \end{aligned}$$

The problem $\mathbf{LP_1}$ and $\mathbf{LP_2}$ have unique optimal solutions at $p^* = (3,0)$ and $q^* = (3,5)$ respectively. Moreover, to solve the problem $\mathbf{LP_2}$ by simplex method, if we use the starting basic feasible solution as p^*, it will take more iterations to reach q^* than we start with any other basic feasible solution of $\mathbf{LP_2}$. We see the benefit of solving $\mathbf{LP_2}$ from scratch using both the primal and dual simplex methods.

A similar example mentioned in Appendix B consists of a h-MOLP and two consecutive LPs, LP^a and LP^b, generated while solving the h-MOLP using the constraint-addition rule. The total number of iterations in solving LP^a is 5. The number of iterations in solving LP^b with and without using the solution basis information of LP^a is 9 and 2. Here also, solving LP^b with the available solution basis of LP^a is more expensive in terms of the number of iterations than solving LP^b from scratch. Section 3.6 will report the results of an experiment where the available optimal basis helps speed up the overall solving time in some cases. However, in other cases, solving from scratch is helpful. We need to have a rule that selectively chooses the solution basis. In the next Section, we devise a strategy that exploits the structure of the underlying hierarchical model by monitoring the input parameter changes and leveraging reoptimization. Towards this end, we define a similarity measure between intermediate LPs appearing while solving the model.

3.4.1 Notion of Similarity between Linear Programs

We ideally call an LP, say LP^1 is similar to another LP, say LP^2, if the solution information of LP^1 helps solve LP^2 faster than just solving it from scratch. Alternatively, we can say two LPs are similar if reoptimization between them is helpful. This section uses some criteria to check whether the two LPs are similar. For this, we consider they differ only by the cost vector, c, and bound vectors, l, u. We formally define the notion of similarity as follows:

Definition 3.7. *Let* $LP^1 := \min\{c^1x \mid Ax = b,\ l^1 \le x \le u^1\}$ *and* $LP^2 := \min\{c^2x \mid Ax = b,\ l^2 \le x \le u^2\}$ *are two linear programs. We are assuming that the feasible sets of both* LP^1 *and* LP^2 *are non-empty.*

1. *Let p be the optimal solution of problem LP^1, and let B be the optimal basis associated with p. So, the reduced cost $\overline{c^1} := c^1 - c^1_B B^{-1} A$ satisfies three properties:*
   (a) $\overline{c^1_i} = 0$, $i \in B(1), \cdots, B(m)$,
   (b) $\overline{c^1_i} \geq 0$, $i \in L$ *and,*
   (c) $\overline{c^1_i} \leq 0, i \in U$.

   Here, $c^1_B := (c^1_{B(1)}, \ldots, c^1_{B(m)})$ and L and U are two disjoint index sets that partition the set of all $j \neq B(1), \cdots, B(m)$ such that $p_j = l_j$, $j \in L$ and $p_j = u_j$, $j \in U$.

2. *Let $e := c^2 - c^2_B B^{-1} A$. We refer e as the estimated reduced cost of LP^2.*

We wish to solve LP^2 using p and B^{-1}, solution information of LP^1. We say LP^1 and LP^2 are "similar" if the following conditions are satisfied:

i. *LP^1 and LP^2 have "similar-objective", i.e.,*

$$\left[1 - \left(\frac{\sum_{j=1}^{n} I_{p_j} |c^2_j - c^1_j|}{\sum_{j=1}^{n} |c^2_j - c^1_j|}\right)\right] \geq \kappa^1,$$

where $0 \leq \kappa^1 \leq 1$ is the given parameter. Here I_{p_j}, an indicator function, is defined as follows:

$$I_{p_j} = \begin{cases} 1, & \textit{if } e_j \leq 0 \textit{ and } \overline{c^1_j} > 0, \\ 1, & \textit{if } e_j \geq 0 \textit{ and } \overline{c^1_j} < 0, \\ 1, & \textit{if } e_j \neq 0 \textit{ and } \overline{c^1_j} = 0, \\ 0, & \textit{Otherwise}. \end{cases}$$

ii. *LP^1 and LP^2 have "similar-bounds", i.e.,*

$$\left[1 - \left(\frac{\sum_{i=1}^{n} \tilde{I}_{p_i} min(|l^2_i - p_j|, |u^2_i - p_j|)}{\sum_{i=1}^{n} min(|l^2_i - p_j|, |u^2_i - p_j|)}\right)\right] \geq \kappa^2,$$

where $0 \leq \kappa^2 < 1$ is the given parameter. Here $\tilde{I}_{p_i}$, the indicator function, is equal to 1 if the i^{th} component of p is outside the variable bound $[l^2_i, u^2_i]$, else is set to 0.

If the above conditions are satisfied, we use p as the starting basic feasible solution for solving LP^2. Otherwise, we will start solving from scratch. To avoid the difficulty of considering objective functions and variable bounds for the similarity computation simultaneously, we study the conditions i. and ii. separately. In condition i., we assume $l^1 = l^2$ and $u^1 = u^2$ when computing the similarity of objectives. Similarly, in condition ii., we define the similarity measure between bounds assuming $c^1 = c^2$.

Parameters κ^1 and κ^2 play a key role in similarity computation. Their values range from zero to one. A value near one assumes two LPs to be dissimilar. A low value, near zero, always leads to reoptimization. Threshold parameters are sensitive, and their ideal values are the one that helps in solving LP^2 faster by providing an appropriate decision of whether the solution basis obtained from LP^1 helps.

Now we provide the logical explanation for "similar-objective" and "similar-bounds", conditions used for obtaining "similarity" between LP^1 and LP^2 given the optimal solution p of LP^1. At optimality, solution basis B, and reduced cost $\overline{c^1} = c^1 - c^1_B B^{-1} A$ obtained after solving LP^1, hold the following optimality conditions

1. $c^1_j - c^1{}_B B^{-1} A_j \geq 0$ for all nonbasic indices $j \in L$ and,
2. $c^1_j - c^{1T}_B B^{-1} A_j \leq 0$ for all nonbasic indices $j \in U$.

p and B can also be feasible for LP^2 if the indicator function $\tilde{I}_{p_i}$ is zero for all $i \in \{1, \ldots, n\}$. This forms the "similar-bounds" condition between LP^1 and LP^2 for the given solution p. Similarly for optimality conditions to hold true for LP^2 given p , the following result forms the "similar-objective" condition:

Proposition 3.8. *Suppose $LP^1 = \min\{c^1 x \mid Ax = b,\ l^1 \leq x \leq u^1\}$ is a linear program. Consider p, B and $\overline{c^1}$ are the optimal solution, solution basis and reduced costs information respectively of LP^1. Consider a perturbed cost vector $c^2 = c^1 + \delta$ where $\delta \neq 0$ is given. Let $e = c^2 - c^2_B B^{-1} A$. For any component $p_i, i \in \{1, \ldots, n\}$, if $\overline{c^1_i} \cdot e_i \leq 0$ and, $\overline{c^1_i}$ and e_i both can not together be zero, the basis B of LP^1 is not the optimal basis of LP^1 with perturbed cost vector c^2.*

Proof. Assume that the set of all $i \notin B(1), \ldots, B(m)$, is partitioned into two disjoint subsets L and U such that $x_j = l_j$ for all $j \in L$ and $x_j = u_j$ for all $j \in U$. Clearly, for B be the optimal basis of LP^1, 1) $\overline{c^1_j} \geq 0,\ j \in L$ and 2) $\overline{c^1_j} \leq 0,\ j \in U$ (see 3.4). If for any $j \in L$, reduced cost of x_j with perturbed cost coefficient c^2_j, $e_j \leq 0$ then increasing x_j will help improving the objective function value, making it a potential candidate to enter the basis, leading B no longer be optimal. Similarly, if for any $j \in U$ $e_j \geq 0$ then decreasing x_j may help improving the objective function value, x_j can become a potential candidate to enter the basis, leading B no longer be optimal basis. It implies that for any variable $x_i, i = 1, 2, \cdots n$, if $e_j \cdot \overline{c^1_j} \leq 0$, the basis B of LP^1 is not the optimal basis of LP^2. □

3.5 Implementation

We now adopt the concept of similarity for solving the hierarchical model (3.1). This idea provides us an improved version of variable-fixing rule. We call it *SimLex* Recall the variable-fixing rule mentioned in Section 3.3.1, where the consecutive LPs solved only differ by the objective cost and the variable bound vectors. The solution obtained from the previous LP is always feasible for the current LP. It simplifies the definition of similarity as follows:

Definition 3.9. *Let $LP^1 := \min\{c^1x \mid Ax = b,\ l^1 \le x \le u^1\}$ and $LP^2 := \min\{c^2x \mid Ax = b,\ l^2 \le x \le u^2\}$ are two linear programs. We assume that the feasible sets of both LP^1 and LP^2 are non-empty. With respect to p and B, the optimal solution and the optimal basis of LP^1, we say LP^1 and LP^2 are "similar" if*

$$\left[1 - \left(\frac{\sum_{j=1}^{n} I_{p_j}|\, c_j^2 - c_j^1|}{\sum_{j=1}^{n} |\, c_j^2 - c_j^1|}\right)\right] \ge \kappa,$$

where $0 \le \kappa \le 1$ is the given parameter. I_{p_j} the indicator function is equal to 1 if $e_j \neq 0$ and $\overline{c_j^1} = 0$, else is set to 0.

If the above condition is satisfied, we use p as the starting basic feasible solution for solving LP^2. Otherwise, we will start from scratch.

Simplex procedure uses Definition 3.9 for selectively invoking reoptimization. Algorithm 5 briefs the procedures of as follows: We start with a feasible set and a list of objectives. We solve them hierarchically with the highest priority objective first. After solving each LP, we do a similarity check. We use the solution basis of the previous LP as starting basic feasible solution to the current LP if it is similar to the previously solved LP. Otherwise, we solve it from scratch. We update the current feasible set using the obtained solution similar to the variable-fixing rule.

Other than the information already present in memory while solving the current LP, the additional previous LP information consists of:

- an array of size n to store the cost coefficient,
- m arrays each of size m, for storing columns of B^{-1}, and
- an array of size n to store reduced cost information.

The first two pieces of information compute the estimated reduced cost, e of the current LP. Using e with the third information $\bar{c}$, we evaluate the similarity score. If B^{-1} is large, storing all the m arrays is expensive. However, we do not need all the columns to be stored. We only require b columns. It is the maximum number of non-zeros entries in any $A_j,\ j = 1, \dots, m$ in A. Let us consider the following steps to compute e for LP^k:

$$e_j = c_j^k - c_B^{k-1}[B^{-1}(1), \dots, B^{-1}(m)]A_j^k = c_j^k - c_B^{k-1}\sum_{i=1}^{m} B^{-1}(i)A_{ij}^k$$

$$= c_j^k - c_B^{k-1} \sum_{i=1,\ldots,\, m|\; A_{ij}^k!=0} B^{-1}(i)A_{ij}^k,$$

where $B^{-1}(1), \ldots, B^{-1}(m)$ are m columns of B^{-1} and $c_B^{k-1}(1), ldots, c_B^{k-1}(m)$ are the associated basic cost coefficients of LP^{k-1}. A_j^k and c_j^k are the corresponding j^{th} column and the associated cost coefficient of LP^k. In practice, LPs are generally sparse and, for such problems, $b << m$ make the implementation practical.

Algorithm 5: SimLex

```
Input: Feasible set $F := \{x \mid Ax = b,\ l \le x \le u\}$; List of $k$ objective vectors
       $[c^1, \ldots, c^k]$.
Output: List of $k$ solutions for each objectives, $S := [y^1, \ldots, y^k]$:
Solve LP$^1$ := $\min\{c^1 x \mid x \in F\}$ and store the its solution $y^1$ to $S$.
Update $F$ using the solution obtained from LP$^1$.
for $t = 2, \ldots, k$ do
    if $LP^t := \min\{c^t x \mid x \in F\}$ and $LP^{t-1}$ are similar (w.r.t $y^{t-1}$) then
        Solve LP$^t$ with starting solution information $y^{t-1}$ ;
    else
        Solve LP$^t$ from scratch. ;
    Save the current solution $y^t$ in $S$ ;
Return $S$ as the final solution to the given input problem.
```

3.6 Computational Result and Summary of the Work

In this section, we give empirical evidence of the effectiveness of our similarity-based method (SimLex) on h-MOLP instances and provide computational details of them. We compare the performance of SimLex with that of the standard lexicographic scheme in CPLEX (Default-CPLEX) from CPLEX-12.10.0, and two well-known rules, variable-fixing and constraint-addition. For all the experiments, we use Primal simplex method.All other settings, including presolves, heuristics, etc., were left undisturbed. Our subroutines are written in Python 3.7 and, for solving LPs, use python CPLEX API, a python package in CPLEX that allows the callable library to be accessed from the python language. The hardware used for the computation is a 64 bit Intel(R) Xeon (R) E5-2673 v4 at 2.30GHz CPUs with 20 cores and 64 GB RAM. To avoid multiple processes sharing common resources, we run one job at a time with the default settings of CPLEX.

Our experiment does not report objective values, as every test instance is solved within the given time limit from each hierarchical procedure. Problem instances for

Table 3.1: Performance summary of SimLex compared to other rules over various κ

				simlex									
instance	def-cplex	const-add	var-fix	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
molp_3_100_20_assignment	**14**	34	14	14	14	14	14	14	14	14	14	**13**	**13**
molp_4_729_729_bensolvehedron	**40**	**46**	143	143	143	143	143	143	143	143	143	143	143
molp_4_900_60_assignment	99	212	99	99	99	99	99	99	99	99	**97**	**97**	**97**
molp_9_100_60_mpp	**65**	81	77	77	77	77	77	77	77	81	84	90	99
molp_10_779_10174_entropy	42763	9300	**6527**	**6527**	**6527**	**6527**	**6527**	**6527**	**6527**	**6527**	23731	21902	21902
molp_10_900_60_assignment	99	441	99	99	99	99	99	99	99	99	**97**	**97**	**97**
molp_12_21_30_dc	**8**	18	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
molp_21_31_138_entropy	52	128	**35**	**35**	**35**	**35**	**35**	**35**	86	86	158	158	158
molp_22_43_213_entropy	113	160	**97**	**97**	**97**	**97**	**97**	**97**	100	100	219	219	219
molp_23_28_218_entropy	2	2313	3	3	3	3	3	3	1	4	**0**	**0**	**0**
molp_27_28_218_entropy	1	151	1	1	1	1	1	1	1	**0**	**0**	**0**	**0**
no. of times it performs better than others	4	1	4	4	4	4	4	4	2	3	5	**6**	**6**

our comparison consist of two different sets of h-MOLPs. In one experiment, we select the first set from MOPLIB, a problem library for multi-objective linear, multi-objective (mixed) integer and vector linear programs [123], where we choose 11 out of 15 available instances and exclude the trivial instances. The selected MOLPs instances are summarized in Table B.1.

We ran the Default-CPLEX, variables-fixing, and constraint-addition rules with SimLex on them. Instead of solving time, we report the total iterations taken to reach the optimal solution, as each instance takes a few seconds to solve using any lexicographic rules. Table 3.1 lists the total iterations taken by each procedure. The last row reports the winning count of SimLex with high κ values over others, concluding that solving LPs from scratch leads to a faster solving time.

The other set of instances is the h-MOLPs of master production schedules (MPS) specific to supply chain scenarios for some consumer products and goods (CPG) industries. We have discussed MPS in Chapter 1. Modeling of MPS is described in detail in the next Chapter 4. Table 3.2 summarizes 13 h-MOLPs modeled for MPS of 4 different supply chain scenarios. For each model, we set the time limit to 7200 seconds.

To compute an ideal κ value, we do the following experiment: We ran one MPS model from each supply chain scenario over the entire range of κ values. With $\kappa = 0.6$, we obtained the best mean solving time and set it as a default threshold value. Table 3.3 reports the solving time for four instances chosen from different supply chains. In the last column, SGM -50 shows the mean solving time. It uses shifted geometric mean (SGM)[117] with a shift of 50 seconds.

Table 3.4 compares the overall performance of our strategy to other procedures. The columns 'def-cplex', 'const-add', 'var-fix', and 'simlex' denote the Default-CPLEX rule, constraint-addition rule, variable-fixing rule, and SimLex procedure, respectively. We report the time taken to run the given model by all the rules. We also report the mean of

Table 3.2: Problem summary of h-MOLPs selected for the computational experiment

supply chain scenario	instance	no. of linear constraints	no. of variables	no. of nonzeros in linear constraints	no. of business objectives	no. of nonzeros in objective function
1	1	1218126	3406752	14987279	33	16322
	2	1342622	3909913	16442083	33	17117
	3	1340445	3897873	16431281	33	17136
2	4	1360885	3428617	33139223	40	981
	5	1356679	3429441	33652942	40	845
	6	1357075	3425452	33990980	40	871
3	7	717200	3992746	8728732	17	521
	8	680525	3808447	8313350	17	452
	9	536716	3198612	6921596	15	208
4	10	1166983	6682634	10941441	17	7735
	11	1252716	7195176	11750289	17	8689
	12	1212667	6940401	11348793	17	8839
	13	1329219	7128846	11994006	17	7585

Table 3.3: Solving times (in seconds) of selected MPS models for an ideal κ for SimLex

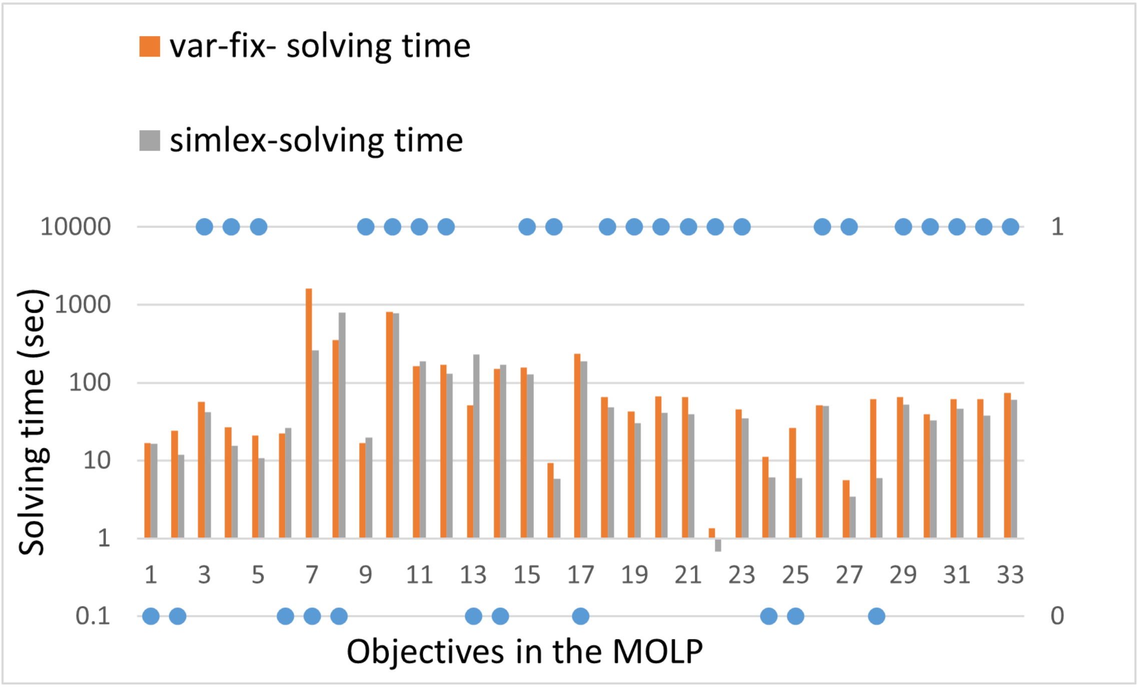
instance	κ								
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
1	4253	4184	4264	4209	4179	3501	4483	6690	6507
5	5222	5163	5103	5142	5142	1687	1639	1497	1519
8	148	147	148	149	145	284	273	268	277
12	422	423	417	429	438	418	402	310	310
SGM-50	1157	1147	1147	1154	1153	**941**	978	995	999

Table 3.4: Result summary of solving times (in sec) of SimLex over other rules

supply chain scenario	instance	def-cplex	const-add	var-fix	simlex	%age simlex benefit over the other best rule
1	1	4381	timeout	4136	**3501**	15
	2	4999	timeout	4659	**3456**	26
	3	6605	timeout	4770	**4025**	16
2	4	5488	timeout	5207	**1687**	68
	5	6441	timeout	6068	**1595**	74
	6	5759	timeout	6626	**1618**	72
3	7	147	505	**143**	284	-50
	8	153	656	**125**	255	-51
	9	**93**	timeout	94	161	-42
4	10	388	492	**376**	379	-1
	11	**425**	691	425	**418**	2
	12	430	856	**411**	421	-2
	13	502	636	**473**	533	-11
SGM-50		1143	2380	1079	**858**	
%		100		94	**75**	
no. of times performs better than others		2	0	5	**7**	

SimLex relative to the mean of def-cplex (the row % in the table). A value below 100 states an improvement over the default. We observed that 7 out of 13 instances, solved by SimLex, are faster than any other procedures (reporting in the bottom row). We also observed that Default-CPLEX performs equally well with the variable-fixing rule. The last column reports the performance of SimLex over other procedures in percentage. For each MOLP instance, a positive score indicates the percentage score by which SimLex solves faster than its best competing procedure. Similarly, a negative score indicates the percentage score by which SimLex performs slower than its best competing procedure. The row SGM-50 in the table shows that SimLex is the fastest for the given MPS models overall. We can also conclude the competitiveness of Default-CPLEX and variable-fixing rule over SimLex. For problem instance number 9, the Default-CPLEX rule beats SimLex by 42%. In some cases, SimLex gives a more than 70% percentage benefit.

To highlight the impact of selectively using available feasible solutions (with hot-start enabled) in the lexicographic solving procedures, we collect information for two specific instances, instance-2 from the supply chain scenario-1 and instance-5 from the supply chain scenario-2. Figure 3.1 reports it for both the variable-fixing-based rule and our SimLex procedure. It contains two sub-images, one for instance-2 and the other for



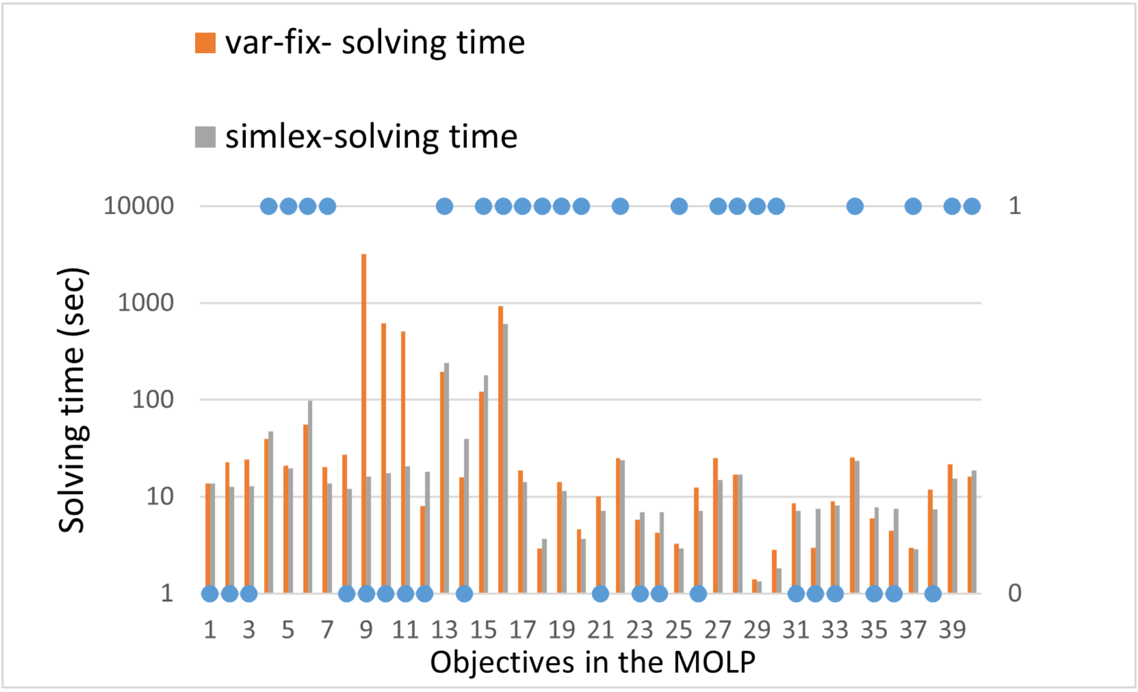


Figure 3.1: Instance -2 (left) and Instances -5 (right). Objective wise performance comparison of SimLex with variable-fixing. Blue points at level 1 indicate solving with hot-start and at level 0 indicate solving from scratch

instance 5. Each shows blue points positioned at level 0 or 1 for the objectives. At level 0, it indicates the respective objective function does not use hot-start and solves from scratch when solving using the SimLex rule. If the point is at level 1, it implies that the current object uses the solution information (with hot-start enabled) obtained from the last objective. Along with the blue point, we include 2 column bars that mention the time taken (in seconds) to solve the instance using the variable-fixing rule and SimLex. We observed that the selective hot-start approach in SimLex over the variable-fixing rule computationally helps.

In summary, we provide a new lexicographic method for hierarchical multiobjective linear programs. It uses input parameters in the model to decide whether the current LP should use the available feasible solution obtained from the previous LP. We apply this idea with two different sets of MOLPs - the first set of instances is chosen from MOPLIB, a library of benchmark multi objectives programs, and the second is the mathematical model of the master production schedules. We did not emphasize an ideal parameter selection procedure for similarity computation. The current selection procedure is specific to the given supply chain scenarios. An ideal parameter selection for the general h-MOLP and incorporation of the simultaneous sensitivity analysis with the concept of similarity are the two main works that need to be done.

Chapter 4

Master Production Schedule as h-MOLP

Master production schedule (MPS) is one of the main components of master planning in the study of supply chain planning. It prepares a detailed 'rough-cut' plan for individual products produced in each period on the planning horizon. We have discussed MPS in detail in Chapter 1, which provides a simple MPS example and approach to model it mathematically. It studies the basic supply chain restrictions, such as material flow from the manufacturing end to the ultimate customer, resource capacity and load constraints, and the essential demand-based objective of minimizing the unmet demand in the "multi-echelon" supply chain planning process. We recall the model (1.12) formulated as follows:

$$\begin{aligned}
\text{LP1:}\quad & \text{obj1:= } \min -xd^1_{t1} - xd^2_{t2} - xd^3_{t3} \\
& \text{subject to } \sum_{i \in O^1} op^i_{1,t} - c^1_t \le 0,\ t = 1, 2, \cdots, T, \\
& op^i_{1,1} - b^i_{1,1} = 0,\ i = 1, 2, 3, \\
& op^i_{1,2} + b^i_{1,1} - b^i_{1,2} = 0,\ i = 1, 2, 3, \\
& \qquad\vdots \\
& op^i_{1,ti} + b^i_{1,ti-1} - b^i_{1,ti} - xd^i_{ti} = 0,\ i = 1, 2, 3, \\
\text{bound:}\ & 0 \le xd^i_{ti} \le \widetilde{d^i_{ti}},\ i = 1, 2, 3, \\
& 0 \le c^1_t \le max\_c^1_t,\ t = 1, 2, 3, \cdots, T, \\
& op^i_{1,t},\ b^i_{1,t} \ge 0\ i = 1, 2, 3,\ \text{and}\ t = 1, 2, 3 \cdots, T. \qquad (4.1)
\end{aligned}$$

For the parameters and decision variables used, we refer to the Section 1.4.1 of Chapter 1. Throughout the chapter, we follow this model to explain the various objectives considered in MPS. MPS applies many business objectives, and most conflict with each other. It becomes even more challenging to model MPS considering all the objectives simultaneously. Some industries set priorities among the objectives. It helps the planner to obtain an acceptable Pareto solution by posing it to a lexicographic model. However, for many objectives solving the MPS lexicographically is computationally expensive. In Chapter 3, we have discussed such challenges and one technique to improve the computation effort. Section 3.1 discusses the literature on h-MOLP. The related literature on MPS is discussed in Section 1.4.1 of the Chapter 1.

This chapter will be more toward modeling MPS in detail, formulating various objectives used, and exploiting customers' input in MPS to improve the performance of the lexicographic technique for MPS. Mostly, satisfying a customer's demand is the highest priority objective in industries. We explore demand-based objectives created by prioritizing the given demand requirements or delivering the requested items in fractions. We call delivering the requested demand in fractions as a *fair-shared* demand. We will study such demand-based objectives most customers may prefer for MPS in Section 4.1. Combining objectives with a weighted-sum approach is a basic way to solve multiobjective programs (MOP). Unlike the lexicographic rule, the weighted-sum method avoids solving several single objective linear programs but faces challenges in obtaining a Pareto optimal solution. We discuss a technique to combine the objectives in the lexicographic method for MOPs. Under certain specific conditions, we find that the method works well with MPS. Further, we find that the fair-shared demand objectives can be combined while solving the MPS using lexicographic method. We discuss them in Section 4.2. Besides fair-shared demands, this section discusses combining the objective of minimizing the unmet demand (or maximizing the required demand) followed by minimizing *backlog*. This backlog (or lateness) objective aims to reduce the delay of those demand items that could not meet their due date. On the computational front, in Section 4.3, we discuss the benefit of our idea by implementing and running it for some industry datasets, comparing it with the standard lexicographic method, and concluding our contribution with some future directions. Finally, in Section 4.4, we explain the steps to model an MPS of a dummy manufacturing industry with a toy example, a *potato chip model*.

4.1 Popular Objectives used in MPS

4.1.1 Maximizing Meeting of Demand

An essential objective of any firm is to keep its customer happy. The demand for final item products is the customer's direct order and forecasted order from previous customers' orders and sales. MPS takes this demand order pictures as input and plans to meet the demands required on the due date with the highest priority. A planner defines the demand-based objective as per the input demand requirements. We can study some of them.

- The demand required for one type of product on a single due date: Such demand request is the simplest in terms of its modeling. Suppose the requirement for the finished product item $d1$ is $\tilde{d}^1_{t1}$ on the due date $t = t1$ day. We define the objective function as
$$\min -xd^1_{t1},$$
where xd^1_{t1} is the decision variable, defined as the total unit of item $d1$ a planner can satisfy the customer by due date $t = t1$. The range of xd^1_{t1} must be defined as $0 \leq xd^1_{t1} \leq \tilde{d}^1_{t1}$ as a trivial bound constraint. The material flow balance and resource load constraints will be similar to model 4.1.

- A customer requires one type of item on different dues dates: If the demand request of the finished product item $d1$, requested on $t = t1$ and $t2$ days, are $\tilde{d}^1_{t1}$ and $\tilde{d}^1_{t2}$, we can define the objective function as
$$\min -w1\ xd^1_{t1} - w2\ xd^1_{t2}$$
where xd^1_{t1} and xd^1_{t2} are the decision variables, defined as the unit of item $d1$ a planner can satisfy to the customer by due dates $t1$ and $t2$, respectively. The ranges of xd^1_{t1} and xd^1_{t2} are $0 \leq xd^1_{t1} \leq \tilde{d}^1_{t1}$ and $0 \leq xd^1_{t2} \leq \tilde{d}^1_{t2}$ as trivial bound constraints. The objective coefficients $w1$ and $w2$, positive real numbers, are set as per the required priority on due dates. If the demand requested on $t1$ is more important than that of $t2$, $w2 < w1$. In case both are of equal importance, we set $w1 = w2$.

- Demand requests are divided into several levels: There could be a situation where the planner cannot fully satisfy the requests. One problem may occur when customers request more than one type of item, and all those items are essential. Due to limited input capacity, the planner can only meet some of the requirements. The planner might meet all the requests of one type of item but could not another item at all. It is not a fair share. To meet this requirement, the planner must meet one level of a fraction of the request. After planning with this partial demand, the planner can similarly plan for the remaining demand request. Consider $\tilde{d}^1_{t1}$ and $\tilde{d}^2_{t2}$ are the demand request of item types $d1$ and $d2$ at due dates $t1$ and $t2$ days, respectively. To have a fair share of demand fulfillment, we can plan in more than one stage. At stage one, we try to meet the α fraction of the total items demanded of each item types. The objective function, in this case, will be
$$\min -w_1\ xd^{11}_{t1} - w_2\ xd^{21}_{t2},$$

where xd^{11}_{t1} is a decision variable, defined as the total items of type $d1$ a planner can satisfy the customer by due date $t = t1$. Similarly, we define xd^{21}_{t2} for item $d2$ at due date $t = t2$. w_1, w_2 are the weights associated with the demands variable, decided as per the importance of the demand variables. The bounds of xd^{11}_{t1} and xd^{21}_{t2} are $0 \le xd^{11}_{t1} \le \alpha\ \tilde{d}^1_{t1}$ and $0 \le xd^{11}_{t2} \le \alpha\ \tilde{d}^1_{t2}$ as a trivial bound constraints. We model the objective function for the other stage. For example, for stage two, the objective function

$$\min -w_1\ xd^{12}_{t1} - w_2\ xd^{22}_{t2},$$

with new decision variables, xd^{12}_{t1} and xd^{22}_{t1}, is defined similarly to xd^{11}_{t1} and xd^{21}_{t1}. The bounds of xd^1_{t1} and xd^1_{t2} are $0 \le xd^{11}_{t1} \le (1-\alpha)\beta\ \tilde{d}^1_{t1}$ and $0 \le xd^{11}_{t2} \le (1-\alpha)\beta\ \tilde{d}^1_{t2}$ as a trivial bound constraints. Here $\beta \in [0, 1]$ is the fraction item unit requested to be fulfilled at stage 2. If $\beta = 1$, we solve the fair share of demand requested in two stages. Otherwise, we can continue solving a partial demand satisfaction problem. Note that the weight vector is the same at each stage for each fair-shared objective function as the delivery (due) dates remain intact over the demand request.

If we implement a k-stage demand satisfaction objective with fair share demand fulfillment, model 4.1 reformulates to the following lexicographic objective.

$$\begin{aligned}
\text{k-DemandsMPS:}\quad & \text{lexmin } ((-w_1\ xd^{11}_{t1} - w_2\ xd^{21}_{t2} - w_3\ xd^{31}_{t3}),\\
& \qquad (-w_1\ xd^{12}_{t1} - w_2\ xd^{22}_{t2} - w_3\ xd^{32}_{t3}),\\
& \qquad \ldots, (-w_1\ xd^{1k}_{t1} - w_2\ xd^{2k}_{t2} - w_3\ xd^{3k}_{t3}))\\
\text{subject to}\quad & \sum_{i \in O^1} op^i_{1,t} - c^1_t \le 0,\ t = 1, 2, 3, \cdots, T,\\
& op^i_{1,1} - b^i_{1,1} = 0,\ i = 1, 2, 3,\\
& op^i_{1,2} + b^i_{1,1} - b^i_{1,2} = 0,\ i = 1, 2, 3,\\
& \vdots\\
& op^i_{1,ti} + b^i_{1,ti-1} - b^i_{1,ti} - xd^i_{ti} = 0,\ i = 1, 2, 3,\\
& -xd^i_{ti} + xd^{i1}_{ti} + xd^{i2}_{ti} + \cdots + xd^{ik}_{ti} = 0,\ i = 1, 2, 3,
\end{aligned} \tag{4.2}$$

$$\begin{aligned}
\text{bound: }& 0 \le xd^{i1}_{ti} \le \widetilde{d^{i1}_{ti}},\ i = 1, 2, 3,\\
& 0 \le xd^{i2}_{ti} \le \widetilde{d^{i2}_{ti}},\ i = 1, 2, 3,\\
& \vdots\\
& 0 \le xd^{ik}_{ti} \le \widetilde{d^{ik}_{ti}},\ i = 1, 2, 3,\\
& 0 \le c^1_t \le max\_c^1_t,\ t = 1, 2, 3, \cdots, T,
\end{aligned}$$

$$op^i_{1,t}, b^i_{1,t} \geq 0, i = 1, 2, 3, \text{ and } t = 1, 2, 3 \cdots, T. \tag{4.3}$$

Here $\widetilde{d^{i1}_{ti}}$, $\widetilde{d^{i2}_{ti}}, \ldots, \widetilde{d^{ik}_{ti}}$ are k proportion of demand requirement of $\widetilde{d^i_{ti}}$ unit of item type d_i on due date ti, for all $i = 1, 2, 3.$, such that $\widetilde{d^{i1}_{ti}} + \widetilde{d^{i2}_{ti}} + \cdots, \widetilde{d^{ik}_{ti}} = \widetilde{d^i_{ti}}$.

- Solve separate objectives, each with the demand requests of items with different priorities: If the customer provides the importance among the requested items, the weighted sum-based approach may lead to scaling issues if the measuring units have different scales. An ideal option would be to solve them separately, which guarantees Pareto optimality. We first meet the high priority demand, and then with the remaining availability, we try to meet the low priority demand required. The lexicographic objective function, in this case, will be as follows:

  $$\text{lexmin } (-w_{11}\ xd^1_{t1} - w12\ xd^1_{t2}, -w_{21}\ xd^2_{t3} - w_{22}\ xd^2_{t4}).$$

  Here xd^1_{t1} and xd^1_{t2} are the demand decision variables associated with item $d1$ with the respective due dates $t1$ and $t2$. Similarly, decision variables xd^2_{t3} and xd^2_{t4} are defined for item $d2$ with due dates $t3$ and $t4$, respectively. These variables are non-negative and are upper bounded by the customer's associated demand for requested items. Other, constraints will be similar to the model (4.1).

4.1.2 Avoiding Lateness

The last section that dealt with modeling aspects of meeting the demand requests did not consider the backlogging or lateness. If customers agree with some lateness in receiving the items they demanded, the planner tries first to meet the demand on the given due date, and if it is not possible to meet all of them on that date, it replenishes to the customer on the late date. In such scenarios, planners target to meet the unmet demand as close to the due date as possible to reduce tardiness.

We consider decision variables based on lateness concerning each demand requirement. For instance, f^1_{t1}, $f^1_{t1+1}, \ldots, f^1_T$ are the decision variables that indicate the production unit of item d^1 at $t1$, $t1 + 1$, to T associated with the demand-based decision variable $\widetilde{d^1_{t1}}$. Similarly, we define lateness variables f^2_{t2}, $f^2_{t2+1}, \ldots, f^2_T$ associated with item d^2, and f^3_{t3}, $f^3_{t3+1}, \cdots, f^3_T$ associated with item d^3, respectively. We define the lateness objective, a linear objective function, as $(1f^1_{t1} + 2f^1_{t1+1} + \cdots, +Tf^1_T) + (1f^2_{t2} + 2f^2_{t2+1} + \cdots, +Tf^2_T) + (1f^3_{t3} + 2f^3_{t3+1} + \cdots, +Tf^3_T)$. We assign weights as per the lateness from the target date - at T, weights are maximum and are set to a minimum at the due date. Since demand-based decision variables are associated with the lateness variables, we add them into the constraint set as follows: For demand item d_1, $\widetilde{d^1_{t1}} = f^1_{t1} + f^1_{t1+1} + \cdots + f^1_T$. Similarly, we have $\widetilde{d^2_{t2}} = f^2_{t2} + f^2_{t2+1} + \cdots + f^2_T$, and $\widetilde{d^3_{t3}} = f^3_{t3} + f^3_{t3+1} + \cdots + f^3_T$. for d_2 and d_3 respectively. The minimum and maximum values of lateness variables $f^1_{t1}, f^1_{t1+1}, \ldots, f^1_T$ will be the same as $\widetilde{d^1_{t1}}$. Similarly, we update the bounds of other lateness variables. Considering both the objectives, 1) minimizing unmet demand and 2) minimizing lateness, we have the following h-MOLP model :

$$
\begin{aligned}
\text{MPS:}\ \ \texttt{lexmin}\ & (-xd^1_{t1} - xd^2_{t2} - xd^3_{t3}),\ ((1f^1_{t1} + 2f^1_{t1+1} + \cdots, +Tf^1_T) \qquad (4.4)\\
& + (1f^2_{t2} + 2f^2_{t2+1} + \cdots, +Tf^2_T) + (1f^3_{t3} + 2f^3_{t3+1} + \cdots, +Tf^3_T))\\
& \text{s.t.}\ \sum_{i \in O^1} op^i_{1,t} - c^1_t \le 0,\ t = 1, \cdots, T,\\
& op^i_{1,1} - b^i_{1,1} = 0,\ i = 1,\ 2,\ 3,\\
& op^i_{1,2} + b^i_{1,1} - b^i_{12} = 0,\ i = 1,\ 2,\ 3,\\
& \vdots\\
& op^i_{1,ti} + b^i_{1,ti-1} - b^i_{1,ti} - xd^i_{ti} = 0,\ i = 1,\ 2,\ 3,\\
& -xd^i_{ti} + f^i_{ti} + f^i_{ti+1} + \cdots, +f^1_T = 0,\ i = 1,\ 2,\ 3,\\
\text{bound:}\ & 0 \le xd^i_{ti} \le \widetilde{d^i_{ti}},\ i = 1,\ 2,\ 3,\\
& 0 \le f^i_{ti} \le \widetilde{d^i_{ti}},\ i = 1,\ 2,\ 3,\\
& 0 \le f^i_{ti+1} \le \widetilde{d^i_{ti+1}},\ i = 1,\ 2,\ 3,\\
& \vdots\\
& 0 \le f^i_{ti+T} \le \widetilde{d^i_{ti+T}},\ i = 1,\ 2,\ 3,\\
& 0 \le c^1_t \le max\_c^1_t,\ t = 1,\ 2,\ 3, \cdots, T,\\
& op^i_{1,t}, b^i_{1,t} \ge 0,\ i = 1,\ 2,\ 3,\ \text{and}\ t = 1, \ldots, T. \qquad (4.5)
\end{aligned}
$$

4.1.3 Other Important Business Objectives

Apart from demand-related objectives, there are many business requirements customers expect to consider in the MPS computation under supply chain planning. Some of them we discuss in brief:

Minimizing the alternate operations to prefer primary operations

Unexpected demand requests from customers and limitations in manufacturing them targeted to be available on the specified due date require alternatives in many industries. An alternative in the industry is a backup, such as performing the alternate operation of bringing items far from the warehouse or producing at a higher cost than the regular production cost per item. The planner does not want to utilize items from alternatives unless necessary. The main reason to keep it a second priority is to avoid unnecessary expenses. An objective of minimizing alternate operations in MPS helps properly utilize material and resources in the industry and avoid alternate operations. Suppose there is a customer demand of 100 units of an item. A mathematical model for MPS runs to maximize the demand satisfaction (or minimize the unmet demand) and meet all the 100 units of demand requests on time. The second objective is to obtain a plan for minimizing the alternative operations in case 100 units of demand request could not meet. The objective will try to push as much as the demand requested meets from the primary source and then provide the demand from the alternatives. Like, alternate operations, other components, like alternate flow set, alternate BOM (bill of materials) and alternate resource, can be modeled.

Minimize the violation in minimum and maximum safety stocks

The minimum and maximum safety stock requirements are the input information provided for MPS that specify the corresponding lowest and highest quantity of a buffer item at a location. In the MPS plan, we aim to avoid violating those buffer items to not dip below or above the specified safety stock range at any planning period.

We can reformulate model (4.1) with minimum safety stock by adding the objective function as follows:

$$\min smin_1^3 + smin_2^3 + \cdots, + smin_{t3}^3. \tag{4.6}$$

Other than the given flow balance and resource load constraints, the additional safety stock constraints will be the following:

$$\begin{aligned}
&op_{1,1}^3 - b_{1,1}^3 - smin_1^3 = 0,\\
&op_{1,2}^3 + b_{1,1}^3 + smin_1^3 - b_{1,2}^3 - smin_2^3 = 0,\\
&\qquad\qquad \vdots\\
&op_{1,ti}^3 + b_{1,t3-1}^i + smin_{t3-1}^3 - b_{1,t3}^3 - smin_{t3}^3 - xd_{t3}^i = 0.
\end{aligned} \tag{4.7}$$

Here $smin_t^3$ is the decision variable at period $t = 1, \ldots, t3$ used to control the safety stock violations not to exceed the stock $b3$ more than $\widetilde{smin}^3$ at any planning horizon. The trivial bounds on safety stock violation variables will be: $0 \leq smin_t^3 \leq \widetilde{smin}^3$, $t = 1, 2, \ldots, t3$. Likewise, we can formulate the maximum safety stock violation where the defined decision variable in the model form constraints of not going below the specified safety stock level.

Minimize violation of raw material

All the MPS models discussed above consider no restriction in most upstream raw material availability. If the customer provides the raw material availability information and expects the planner not to go beyond this specified material capacity, minimizing the violation of raw material is the objective to push raw material consumption as near as available. We define the violations variables $rmv_t^1, t = 1, \dots, T$ and add the raw material flow balance constraints as:

$$rmv_t^1 - op_{1,t}^1 - b_{1,t}^1 = -\tilde{rmv}_t^1 \text{ for all} t = 1, \dots, T.$$

The objective function will be to minimize the weighted-sum of violation variables:

$$\sum_{t=1}^{T} w_i rmv_t^1.$$

Here w_i is set as per the priority.

Minimizing Operations associated with the Initial period

To minimize the build ahead, the planner penalizes earlier operations more than later operations. The objective function of this aim is to minimize the weighted objectives as follows:

$$\min \sum_{t=1}^{T} (T - 1 + t)\{w_1 op_{1,t}^1 + w_2 op_{1,t}^2 + w_3 op_{1,t}^3\}.$$

Here, three different operations, $op1$, $op2$, and $op3$, each loading to resource type $r = 1$, are weighted by positive parameters $w_i = 1,\ 2,\ 3$ as per the priority among the operations.

4.2 Combining Objectives

Suppose customers provide the information of priorities among objectives to the planner in the computation of MPS. In that case, the lexicographic method should be the most preferred solving technique by modeling it to an h-MOP. The reasons for selecting this method are that 1) it will always provide a Pareto optimal solution. 2) There is no requirement for normalizing the objective functions. Moreover, 3) it does not suffer from the problem of an ideal weight computation when obtaining one solution point. We recall Chapter 3 for the lexicographic technique. Other than the advantages of preferring the lexicographic method over the weighted-sum for MPS, the difficulty is that it requires the solution of many single objective functions to obtain just one solution point. We do not see this in the weighted-sum method [23]. We can study various weighted-sum methods in [14] that give different ways to combine objectives. However, there are difficulties with almost all of them [124, 125], - 1) they do not guarantee a Pareto optimal solution, 2) when only one solution point is needed, selecting weights is challenging, and 3) MPS with conflicting objectives with varied unit scales is unsuitable. This tradeoff motivates us to combine weight-sum techniques with the lexicographic method for h-MOP.

For h-MOP, we introduce a technique that 1) reduces the number of single objective runs we usually solve in the lexicographic method and 2) the resulting solution adheres to Pareto optimality even after combining some objectives. The idea works as follows: Before calling the lexicographic method on the MPS model, we check how many of the objectives can be combined using a simple weighted-sum approach and what should be the suitable weights. Consider we have two consecutive objectives, obj^1, and obj^2, in the model, defined as follows:

$$\text{obj}^1 := a_{11}\ x_1 +\ a_{12}\ x_2 + \cdots + a_{1n}\ x_n \text{ and } \text{obj}^2 := a_{21}\ x_1 + a_{22}x_2 + \ldots + a_{2n}x_n.$$

We are given that obj^1 is a high priority objective than obj^2. If we have to decide whether obj^1 and obj^2 can be combined and solved as a weighted-sum approach, they must satisfy the following conditions:

1. Priority-condition: The minimum possible absolute value among the pairs (a pair of decision variable and the associated coefficient), $a_{11}x1,\ a_{12}x_2, \ldots, a_{1n}x_n$ should be more than the maximum absolute value among the pairs $a_{21}x1,\ a_{22}x_2, \ldots, a_{2n}\ x_n$, i.e.,
$$\frac{\alpha\ \min\{\mid a_{11}x1 \mid,\ \mid a_{12}x_2 \mid, \cdots, \mid a_{1n}x_n \mid\}}{\max\{\mid a_{21}x1 \mid,\ \mid a_{22}x_2 \mid, \cdots, \mid a_{2n}x_n \mid\}} \geq \beta^m, \tag{4.8}$$
here β^m is a positive threshold to ensure that the positive multiplier α should always pertain to giving high priority to obj^1. We assume that the minimum chosen absolute value among the pairs in the objective function is nonzero.

2. Stability-condition: The multiplier should not exceed the maximum tolerance value. We can limit it by a positive threshold value β^M.
$$\alpha\ \max\{a_{11}x1,\ a_{12}x_2, \ldots, a_{1n}x_n\} \leq \beta^M \tag{4.9}$$

We name it *ObCrunch*. The steps to solve the model using ObCrunch that applies the above conditions are mentioned in Algorithm 6. We start with the vector of the objective function C indexed with their priority orders. We start with the first objective as the current processing objective function and check whether it satisfies priority-condition and stability-conditions with the next objective. If both conditions are satisfied, we combine them, and the combined objective functions check the conditions to combine the third objective. Otherwise, we do not combine them, leave the current objective as is and follow the same process by considering the second objective as the current processing objective function. We follow a similar procedure unless we complete all the objectives in C.

Algorithm 6: ObCrunch: Combining Objectives in Lexicographic Method

```
Input: An MPS model with objective functions vector $C := [\text{obj}^1, \ldots, \text{obj}^K]$.
Output: List of $K$ solutions for each objectives, $S := [y^1, \ldots, y^K]$.
Initialize: Set $n = 1$, cur_obj = $\text{obj}^n$, next_obj = $\text{obj}^{n+1}$ and $N = \emptyset$.
Step I: if $\exists t_n \in \mathbb{R}_+$ : cur_obj and next_obj satisfy equations (4.8) and (4.9)
then
    Update cur_obj $= t_n \, \text{obj}^n + \text{obj}^{n+1}$ ;
else
    Store cur_obj to $N$ and update cur_obj $= \text{obj}^{n+1}$;
Step II: Update n = n+1;
if $n == K$ then
    Solve the model with $N$ as objective functions and compute its solution $x^*$;
    For each $\text{obj}^k$ in $C$ compute $y^k = \sum_{i=1}^{n} a_{ki} \, x_i^*$ and store in $S$;
else
    Update next_obj $= \text{obj}^{n+1}$ and go to Step I;
Step III: Return $S$ as the final solution to the given input problem.
```

The advantage of this idea is that we always get a Pareto optimal solution and reduce the requirements of the solution of many single objective programs. Even deciding suitable values of the parameters used is not that problematic.

The major challenge is the availability of the bounds information of the coefficient and variable product components in the objective functions. For ObCrunch to work, we must know them before calling the solver. However, the MPS models can leverage the idea of the weighted sum with the lexicographic rule as the planner can obtain the bounds information from the supply chain input to the MPS. The inputs come from the demand details and the output of S&OP. We now discuss some of the objectives that the planner can combine them.

Combining fair-shared demand objectives (ANS-ANS): Recall from the above Section 4.1.1 that components in demand-based objective functions are the pair of weight coefficient and the demand-based decision variable. The total demand requests will be the bounds for these decision variables. A fair-shared demand is the demand request that meet in stages. From the fair-shared model (4.3), the k objective functions can be combined as the upper bound of any decision variable will be easily obtained from the requested demands. We consider the minimum nonzero absolute value of the coefficient variable pairs is unity. For $k = 2$, the combined objective function will be

$$\alpha(-w_1\, xd^{11}_{t1} - w_2\, xd^{21}_{t2} - w_3\, xd^{31}_{t3}) + (-w_1\, xd^{12}_{t1} - w_2\, xd^{22}_{t2} - w_3\, xd^{32}_{t3}).$$

The value of $\alpha > \max xd^{12}_{t1},\ xd^{22}_{t2},\ xd^{32}_{t3} \leq \max \widetilde{d^{12}_{t1}},\ \widetilde{d^{22}_{t2}},\ \widetilde{d^{23}_{t3}}$. Here α should also respect stability-condition. Combining demand objective followed by lateness objective functions (ANS-BL) is also possible. Recall the equation $xd^{1}_{t1} + f^{1}_{t1} + f^{1}_{t1+1} + \cdots, +f^{1}_{T} = 0$, in Section 4.1.2, which implies the available upper bound information of the decision variable of demand objective function must be an upper bound to the corresponding decision variables in lateness. Similarly, we can combine demand objectives of different priority levels (ANS1-ANS2) as the upper bounds of their respective decision variables are known to us.

Though the availability of upper bounds of the components in the objective functions can help combine two or more consecutive objectives, it fails to respect the stability condition. Due to varied demands, the multiplier coefficient, α, will sometimes go very large. Coefficients with huge values or significant variations in the objective function can cause trouble in various solving processes, especially pre-solving and optimizing steps. An optimization CPLEX highlights the numerical difficulty in its user manual [126]. Drawback under such weighted-sum is studied in [125].

In demand fair-shared requested demand objective, which we name ANS-ANS, we tightened the upper and lower bounds of the demand decision variables. Consider the two fair-shared objectives from the model k-DemandsMPS, named ANS^a and ANS^b, are combined as follows:

$$\text{combined\_ANS\_ANS} = -w_1(\alpha\, xd^{11}_{t1} + xd^{12}_{t1}) - w_2(\alpha\, xd^{21}_{t2} + xd^{22}_{t2}) - w_3(\alpha\, xd^{31}_{t3} + xd^{32}_{t2}) \tag{4.10}$$

Now to find the minimum value of α that holds both the priority and feasibility condition, we consider a two-level fair-shared demand request of 2 unit. Figure 4.1 illustrates it, where the maximum demand in the first level is denoted by a decision variable x and in the next level is by y. We have provided two types of requests 1) At level 1, the demand of 1 unit followed by 1 unit, and 2) At level 2, the demand request of 1.5 units followed by 0.5 units. If we solve this problem lexicographically, the model will first try to meet the demand requirement at level one. If it can not meet all the demand at level one, it will not meet the demand required at the next level. So from the combined objective combined_ANS_ANS, we will obtain the same solution with any positive value

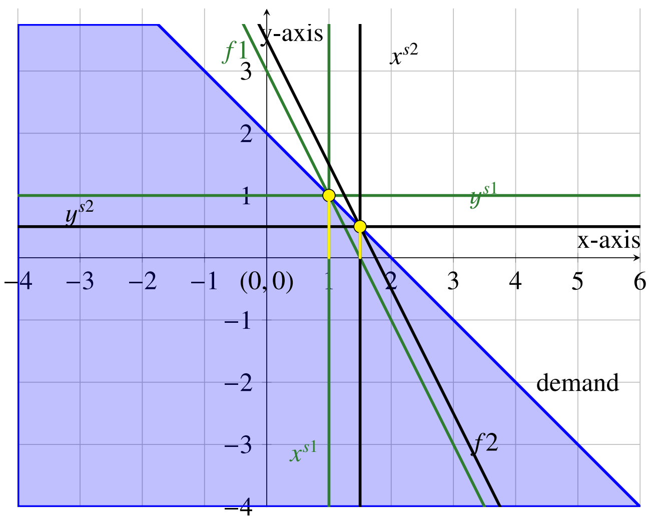


Figure 4.1: Fair-Shared Demand Allocation

of the multiplier, α. Now, due to the input material's availability, we can meet more than the demand requested at level 1. For the first type of request, we meet the demand requests of 1 unit to the customer, followed by the remaining amount. In the second type of request, we provide all the 1.5 units, and the remaining we meet in the second level. Two vertical lines show the Pareto optimal solutions after solving the first objectives in both types of fair-shared demands. The single acceptable point in the Pareto set is the accepted answer. We have also drew the lines $f1 : 2x + y = 3$ and $f2 : 2x + y = 3.5$ that represent the weighted-sum objective functions. Both pass from the respective selected Pareto optimal points. In fact, the weighted-sum objective function $\alpha x + y \geq x + y$ for any positive value of α. Likewise, the combined objective 4.10 with any positive α will provide the same optimal point that we get from the lexicographical objective in the model k-DemandsMPS. We discuss its computational effectiveness over standard lexicographic methods in the next section, Section 4.3.

4.3 Implementation of Combining Objectives: Benefits and Challenges

To provide the benefits of reducing overall time in the computation of the MPS, we implement it into a commercial supply chain master-planning software using the idea of combining objectives. The implementation is done and offered to the customer at different flags. At flag 1, we combine demand objectives under fair-share categories. The customer accepts a fair share of demand requests at, say, k stages. Usually, k is set to 3 to

5. In other words, customers want a planner first to meet their assumed $\alpha 1$ fraction of the total requests. Then they ask $\alpha 2$ fraction of the remaining demand requests. It continues to meet the demand requests. The remaining items are requested at the last stage, αk fraction. Note that $\alpha 1 + \alpha 2 + \ldots, \alpha k = 1$. So, flag1 can combine the k sequence of objectives. With flag 2, we combine demand objectives with backlog objectives. It can combine two sequences of objectives in the lexicographic solve. Enabling flag 3 switches flags 1 and 2 on. Whichever combination, a fair share of demand requirements or demand backlog pair, is possible, it combines sequentially. The last flag, flag 4, allows the general combination using the procedure mentioned in Algorithm 6 with the appropriate selection of parameters used in the procedures. MPS solver accepts these flags as input. Before triggering the lexicographic solve in the MPS solver and without changing the formulation, the overall number of objectives by combining some of them as per the flag values is often reduced, providing the benefit in run time.

Since customers always set demand and backlog objectives as the highest among all business key performance indices, the proposed plan of combining the objectives over flag numbers 1, 2, and 3 was quite logical. Moreover, the Lexicographic method does not violate plan quality by utilizing a weighted-sum with some sequence of objectives. We implemented the variable-fixing technique, a standard lexicographic technique (see Chapter 3 for detail) with the weighted-sum, which we call ObCrunch, in Python 3.7 and used python CPLE API for modeling and CPLEX-12.0.0.0 for the linear program solves. We compare our method of ObCrunch with the default lexicographic scheme available in CPLEX. For all the experiments, we use the Primal simplex method. All the other settings, including presolves, heuristics, etc., were left as is. The machine we used is 64-bit Intel (R) Core(TM) i7-6820HQ at 2.7GHz CPUs with 16 GB RAM.

We ran the idea with various data sets over five different CPG customers. Table 4.1 reports the resulting summary of the experiments. Column two lists 19 datasets, and column three reports the flag number chosen for combining the objectives. For each customer dataset, we run our experiment with different flags depending upon the presence of demand fair-share and demand objective followed by backlog objectives. We see a consistent benefit to customers. We find that the idea helps 18% on average improvement in run time than the default CPLEX method. Plan quality for all the datasets for both the solving methods is the same since the combination technique does not violate the quality of the solution.

For general combination, enabling flag 4 covers flags 1, 2, and 3 if the underlying objectives appear in the dataset. Besides demand-based and backlog-based objective functions of these flags, flag4 also combines demands and backlog objectives of different

Table 4.1: Solving time using Default Lexicographic in Cplex and ObCrunch for the data sets from CPG customers

Sr. no.	Dataset	Total no. of objectives	Objectives combined	Flag number	Existing Lexicographic run (CPLEX 12.10 default) time (in sec)	ObCrunch (Variable fixing + Weighed sum) time (in sec)	ObCrunch %age Improvement
1	CPG11	75	24	2	3055	1666	45.47
2	CPG12	75	24	3	3055	2290	25.04
3	CPG21	208	34	1	2390	1993	16.61
4	CPG22	208	34	2	2390	1960	17.99
5	CPG23	208	34	3	2390	1932	19.16
6	CPG31	26	6	2	1213	1036	14.59
7	CPG32	26	6	3	1213	1051	13.36
8	CPG33	26	6	2	1099	967	12.01
9	CPG34	26	6	3	1099	953	13.28
10	CPG35	26	6	2	777	628	19.18
11	CPG36	26	6	3	777	649	16.47
12	CPG41	24	6	2	2640	2182	17.35
13	CPG42	24	6	3	2640	2099	20.49
14	CPG43	24	6	2	2815	2320	17.58
15	CPG44	24	6	3	2815	2378	15.52
16	CPG45	24	6	2	2751	2219	19.34
17	CPG46	24	6	3	2751	2352	14.50
18	CPG51	60	46	2	4143	3608	12.91
19	CPG52	60	46	3	4143	3450	16.73

priorities and consecutive objectives of safety stock violations. The experiment with flag 4 is done with the same settings as above. For most of the datasets, the run time performance with the experiment is similar to the experiment reported in Table 4.1, as demand and backlog base objectives are already covered with the flag = 1,2,3. We do not cover safety stock further in the experiment because the objective of safety stock violation is far less important than the demand and backlog-based objective for the customers we targeted. Instead of combining the safety stock or low priority objectives, the customer is okay with the suboptimal plan, so requests for an early LP solver are stopped.

In summary, we explained, for example, the method to model the MPS and discussed the essential objectives for the customer. We found that consecutive objective functions in the sequence of objective functions in MPS can be combined. We discussed how to combine them. We devised a general rule to combine the objectives in the lexicographic method and named it ObCrunch. Further, we found that the idea of combining the weighted-sum rule with the lexicographic method benefited some consumer and goods industries. We did not explore much on ObCrunch can be a future direction of the work on this front.

4.4 MPS in Potato Chip Manufacturing Model

To study MPS, we introduce a potato chip manufacturing company, a dummy example. The company produces two products, 1) Italian spicy and 2) Indian masala potato chip. The MPS computation plans for each commodity in the supply chain with the following input information and output requirements:

- Output business objectives: We consider three business objectives in the model: 1) minimize the unmet demand as the highest priority objective, 2) minimize the backlog, and 3) minimize the operation earliness as the lowest priority objective.

- Planning horizon: We plan for potato chip production planning over a week (seven days) of the planning horizon.

- Resource units: Two resources are needed in the manufacturing process - 1) R1, a resource unit, is required to slice the raw potatoes. The maximum capacity of the machine is 12 hours per day. Its production rate is 500 - each resource unit, R1, can process 500 units of slicing operations per day. 2) R2, a resource unit, is required to fry the sliced raw potato and pack the two potato chip flavors. The maximum capacity of the resource is 12 hours per day. Each resource unit of type

R2 can process 50 units of frying and packing operations per day for each potato chip flavor.

- *Processing units* are the operations that load the resource and convert the input items to output items. 1) OP1 is the operation that utilizes R1 resources and loads raw potatoes to convert them to slices of potatoes. The rate of consumption of raw potatoes at any period by OP1 is unity. Similarly, the rate at which it produces sliced potatoes at any period is unity. OP1 performs just in time (JIT) production. That is, there is no delay in processing - On the same day, it loads the input and produces the output. 2) OP2 and OP3 are the operations that utilize R2 resources and load sliced raw potatoes to convert them to final products, Indian masala (F1) and Italian spicy (F2), respectively. The consumption rate of sliced potatoes at any period by OP2 and OP3 are unity. Similarly, the rate at which it produces F1 and F2 at any period is unity. Like OP1, OP2 and OP3 also have JIT productions.

- There is no delay in transportation operation to pick up the final product and deliver it to the customers.

- The production start date is March 1, and the end date is March 7.

- Demand requirements of item D1, the Indian masala, and D2, the Italian spicy chip, are 1800 packets each. The demand for Italian spicy chips has been forecasted higher than for Indian masala. So D2 is given more priority than D1. Both have due dates of March 3.

We illustrate its supply chain diagram in Figure 4.2. Input and output units are denoted with green color. The input item is labeled with the term “Infinite,” which means that the firm’s supply of raw potatoes is unrestricted. The buffer item I (Sliced potatoes) and buffer items F1 and F2 are colored in gray. The item I is labeled with “intermediate", which indicates the item is of intermediate type and will be input to other operations to be processed. Resources are colored in blue and are labeled by their resource capacities.

To meet the requirements of 1800 units with the due date, March 3, for each, we need to see the material and resource capacities available within this date. The flow of materials from start to end on the horizon and from raw potatoes to ultimate customer demands of potato chips are depicted in a network flow diagram in Figure 4.3. The top horizontal line represents the horizon. The network structure consists of nodes and edges. Nodes represent the material at a given period, and the vertical line connecting two nodes represents the process, an operation needed to consume one material item and produce another. The vertical edge indicates it is same-day (JIT) production. A horizontal edge between two material nodes of the same types indicates the amount of material carried over from one period to the next. Nodes labeled with $r1,\ r2,\dots,r7$ denote raw material available at $t = 1,\dots,7$. Similarly, $i1,\ i2,\dots,i7$ denote intermediate buffers, and $f1,\ f2,\dots,f7$ denote finished products.

Now we come to modeling front - Slicing operations are $\texttt{OP1T1},\cdots,\texttt{OP1T7}$ for day $t = 1,\dots,7$. Frying and Packing operations to produce the finished products of two flavors, $F1$ and $F2$, are denoted by $\texttt{OP2T1},\cdots,\texttt{OP2T7}$ and $\texttt{OP3T1},\cdots,\texttt{OP3T7}$, respectively, for day $t = 1,\dots,7$. $\texttt{CBAL1T1}$, $\texttt{CBAL1T2}$, $\dots,\texttt{CBAL1T7}$ denote inventory carryover over sliced products. Similarly, carrying over the inventory of final products, $F1$ and $F2$ are $\texttt{CBAL2T1}$, $\texttt{CBAL2T2}$, $\dots$, $\texttt{CBAL2T7}$ and $\texttt{CBAL3T1}$, $\texttt{CBAL3T2},\dots,\texttt{CBAL3T7}$, respectively. If the demand requirement can not meet on the due date, the late production will meet the requirement from day $t = 4,\cdots,7$. The operations to meet the demand on and after the due date are denoted by $\texttt{F1T3},\dots,\texttt{F1T7}$ for demand items D1. Similarly, $\texttt{F2T3},\dots,\texttt{F2T7}$ are for demand item D2. This network structure helps in forming the following flow balance and backlog-based demand constraints:

At buffer node $r1,\ r2,\dots,r7$, there will not be any constraints due to available supply. At $i1,\ i2,\dots,i7$, we have:

$$\begin{aligned}
BAL1T1 &: -OP2T1 + OP1T1 - OP3T1 - CBAL1T1 = 0\\
BAL1T2 &: -OP2T2 + OP1T2 - OP3T2 + CBAL1T1 - CBAL1T2 = 0\\
&\vdots\\
BAL1T7 &: -OP2T7 + OP1T7 - OP3T7 + CBAL1T6 - CBAL1T7 = 0
\end{aligned}$$

The material flow balance constraint, for example, at $i2$, which we name as $BAL1T2$, equates the total flow-in material produced by the upstream operation $OP1T2$ and carried over inventory $CBAL1T1$ from the previous period with total flow-out material consumed by downstream operations $OP2T2$ and $OP3T2$. The inventory remains left at $CBAL1T2$ for the next period. Similarly, we write the constraints for material items $f2$ for both chip types. Note that the balance constraints generated at nodes $f3,\ f4,\ f5,\ f6$, and $f7$ contain extra backlog variables used to meet customers' demands after the due date assigned to the planner.

$$\begin{aligned}
BAL3T1 &: OP3T1 - CBAL3T1 = 0\\
BAL3T2 &: OP3T2 + CBAL3T1 - CBAL3T2 = 0\\
BAL3T3 &: OP3T3 + CBAL3T2 - CBAL3T3 - F2T3 = 0\\
&\vdots\\
BAL3T7 &: OP3T7 + CBAL3T6 - CBAL3T7 - F2T7 = 0
\end{aligned}$$

and

$$BAL2T1 : OP2T1 - CBAL2T1 = 0$$
$$BAL2T2 : OP2T2 + CBAL2T1 - CBAL2T2 = 0$$
$$BAL2T3 : OP2T3 + CBAL2T2 - CBAL2T3 - F1T3 = 0$$
$$\vdots$$
$$BAL2T7 : OP2T7 + CBAL2T6 - CBAL2T7 - F1T7 = 0$$

The other constraint from the network structure is to balance the total demand needed by the customer. Here `AMT1` and `AMT2` are the demand fulfilled to the customers of types $D1$ and $D2$, respectively. The balance constraints for $D1$ and $D2$ are as follows:

$$D1BAL : -AMT1 + F1T3 + F1T4 + F1T5 + F1T6 + F1T7 = 0$$
$$D2BAL : -AMT2 + F2T3 + F2T4 + F2T5 + F2T6 + F2T7 = 0$$

Limitations in the capacities of resources form constraints other than flow balance constraints, called “resource load constraints”. Constraints on $R1$ and $R2$ at period $t = 1, \cdots, 7$ are as follows:

$$LOADR11 : 0.002OP1T1 - CAP1BDT1 = 0$$
$$LOADR12 : 0.002OP1T2 - CAP1BDT2 = 0$$
$$\vdots$$
$$LOADR17 : 0.002OP1T7 - CAP1BDT7 = 0$$

$$LOADR21 : 0.02OP2T1 + 0.02OP3T1 - CAP2BDT1 = 0$$
$$LOADR22 : 0.02OP2T2 + 0.02OP3T2 - CAP2BDT2 = 0$$
$$\vdots$$
$$LOADR27 : 0.02OP2T7 + 0.02OP3T7 - CAP2BDT7 = 0$$

We also take care of allowable capacities of resources and the maximum demand the planner can meet by creating trivial inequalities as follows:

$$0 \le CAP1BDTi \le 12, \ i = 1, \cdots, 7,$$
$$0 \le CAP2BDTi \le 12, \ i = 1, \cdots, 7,$$
$$0 \le AMT1 \le 1800,$$
$$0 \le AMT2 \le 1800.$$

Finally, we obtain objective functions for business objectives considered under MPS. Construction of objective functions is done before solving any of them.

- Minimize the unmet demands :

$$\text{minimize } -1\,AMT1 - 2\,AMT2.$$

Here the weight assigned to $AMT2$ is lesser than that of $AMT1$. It is because Italian spicy flavored potato chips that $AMT2$ points to are in higher demand than the Indian masala flavored chip that $AMT1$ points to.

- Minimize the demand backlog or reduce the lateness in meeting the demand on the due date.

$$\begin{aligned} \texttt{minimize}\ 1\,F1T3 + 2\,F1T4 + 3\,F1T5 + 4\,F1T6 \quad (4.11)\\ +5\,F1T7 + 1.1\,F2T3 + 2.1\,F2T4 + 3.1\,F2T5 \quad (4.12)\\ +4.1\,F2T6 + 5.1\,F2T7. \quad (4.13)\end{aligned}$$

The backlog variables are time-weighted - we assign less weight to the variables near the due date to impose a high penalty for more delayed production.

- Minimize the operations earliness or reduce the build of the product ahead of time:

$$\begin{aligned} \texttt{minimize}\ 7OP2T1 + 6OP2T2 + 5OP2T3 + 4OP2T4\\ +3OP2T5 + 2OP2T6 + OP2T7 + 7OP1T1\\ +6OP1T2 + 5OP1T3 + 4OP1T4 + 3OP1T5\\ +2OP1T6 + OP1T7 + 7OP3T1 + 6OP3T2\\ +5OP3T3 + 4OP3T4 + 3OP3T5 + 2OP3T6 + OP3T7\end{aligned}$$

Our target is to minimize the early operation as much as we can. The objective function sets a high penalty for early operations production as late as possible.

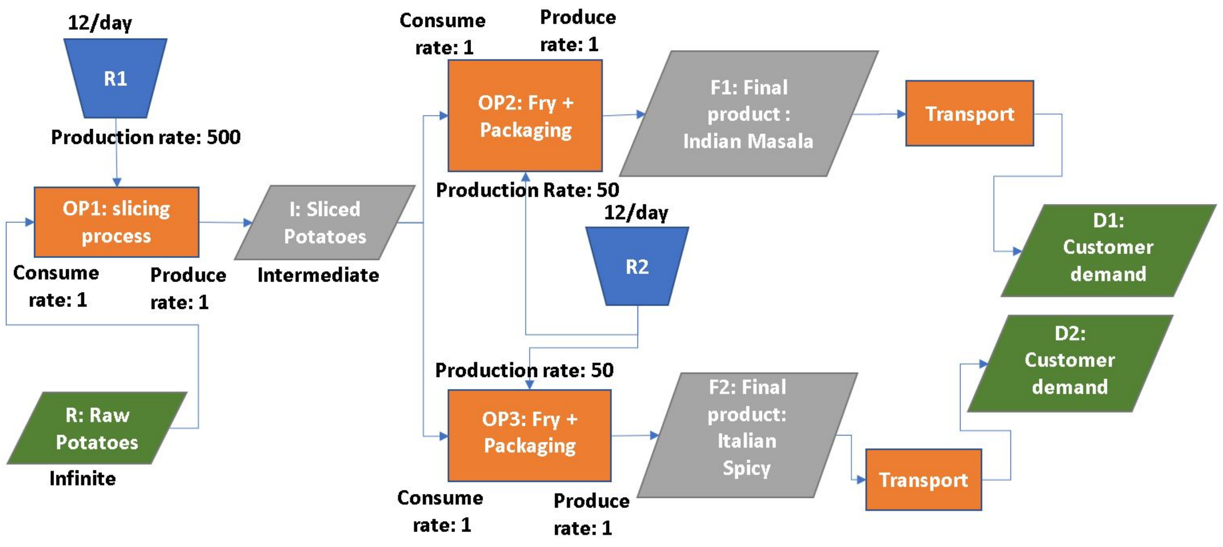


Figure 4.2: Supply Chain Diagram of a Potato Chip Manufacturing Industry

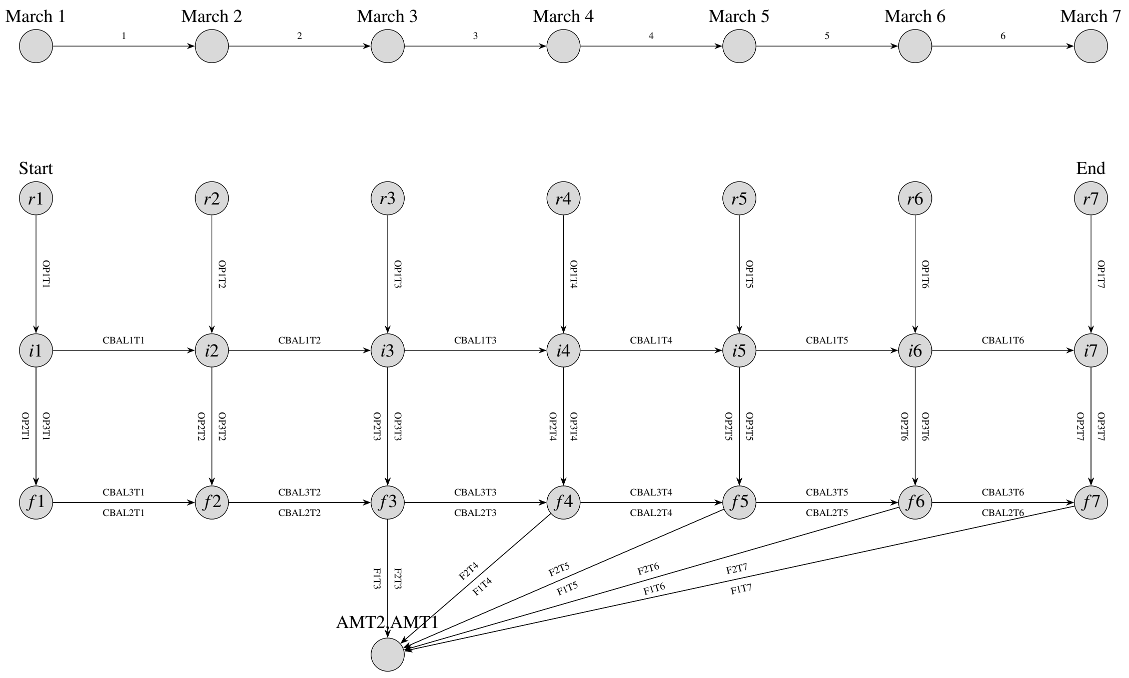


Figure 4.3: Network Flow of Material Over the Planning Horizon

Chapter 5

Master Production Schedule with Campaign Planning Restriction

This chapter studies manufacturing planning that considers campaign planning restrictions. Campaign planning (CP) plays an essential role in the batch production of varieties of products from the same assembly line in manufacturing industries. Its goal is to plan activities to reduce unnecessary production overheads, such as changeover time, inventory, etc., while simultaneously improving demand satisfaction. In chapter 1, we have described CP in detail and discussed the related literature. Our contribution to CP in this chapter is twofold.

1. We improve the existing heuristic that considers the campaign constraints as the changeover and limited resource restrictions. The idea is to model the CP problem as a sequential decision problem (SDP) and use the Cross-entropy (CE) method, an evolutionary algorithm used for policy learning to improve the quality of the existing heuristic, for CP.

2. We provide an exact formulation for master production scheduling (MPS) that respects CP constraints. It is an extension of the work done in the previous chapter on modeling MPS.

For the background of SDP and CE, we refer to Chapter 1.

Our work on MPS is close to [75] by NB Kamath, *et al.*, which includes CP with MPS heuristically by imposing campaign constraints locally. We refer to it as a 'heuristic method'. At each bucket, there is a restriction on the maximum number of running operations belonging to a group that produces similar products and a restriction on total changes of states of operations from the idle state to running from one bucket to another. We recall that we partition the production horizon into discrete time units, such as hour, day, week, and month (depending on the type of planning problem). We refer to each unit as a bucket. The steps heuristic method generally follow: Firstly, it does the production planning (MPS), considering all the business objectives hierarchically without looking into any violation of campaign planning restriction. This computed planning helps to evaluate the *weighted consumption profile (WCP)*, a measure used to set the priority values for each running assembly operation. Then violations in planning are avoided by inspecting each bucket by turning off/on the assembly operations as per its priorities. This decision is based on a linear weighted function, a picture of the on-hand inventory, demand, and safety stock signals in a pre-determined number of future buckets. This process continues over the entire campaign planning horizon. Figure 5.1 depicts the procedures used in the heuristic. It uses the knowledge of restriction on several running operations within and across the buckets (discrete-time intervals in the production horizon). We find some challenges with this method:

- It is a heuristic approach - it imposes campaign constraints locally, leading to a suboptimal plan,
- The obtained plan output is parameter sensitive, and
- One output plan requires multiple smaller-sized MPS to solve.

The proposed resolutions to address these issues are:

1. Improving the 'heuristic method' by formulating the campaign planning problem as a sequential decision problem and finding the ideal parameter values using the CE method. We call it 'improved heuristic'. Instead of using the ad-hoc rule of supplying the initial weights for the computation of WCP as in the heuristic method, it uses an intelligent weight vector, selecting the best possible policy.

2. Reformulate the basic mathematical model of MPS by incorporating campaign constraints. We call it 'exact method'. Though it changes the model from linear to integer, the benefits we get are 1) the model returns a globally optimal solution by a single MILP solver call, and 2) it computes other important KPIs without violating the campaign constraints and avoiding additional modeling effort.

We start with improved heuristic in Section 5.1. The idea of 'exact method' that applies CP constraints on MPS is discussed in detail in Section 5.2. We report some computational results, summarize our work and highlight future work in Section 5.3. Finally, we discuss the importance of campaign planning in one of the tire manufacturing industries in Section 5.4.

Table 5.1: Plan quality dependency on input weight vector

Weight vector for WCP	Weight vector used campaign objectives	Total Demand	Satisfied/max lateness
1000-1000-1000-1000-1000	1000-1000-1000-1000	12000	9000/1
1523-1483-2089-1758-893	14753-11097-632-703	12000	9000/4
9523-8483-2089-1758-893	14753-11097-632-703	12000	12000/4

5.1 Campaign Planning as SDP

The heuristic method imposes the campaign constraints bucket-wise and then resolves the MOLP. Thus for every bucket, there is at least one computationally expensive h-MOLP solver call. In addition, the obtained plan is highly sensitive - it depends upon the weights chosen as input. We experiment by running the heuristic method for the computation of MPS of a dummy supply chain, which we will describe in Section 5.2, over different input weight vectors. We observe that the plan quality of MPS varies with different input weights. We report it in Table 5.1. Two input weight vectors are used in the run - one for WCP calculation and the other for objective function calculation reported in columns one and two, respectively. For the requested total demand reported in column three, the obtained plan quality in terms of total demand satisfied and total delay from the target date (lateness) is mentioned in the last column.

We incorporate two constraints to the MPS model to respect the campaign constraints in the MPS. They are: 1) bounds on the number of ongoing operations in a given bucket and for a given group of operations and, 2) a limit on the changes of those active operations running from one bucket to another.

The heuristic method models the MPS and computes WCP, which helps decide the campaign selection. Given the campaign constraints, it decides which operations are to be disabled, enabled, and stopped. It is evaluated from measures such as on-hand inventory, demand, and safety stock signals in a pre-determined number of future buckets. The weights associated with the metric is user determined. The consumption profile of an operation is an effective supply required running in a given bucket. The effective supply is equal to the difference between the demand required minus starting inventory. A WCP is the weighted sum of the profile over the first few buckets. The selection of parameters associated with this metric is highly sensitive.

Selecting weights for consumption profiles is a common rule of thumb, and there is no standard rule for its construction that would be effective for most CP problems. Such procedures are myopic and lead to a local solution with poor plan quality and high computation time. The improved heuristic models are posed as a sequential decision

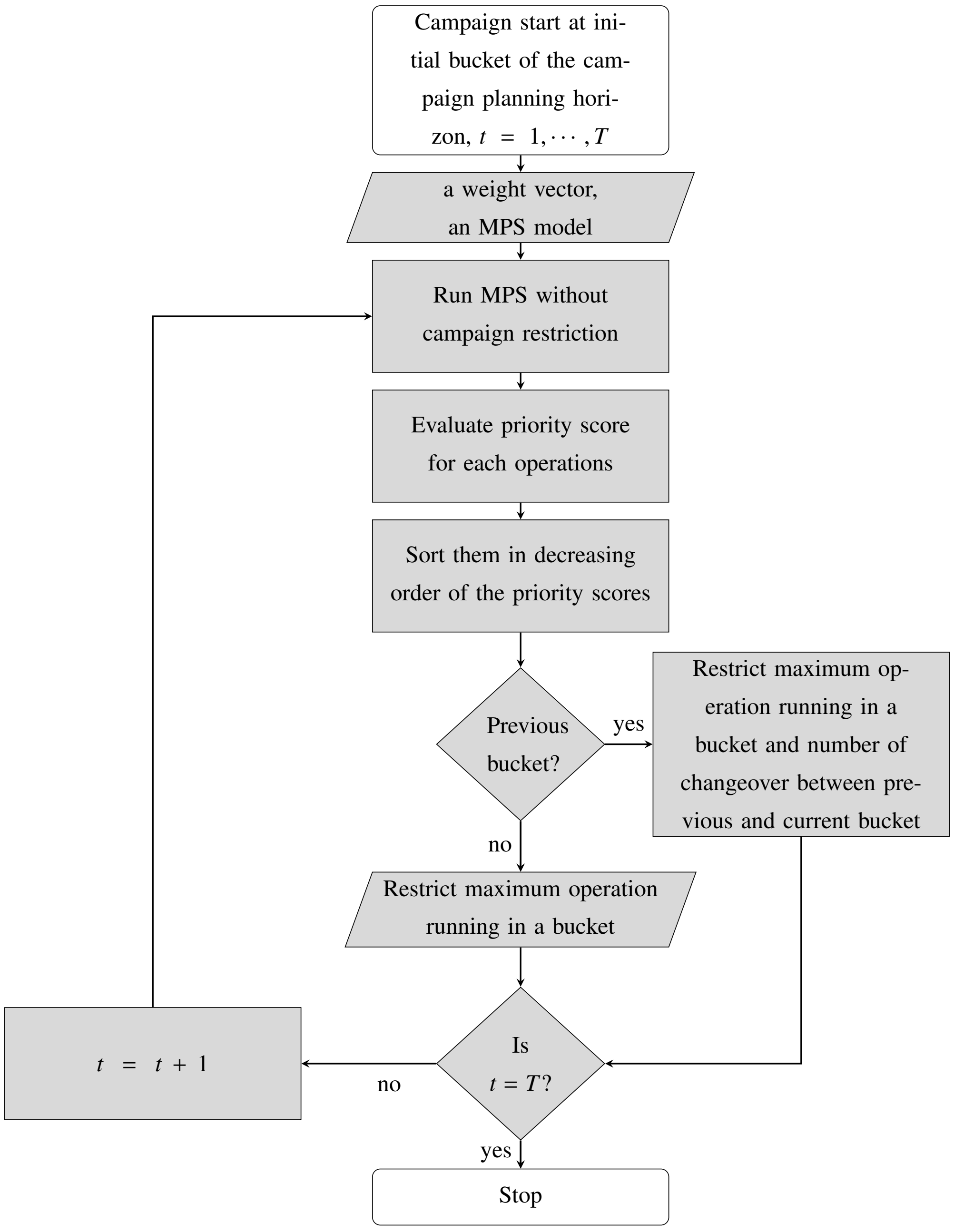

Campaign start at initial bucket of the campaign planning horizon, $t = 1, \cdots, T$
a weight vector, an MPS model
Run MPS without campaign restriction
Evaluate priority score for each operations
Sort them in decreasing order of the priority scores
Previous bucket?
yes
Restrict maximum operation running in a bucket and number of changeover between previous and current bucket
no
Restrict maximum operation running in a bucket
Is $t = T$?
no
$t = t + 1$
yes
Stop

Figure 5.1: Steps followed in the heuristic method

problem (SDP) where input weights are policies. Our objective is to compute the policy that provides the weight vector that leads to the most improved MPS from the modeled SDP.

5.1.1 Sequential Decision Problem

In sequential decision problems (SDPs), the utility of actions taken by a decision maker does not only depend on a recent decision but the whole sequence of the decision maker's actions. A policy is a sequence of actions required to determine the utility, objective, or average reward. When an agent in a given state takes action, it receives an immediate reward, and the system occupies a new state. The utility depends on the sequence of states and state-action pairs. We can consider campaign planning as a campaign objectives minimization problem (a utility) respecting campaign constraints that can be formulated as an SDP. The planner (the decision maker) has to decide on an effective campaign selection (an action) at each time bucket (decision epoch) and also has to target a set of key performance indices (KPIs) at the end of the planner. Here, these KPIs can be considered elements of a suitable utility function.

SDP formulation of the CP follows:

Decision epoch - Production horizon: $T = \{bkt_{starts},\ bkt_{starts} + 1,\ bkt_{starts} + 2, \ldots, bkt_{end}\}$;
State: S = Set of all possible configuration of the WCP for every campaign operations set OP;
Action: A = Set of all permutation of operations which are enabled at a state s. For example, a = 11001 is one of the actions at state $s \in S$ where there are five campaign operations and, first, second and fifth of them are enabled. After taking this action, we proceed to next state where again we will get different WCPs;
Reward: Reward$(s_{j,i})$ = $\max_{a_q \in A_{s_{j,i}}}\{Eval(s_{j,i},\ a_q)\}$;
Transition Probability: Deterministic;
Given a state s_t and an action set A_{s_t} as an input, for each time bucket t, from start till termination of the campaign horizon (duration in planning horizon that must respect campaign restrictions), we obtain the following action:

$$a_t^* = \arg\min_{a_i \in A_{s_t}}\{\text{Exp}(\text{Reward}(s_t,\ a_i))\}. \tag{5.1}$$

Here $(s_t,\ a_i)$ is a state obtained by taking an action a_i on state s_t.
a^*_t can be extended as,

$$a^*_t = \arg\min_{a_i \in A_{s_t}} \sum_{\left\{\begin{array}{l} S_{j,i} \in \text{set of possible states generated} \\ \text{after keeping an OP value as it is} \\ \text{and disabling others in } (S_t, a_i) \end{array}\right\}} Prob((s_t, a_i),\ s_{j,i}) \times \text{Reward}(s_{j,i}).$$

where $Prob((s_t, a_i), s_{j,i})$ is the transition probability from (s_t, a_i) to $s_{j,i}$ and Reward$(s_{j,i})$ is minimum evaluation-value of the possible next state, i.e.,

$$\text{Reward}(s_{j,i}) = \min_{a_q \in A_{s_{j,i}}} \{Eval(s_{j,i}, a_q)\}.$$

Transition probability is deterministic: For each state and action we specify a new state. The evaluation function has the following form:

$$Eval = function(w_1 \times f_1 + w_2 \times f_2 + w_3 \times f_3 \cdots + w_d \times f_d,\ w_{d+1},\ w_{d+2}, \ldots, w_{d+k}), \quad (5.2)$$

where $Eval$, gives evaluation-values of the given configuration of consumption profiles (state) in the SCP plan. It is a linear combinations of the features $(f_1, f_2, f_3, \cdots, f_d,)$ weighted by coefficients $(w_1,\ w_2,\ w_3, \ldots, w_d)$. Note that features are the objective values of the campaign metrics of every layer, such as, demand not satisfied, lateness, earliness, inventory evaluated from the run. The k weights $w_{d+1}, \ldots, w_{d+k}$ are associated with first k lookahead consumption profiles.

Our motivation for modeling the CP as SDP and learning the underlying policy using Cross-entropy method comes from a popular game 2048 [127], an addictive single-player, non-deterministic puzzle game modeled into an MDP framework [128]. There are 16 tiles on the 2048 board. The action is to move the tile up, down, left, or right to combine various tiles ∈ starting with a tile of 2 and combining them to reach 2048. The tile of 2 combines with the tile of 2 and makes a tile of 4. At the same time system pops up a tile of 2 or 4 with equal probability. Similarly, tile of 4, 8, 16, ... 1024 combines with the tile of the same number. The game terminates if no further moves are possible or one of the tiles gets the number 2048.

In our case of the improved heuristic, we start with the consumption profiles associated with each campaign operation as the initial state. We fix some operations to zero to restrict the number of operations running in a bucket and the restriction in the changeover operation. It is one of the actions in the model. After fixing some variables, we call the solver to solve the updated h-MOLP model. The evaluation function evaluates the score from the solution obtained from the LP solve. The process continues till we reach the final stage. The better the input weights assigned to the model for the computation of WCP and *Eval*, the better the MPS plan quality we will obtain. To learn an ideal weight, we use the CE method.

5.1.2 Steps in Cross-entropy Method

CE method follows the following steps for an optimal policy for our model at any iteration t. We perform an initialization process by choosing the initial k weights for WCP and the weights associated with d objective functions - mean μ_{ti} and standard deviation σ_{ti}, for individuals w_i, iteration $i = 1, \cdots, k + d$. We then generate N random sample vectors for every elements in a vector using normal sample distribution with parameter vectors

$$(\mu_{t1}, \ldots, \mu_{tk}, \ldots, \mu_{t(k+d)} \text{ and } sigma_{t1}, \cdots, \sigma_{tk}, \cdots, \sigma_{t(k+d)}).$$

For each generated sample as an input weight vector, we use the policy (discussed above, based on evaluation function) that returns the corresponding utility value, say $Eval(w_j)$, $\forall j = 1, 2, 3, \cdots, N$. We sort these sample vectors by the generated output values (in descending order). Assign the top output value as $OutTop_t$. If the stopping cricteria meets we stop the process with the learned weight W_t with the utility value $OutTop_t$. Otherwise, we top m samples from the sorted N population and evaluate mean and standard deviation vectors from them and repeat the same process. A flow chart shown in Figure 5.1.2 illustrates the per iteration set of steps followed in the CE method applied to the campaign planner heuristic. Samples that are the weights generated from distribution on continuous space can be arbitrary. The distributions used to get generated random weights at each iteration are assumed to be Normal with unknown parameters as the state space in the campaign problem is continuous.

The initial parameters for the CE method are the following:

Mean weights w, it an initial guess of the unknown mean parameters to the Normal distributions.

$$w0 = w0_1, w0_2, \ldots, w0_c, w0_{c+1}, w0_{c+2}, \ldots, w0_{c+d}$$

contains c weights for WCP and d weights for initializing evaluation function. Standard deviation $s0$, a vector consists of standard deviations associated to the elements of $w0$ - the $i'th$ pairs $(w0_i, s0_i)$ such that $w0_i \in w_0$ and $s0_i \in s0$ corresponds to the empirical mean and standard deviation to the $i'th$ normal random distribution. After every iteration, a new vector of mean and standard deviation pairs gets updated. After a long run, from the principle of the strong law of large numbers, the empirical mean vector (weight vectors) converges to a true mean vector. Selection of $w0$ and $s0$ is crucial. A better selection

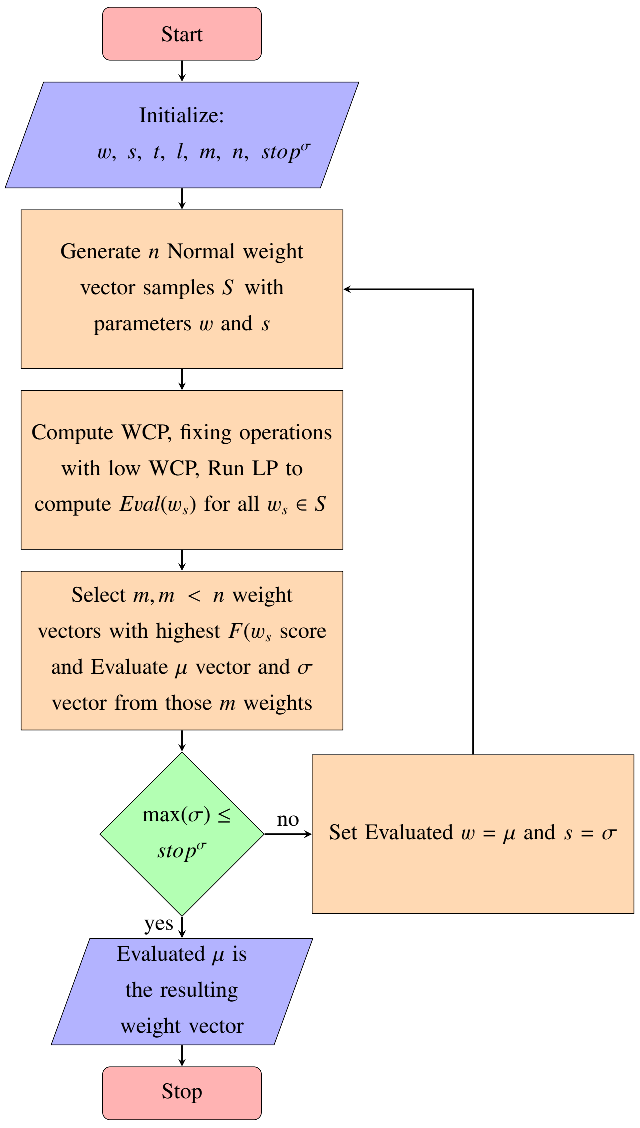

Start
Initialize:
$w,\ s,\ t,\ l,\ m,\ n,\ stop^{\sigma}$
Generate n Normal weight vector samples S with parameters w and s
Compute WCP, fixing operations with low WCP, Run LP to compute $Eval(w_s)$ for all $w_s \in S$
Select $m, m < n$ weight vectors with highest $F(w_s$ score and Evaluate μ vector and σ vector from those m weights
$\max(\sigma) \leq stop^{\sigma}$
no
Set Evaluated $w = \mu$ and $s = \sigma$
yes
Evaluated μ is the resulting weight vector
Stop

Figure 5.2: Cross-entropy method for learning the input weight vector used in the improved heuristic

of initial means and standard deviation vectors leads to faster convergence of CE to the best possible solution. We use expert opinion-based $w0$ and $s0$ selection which implicitly gives a prior belief that is dependent on the previous plan quality of similar dataset for a given customer. Expert suggests that weights associated with consumer profile should be in decreasing order over the horizon, i.e., the weight associated with the current bucket should be assigned a higher value than the weight associated with the next bucket. Similarly, weights associated with objective function campaign planning should be assigned the weights proportional to the priority given to the respective KPIs. For example, the weight associated with the demand not satisfied should be given more weight than the lateness. However, the optimal weights obtained after learning from CE may differ from the expert suggestion.

Sample size s and the number of iteration $itrn$: Each iteration generates s random weight vector samples using a Normal distribution with mean vector $w0$ and standard deviation vector $s0$. The mean vector and standard deviation vector consist of different mean weights associated with consumption profile and campaign planning objective function. Given every set of mean-standard deviation pairs as normal parameters, s number of random weights get generated.

$Stop^{\sigma}$, the stopping criteria can be 1) the maximum number of iterations, if convergence is computationally expensive 2) the standard deviation touches the lower bound. In our case, 0.00001 is set as the lower bound of the standard deviation. Thus, if the maximum of the standard deviation vector elements reaches less than 0.00001, we terminate the Cross-entropy process. In our implementation, it is set as one-eighth of the mean value. Sub-sample is the best m sample to be chosen to evaluate the empirical mean and standard deviation. Here "best" samples represent those samples whose evaluation function values are better.

The improved heuristic is implemented with a small dummy data set. We call it a "small-industry supply chain scenario". It has two plants that produce four items I1, I2, I3, and I4. There are two groups for each plant. So, there are four groups, G11 and G12 belong to plant1, and G21 and G22 belong to plant2. I1 and I4 belong to group G11 and G21 of plant1 and plant2, respectively, and I2 and I3 belong to group G12 and G22 of plant1 and plant2, respectively. We run our experiment with various demand requests from the customer ends. First, we mathematically model the MPS for this problem. We refer to Chapter 3 for the steps to model the MPS. We can find its modeling detail, production rate, consumption rate, maximum capacity of the resource, and network flow constraints from the formulated model. We describe it in Appendix C.

Table 5.2: Performance comparison of improved planner over the heuristic method on small-industry dataset

Demand				Total	Heuristic method		Improved heuristic	
item1	item2	item3	item4	Requests	Satisfied	Lateness	Satisfied	Lateness
3000	3000	3000	3000	12000	9000	1	12000	4
4000	3000	4000	3000	14000	11000	3	14000	5
1500	2000	3300	2500	9300	7300	0	9300	9
4500	3000	3300	1500	12300	9300	2	12300	5

The hardware used for the computation is a 64 bit Intel(R) Xeon (R) E5-2673 v4 at 2.30GHz CPUs with 20 cores and 64 GB RAM. We use CPLEX-12.10.0, for solving the intermediate LPs hierarchically. Initial mean and standard deviation vectors are

$$w0 : [100000, 50000, 50000, 20000, 5000, 100000, 50000, 10000, 5000],$$

and

$$s0 : [100000/8, 50000/8, 50000/8, 20000/8, 5000/8, 100000/8, 50000/8, 10000/8, 5000/8].$$

Sample size which is the sample population of weights we generate every iteration is $n = 45$ and the subsample, which is the total number of weights vectros selected as per the Eval score, is $m = 15$. We set stopping criteria $stop^{\sigma} = 0.00001$

A comparison table also compares performance indices, demand satisfaction, and lateness. Figure 5.1.2 clearly illustrates that the result evaluated from the learned weight outperforms the existing weight assigned randomly to the solver.

To validate the robustness of the computed weights, we run our improved planner with different demand scales of a Small-industry supply chain scenario. In this, 2000 units are the base demand items requirement. We use `X` to represent items requested, `X` $*$ `k` represents the requested items if we scale the demand required by a k fraction to `X`. For each instance the total demand after scaling is denoted by `TD`. `S` denotes the total demand meet to customer and, `TDLT` denotes the total lateness in days from the due date. The table shown in Table 5.3 provides the results obtained by running the improved heuristic on various input demand scales. It provides a better plan than the existing plan over each demand scale. In the next section, we will perform the same experiment with the 'exact method'. We will find that the current solution obtained by the improved heuristic using an ideal weight is globally optimal.

The above analysis and result, and the convergence plot obtained from running improved method provide an effective methodology for the better plan quality. However,

Table 5.3: A result summary of the heuristic method and the improved campaign planner with various input demand scale

Demand :=X = 12000		heuristic method		improved heuristic	
Demand*k	TD	SAT	TDLT	SAT	TDLT
X*0.2	2400	1800	0	2400	9
X*0.4	4800	3600	0	4800	9
X*0.6	7200	5400	0	7200	9
X*0.8	9600	7200	0	9600	9
X*1.0	12000	9000	1	12000	4
X*1.2	14400	10800	3	14400	6
X*1.4	16800	12600	4	16800	6
X*1.6	19200	14400	5	19200	6
X*1.8	21600	16200	6	21600	7
X*2.0	24000	18000	6	24000	7
X*2.2	26400	19800	6	26400	8
X*2.4	28800	21600	6	28800	8
X*2.6	31200	23400	6	31200	8
X*2.8	33600	25200	6	33600	15
X*3.0	36000	27000	6	36000	15
X*3.2	38400	28800	6	38400	15
X*3.4	40800	30600	13	40800	22
X*3.6	43200	43200	217	43200	22
X*3.8	45600	45600	217	45600	22
X*4.0	48000	48000	217	48000	22

one can analyze that though the above experiment looks promising, all the experiments are performed with small data set. In practice, for the large data set, it would be challenging to perform weight learning. For example, consider a scenario where a campaign planner takes an hour to complete one run. If the standard CE method is applied and considered, it requires, on average, 5000 runs to converge the evaluation function and return the weights. The whole run would take at least 5000 hrs, i.e., 208 days! It is practically not possible to request a customer to wait for these many days. To speed up the process, we can parallelize the CE method. A batch of weight vectors is generated at each iteration in the standard CE method, called a sample set. For every generated sample, a campaign planner is triggered. All samples are independent as they do not share any information among them.

Though 'improved heuristic' showed computational effectiveness, learning an ideal input parameter requires many LP solver calls. Even after the parallel run, it is not feasible for large dataset to learn the input weights. For large-sized models training process is expensive. To resort to it, we focus on an 'exact method' which does not depend upon the input weights and requires a single solver call.

5.2 Formulation of Campaign Planning as a Mixed Integer Program

We recall Chapter 4 for mathematical modeling of MPS with demand satisfaction as one of the business requirements as follows: in a given supply chain problem as follows:

$$\text{LP1:}\quad \min -xd^1_{t1} - xd^2_{t2} - xd^3_{t3} \tag{5.3}$$

$$\text{s.t.}\quad \sum_{i \in O^1} op^i_{1t} - c^1_t \leq 0, \quad \text{for all } t = 1, \ldots, T, \tag{5.4}$$

$$op^i_{11} - b^i_{11} = 0 \quad \text{for all } i = 1,\ 2,\ 3, \tag{5.5}$$

$$op^i_{12} + b^i_{11} - b^i_{12} = 0 \quad \text{for all } i = 1,\ 2,\ 3, \tag{5.6}$$

$$\vdots \tag{5.7}$$

$$op^i_{1ti} + b^i_{1ti-1} - b^i_{1ti} - xd^i_{ti} = 0 \quad \text{for all } i = 1,\ 2,\ 3, \tag{5.8}$$

$$\text{bound: } 0 \leq xd^i_{ti} \leq \widetilde{d^i_{ti}} \quad \text{for all demand item } i = 1,\ 2,\ 3, \tag{5.9}$$

$$0 \leq c^1_t \leq max\_c^1_t \quad \text{for all bucket } t = 1,\ 2,\ 3, \cdots, T, \tag{5.10}$$

$$op^i_{1t}, b^i_{1t} \geq 0 \quad \text{for all } i = 1,\ 2,\ 3, \text{and } t = 1,\ 2,\ 3 \cdots, T. \tag{5.11}$$

Here requirement of demands is of equal priority. $\widetilde{d^j_{t_x}}$ are the demand requirements, where j and t_x denote the corresponding item code and due date to receive the demand requirement. The decision variables $c^r_1,\ c^r_2, \ldots c^r_{\tilde{t}}$ are the amount of resource (associated to each resource $r \in R$) required to process the associated operations at time bucket $t = 1,\ 2,\ 3, \cdots, \tilde{t}$. Each variable c^r_t is upper bounded by the known amount of resource, maximum capacity($max\_c^r_t$). Similarly, the decision variable op^i_{jt} defines the operation $i \in O^j$ with the resource j utilized at time bucket t that is needed to produce one unit of the product item. Associated to each inventory location $i \in I$ and resource type j, a decision variable b^i_{jt} defines the amount of inventory carried from time bucket t to $t + 1$. We also define an associated decision variable xd^j_t that denote the demand (of type j) that could be satisfied over the given due date t over the known supply chain settings. Here R and I denote the index sets of resources and inventory items available in the production process. O be the index set of operations with subsets $O^k \subseteq O$ that can utilize the resource $r^k \in R$. A known amount 'load_per' is the amount of resource utilized by one unit of operation.

The bucket to bucket planning of the supply chain creates a network structure that helps in posing a network-type mathematical formulation. For a simplistic formulation, assume there is only one resource r that can load three operations $O1,\ O2$ and $O3$. Here $O^r = \{1,\ 2,\ 3\}$ is the index set of production operations and, $r = 1$ is the resource type. Each operation type consumes raw material (available in infinite amounts) and produces the corresponding finished goods $d1$, $d2$, and $d3$. We set the planning horizon as a daily bucket window, $t = 1,\ 2, \ldots, T$ days. We can make it more simplified by considering load_per to 1 and the lead time to be zero. We set the rate at which operations consume items as input to produce per unit item to be unity. The demand requirements of the finished products are: $\widetilde{d^1_{t1}}$ units of item $d1$ on $t = t1$ day, $\widetilde{d^2_{t2}}$ units of item $d2$ on $t = t2$ day and $\widetilde{d^3_{t3}}$ units of item $d3$ on $t = t3$ day.

Other than the information for MPS formulation mentioned in Section 1.4.1, two additional information we must consider in the campaign planning formulation with MPS

1. bounds on the number of ongoing operations in a given bucket and for a given group of operations and,

2. a limit on the changes of those active operations that are running from one bucket to another.

With this set of information, the planner must optimize each business requirement without any violation of the hierarchy among them and, minimize the setup costs that are needed to support multiple operations for given resources.

LP1 consists of inventory balance constraints that balance total inflow, outflow, and inventory carryover of materials at a location and a particular time bucket and resource load constraints that consider the capacity utilization of resources. We also have to consider other KPIs required in MPA. Optimizing them simultaneously over the given supply chain constraints is not possible. We solve them in a hierarchy by associating each business requirement with a priority value. If we do not consider campaign planning during MPS, the plan may be infeasible or may lead to a suboptimal result. Running setup minimization problem after the MPS relies on the solution already obtained from MPS with no campaign constraints. If we run the setup minimization problem before the MPS, high-priority business requirements may go for a toss. We handle this by incorporating CP constraints with MPS formulation as follows:

1) We add constraints that add a restriction on the number of operations running in a time bucket for a given group of operations sharing a common resource. A positive parameter mo_t^g that denotes the maximum running operations in a time bucket t, for a given group of operations g, is known to planner. It constrains the campaign changes in a bucket for the shared resources.

Let us consider $O1,\ O2,\ O3, \dots Ok$ be k types of operations that belong to g, supported by a shared resource r. For such case, $O^r = \{1,\ 2,\ 3 \dots, k\}$. The decision variable op_{rt}^i defines the operation $i \in O^k$ loading the resource r at time bucket t. Since any operation that belongs to g can load r at any time bucket $t = 1,\ 2,\ 3, \dots, T$, we define a binary variable xop_{rt}^i associated to each op_{rt}^i. $xop_{rt}^i = 0(1)$ indicates that the corresponding operation belongs to g, that loads r is disabled (enabled) at t. Further, given the upper bound mo_t^g, for any g we can have

$$xop_{rt}^1 + xop_{rt}^2 + \cdots xop_{rt}^k \leq mo_t^g \quad t = 1,\ 2,\ 3, \dots, T. \tag{5.12}$$

Equation (5.12) is the knapsack constraint that respects campaign constraint. In addition, each variable op_{rt}^i in LP1 will be replaced by $xop_{rt}^i \cdot op_{rt}^i$.

2) We add constraints that add a restriction on the number of changes in the set of active operations. The active operation is the running operation in the user-defined group of operations for a given time bucket. A user-defined group of operations is the collection of those operations that belong to similar products. A positive parameter co_t^g denotes the maximum possible changes in the running operations (from the idle state to running state or vice versa) in $t \geq 2$ and for a group g. Note that this constraint further assists in maintaining a gradual shift of campaigns for a resource over time.

Let xop_{rt}^i and $xop_{r(t+1)}^i$ are binary variables associated to operation i and belongs to g at t and $t + 1$ respectively. A binary decision variable $down_{rt}^i$, is equal to $xop_{rt}^i \cdot (1 -$

$xop^i_{r(t+1)}) = 1$ if the active operation Oi at t is disabled at $t+1$. Similarly, an indicator decision variable up^i_{rt} is equal to $(1 - xop^i_{rt}) \cdot xop^i_{r(t+1)} = 1$ if the disabled operation Oi at t is active at $t+1$. For a given g and r, total number of operations that are active at t and disabled at $t+1$ is denoted by $down^g_t$. Similarly, total number of operations that are disabled at t and active at $t+1$ is denoted by up^g_t. They are expressed as follows:

$$down^g_t = \sum_{i \in O^k} down^i_{rt} \text{ and } up^g_t = \sum_{i \in O^k} up^i_{rt}.$$

For a given co^g_{t+1}, at bucket t where $1 \leq t \leq T-1$, we have

$$\min(up^g_t, down^g_t) \leq co^g_{t+1}. \tag{5.13}$$

We can reformulate constraint (5.13) as follows:

$$y^g_t \leq down^g_t;\ y^g_t \leq up^g_t;\ y^g_t \geq down^g_t + up^g_t - 1;\ y^g_t \geq 0.$$

Applying them to LP1 forms mixed-binary nonlinear program (MBNLP). Nonlinearity terms in the MBNLP, such as the product of binary variables $down^i_{rt} = xop^i_{rt} \cdot (1 - xop^i_{r(t+1)})$ and $up^i_{rt} = (1 - xop^i_{rt}) \cdot xop^i_{r(t+1)}$ can be linearize as follows:

$$\begin{aligned} down^i_{rt} &\leq xop^i_{rt} & \quad up^i_{rt} &\leq (1 - xop^i_{rt}) \\ down^i_{rt} &\leq (1 - xop^i_{r(t+1)}) \quad \text{and} & up^i_{rt} &\leq xop^i_{r(t+1)} \\ down^i_{rt} &\geq xop^i_{rt} - xop^i_{r(t+1)} & up^i_{rt} &\geq xop^i_{r(t+1)} - xop^i_{rt} \\ down^i_{rt} &\geq 0, & up^i_{rt} &\geq 0. \end{aligned}$$

Similarly, we can linearize $z^i_{rt} = xop^i_{rt} \cdot op^i_{rt}$. This linearization reformulates a mixed binary integer program as follows:

$$\text{MBIP1:} \quad \min -xd^1_{t1} - xd^2_{t2} - xd^3_{t3} \tag{5.14}$$

$$\text{s.t.} \quad \sum_{i \in O^1} z^i_{1t} - c^1_t \leq 0, \quad \text{for all } t = 1, 2, 3, \ldots, T, \tag{5.15}$$

$$z^i_{11} - b^i_{11} = 0 \quad \text{for all } i = 1, 2, 3, \tag{5.16}$$

$$z^i_{12} + b^i_{11} - b^i_{12} = 0 \quad \text{for all } i = 1, 2, 3, \tag{5.17}$$

$$\vdots \tag{5.18}$$

$$z^i_{1ti} + b^i_{1ti-1} - b^i_{1ti} - xd^i_{ti} = 0 \quad \text{for all } i = 1, 2, 3, \tag{5.19}$$

$$xop^1_{1t} + xop^2_{1t} + xop^3_{1t} - mo^1_t \leq 0 \quad \forall t = 1, \ldots, T. \tag{5.20}$$

$$y^1_t - down^1_t \leq 0 \quad \forall t = 1, \ldots, T-1, \tag{5.21}$$

$$y^1_t - up^1_t \leq 0 \quad \forall t = 1, \ldots, T-1, \tag{5.22}$$

$$y_t^1 - down_t^1 - up_t^1 + 1 \geq 0 \quad \forall t = 1, \ldots, T-1, \tag{5.23}$$

$$down_t^1 - \sum_{i \in O^k} down_{1t}^i = 0 \quad \forall t = 1, \ldots, T-1, \tag{5.24}$$

$$down_{1t}^i \leq xop_{1t}^i \quad \forall t = 1, \ldots, T, \; i = 1, \; 2, \; 3. \tag{5.25}$$

$$down_{1t}^i \leq (1 - xop_{1(t+1)}^i) \quad \forall t = 1, \ldots, T, i = 1, \; 2, \; 3. \tag{5.26}$$

$$down_{1t}^i \geq xop_{1t}^i - xop_{1(t+1)}^i \quad \forall t = 1, \ldots, T, \; i = 1, \; 2, \; 3 \tag{5.27}$$

$$up_t^1 - \sum_{i \in O^k} up_{1t}^i = 0 \quad \forall t = 1, \ldots, T-1, \tag{5.28}$$

$$up_{1t}^i \leq (1 - xop_{1t}^i) \quad \forall t = 1, \ldots, T, \; i = 1, \; 2, \; 3, \tag{5.29}$$

$$up_{1t}^i \leq xop_{1(t+1)}^i \quad \forall t = 1, \ldots, T, \; i = 1, 2, 3, \tag{5.30}$$

$$up_{1t}^i \geq xop_{1(t+1)}^i - xop_{1t}^i \quad \forall t = 1, \ldots, T, \; i = 1, \; 2, \; 3, \tag{5.31}$$

$$z_{1t}^i \leq \overline{op_{1t}^i} \cdot xop_{1t}^i \quad \forall t = 1, \ldots, T, \; i = 1, \; 2, \; 3, \tag{5.32}$$

$$z_{1t}^i \leq op_{1t}^i \quad \forall t = 1, \cdots, T, \; i = 1, \; 2, \; 3, \tag{5.33}$$

$$z_{1t}^i \geq op_{1t}^i - (1 - xop_{1t}^i) \cdot \overline{op_{1t}^i} \quad \forall t = 1, \; 2, \; 3, \cdots, T, \; i = 1, \; 2, \; 3. \tag{5.34}$$

$$\text{bound: } 0 \leq xd_{ti}^i \leq \widetilde{d_{ti}^i} \forall \text{ demand item } i = 1, \; 2, \; 3, \tag{5.35}$$

$$y_t^1 \geq 0 \quad \forall t = 1, \cdots, T-1,, \tag{5.36}$$

$$0 \leq c_t^1 \leq max\_c_t^1 \quad \text{for all bucket } t = 1, \ldots, T, \tag{5.37}$$

$$op_{1t}^i, b_{1t}^i, down_{1t}^i, up_{1t}^i \geq 0 \; \forall i = 1, \; 2, \; 3, \; \& \; t = 1, \; 2, \; 3 \cdots, T. \tag{5.38}$$

$$xop_{1t}^i \in \{0, 1\} \; \forall t = 1, \; 2, \; 3, \cdots, T, \; i = 1, \; 2, \; 3. \tag{5.39}$$

All the variables, unless otherwise mentioned in MBIP1, are non-negative. It consists of binary variables and knapsack constraints. For each campaign operation, there is an associated binary variable. The total number of knapsack constraints is equal to the product of total buckets in the planning horizon and the number of user-defined groups of operations.

5.3 Implementation, Computational Results and Conclusion

In this section, we give empirical evidence of the effectiveness of our method. We model the MPS for two supply chain scenarios, 'S1' and 'S2', that require CP and report the performance observation by comparing our model with the heuristic method. The hardware used for the computation is a 64−bit Intel(R) Core(TM) E5- 2640 V3 at 2.60GHz CPUs with 16 cores and the main memory 128GB RAM. We use cplex12.9 [27] solver for solving our model.

Table 5.4: Demand planning summary of supply chain scenarios, 'S1' by heuristic-based planner and our MIP-based planner.

scaled-demand -type (total-demand* scaled factor)	demand-requirement	demand-satisfied			lateness		
		heuristic-method	exact-method	percentage benefit	heuristic-method	exact-method	benefit (in days)
12000*0.2	2400.0	1800.0	2400.0	33.3	0	9	-9
12000*0.4	4800.0	3600.0	4800.0	33.3	0	9	-9
12000*0.6	7200.0	5400.0	7200.0	33.3	0	9	-9
12000*0.8	9600.0	7200.0	9600.0	33.3	0	9	-9
12000*1.0	12000.0	9000.0	12000.0	33.3	1	4	-3
12000*1.2	14400.0	10800.0	14400.0	33.3	3	6	-3
12000*1.4	16800.0	12600.0	16800.0	33.3	4	6	-2
12000*1.6	19200.0	14400.0	19200.0	33.3	5	6	-1
12000*1.8	21600.0	16200.0	21600.0	33.3	6	7	-1
12000*2.0	24000.0	18000.0	24000.0	33.3	6	7	-1
12000*2.2	26400.0	19800.0	26400.0	33.3	6	8	-2
12000*2.4	28800.0	21600.0	28800.0	33.3	6	8	-2
12000*2.6	31200.0	23400.0	31200.0	33.3	6	8	-2
12000*2.8	33600.0	25200.0	33600.0	33.3	6	15	-9
12000*3.0	36000.0	27000.0	36000.0	33.3	6	15	-9
12000*3.2	38400.0	28800.0	38400.0	33.3	6	15	-9
12000*3.4	40800.0	30600.0	40800.0	33.3	13	22	-9
12000*3.6	43200.0	43200.0	43200.0	0.0	217	22	195
12000*3.8	45600.0	45600.0	45600.0	0.0	217	22	195
12000*4.0	48000.0	48000.0	48000.0	0.0	217	22	195

The scenario 'S1', mimics a small supply chain of a firm with two plants that produce four items I1, I2, I3, and I4. There are two groups for each plant: G11 and G12 belong to plant1, and G21 and G22 belong to plant2. Items I1 and I4 of group G11 and group G21 share the same resource. Similarly, I2 and I3 belong to groups G12 and G22 and share the same resource. The demand for each of the items is 3000 at the distribution center. The supply chain horizon used in 'S1' is 26 buckets and, each bucket is of daily bucket type. There are 25 business objectives that MPS needs to consider. The campaign constraints restrict the production changeover in the same group to one and the number of permissible operations per group per bucket to one. The scenario, 'S2', unlike 'S1', considers the larger supply chain and details one of the tire industries manufacturing supply chain problems. The mathematical formulation of MPS for 'S2' contains millions of continuous and discrete variables and constraints. Horizon consists of 86 buckets, daily, weekly, and monthly types. There are 87 business objectives that needs to optimize. There are hundreds of thousands of units of demand requirement, and there are more than twenty-two thousand groups present in this supply chain network. For both the scenarios, we model the respective MPS, obtain plan output and collect computation time in getting the planned output from our model with the existing heuristic. For 'S1', we report the scaled demand requirement, demand fulfillment, and lateness for both the MPS methods

Table 5.5: Demand planning summary of supply chain scenarios, ‘S2’ by heuristic-based planner and our MIP-based planner.

demand-type	demand-requirement	demand-satisfied		benefit	percentage benefit
		heuristic method	exact method		
0	14801059.0	14763993.0	14780603.0	16610.0	0.1125034
1	3708725.0	3658033.0	3691402.0	33369.0	0.9122116
2	308322.0	305494.0	305583.0	89.0	0.0291331
3	580865.0	551365.0	568238.3	16873.3	3.0602686
4	2978081.0	2859776.0	2944522.8	84746.8	2.9634052
5	596367.0	536562.0	589611.7	53049.7	9.8869632
6	11892967.0	11556315.0	11794757.0	238442.0	2.0633048
7	1607441.0	1462872.4	1552322.0	89449.6	6.1146568
8	380675.0	336808.0	365187.4	28379.4	8.4259894
9	1902510.0	1686265.9	1741089.4	54823.5	3.2511777
10	156067.0	155793.0	155966.0	173.0	0.1110448
11	672298.0	669415.0	663557.0	-5858.0	-0.875092
12	7006306.0	6810424.5	6722108.5	-88316.0	-1.296777

in Table 5.4. Lateness, the total number of days late from the target date, is set with low priority than the demand satisfaction. The column, percentage benefit, indicates the percentage change in demand met, reports a positive value if our MIP-based method (exact method) performs better. Also, a positive value in the column benefit (in days) represents the number of days saved in meeting the demand requirements if our procedure is preferred. Similarly, for ‘S2’, we report the priority-based customer demand requirements, demand fulfillment, and the benefits in demand met in Table 5.5. An ideal planner should focus more on high-priority demand requirements than the demand requirement with low priority. In the last column, percentage benefit reports the percentage change in demand met. The positive percentage value represents exact formulation-based MPS performing better and, the negative value shows its degradation over the heuristic. The exact method is MIP and solving it is generally much slower than that of LPs. However, it requires only one MIP solver call in contrast to the heuristic planner which requires many LP solvers. We observe that for both supply chain scenarios, our method performs better than the heuristic. The time taken in MPS computation from the exact method is 1.76 sec for supply chain scenario ‘S1’, which is nearly 70 times faster than the heuristic. Similarly, for larger supply chain ‘S2’, our model takes 111 seconds which is more than 10 times the heuristic that takes 1791 sec.

5.4 Campaign Planning in one of the Tire Manufacturing Industries

One of the more complex requirements of campaign planning is encountered in tire manufacturing. Master planning for tire manufacturing is particularly challenging due to the global manufacturing and distribution network and complex manufacturing processes. Tire manufacturing starts with the first step, a 'Green Tire' production, by assembling multiple layers of rubber and chemicals as required by the final finished tire performance characteristics. A 'Green Tire' then goes inside a 'Mould', which goes inside a 'Container' and is placed in a 'Cavity' of a 'Press'. The 'Press' is then taken through the 'Curing' process of around 20 minutes before 'Tire Articles' are obtained. 'Tire Articles' are taken through further quality checks, finishing, special treatments, and packaging to get the final 'SKU' that can be allocated, shipped, and sold against customer orders and forecasts across various sales channels.

There is a many-to-many relationship between the 'Green Tire', 'Press' and 'Article', requiring efficient planning of these processes to avoid costly changeovers. Since the outcome of the curing process is tightly linked with operational planning at the factory level, just the curing process cannot be modeled as a standalone optimization problem. It has to take place with other constraints and objectives at local and global plants, warehouses, and distribution lanes. This problem is solved by embedding 'campaign planning heuristics method' within the hierarchical optimization problem. However, even with ingenious mix of heuristics and hierarchical LP, the problem becomes very hard to solve and could run for as much as 100 hours on powerful boxes. This has necessitated further breaking down of the overall operation planning into different stages where demands and forecasts are propagated up to the stage of curing process, curing process is optimized using campaign planning and the resulting operation plans are taken as starting solution for rest of the global optimization problem. The 'segmented campaign planning approach' compromises the global optimality of the plan. Further, even the campaign planning problem could take as much as thirty hours to solve. These runs are part of an operational plan that runs every week, but these runtimes could still obstruct the business process flow.

5.4.1 Industrial Outlook of the Challenges of Campaign Planning

Campaign planning is the industry terminology for optimizing and managing the resources and work centers with the switchover requirements. The term 'campaign' basically comes from the simplistic approach to minimizing the changeover cost – campaign (keep running) the setup once it is loaded so that changeover costs are minimized. It is easy to see that the heuristic approach is myopic and leads to sub-optimal planning of resources. However, the challenge in adopting an optimal method is the complexity and solve-time of the model. With the presence of constraints for a global supply chain, a careless formulation of changeover minimization can easily keep solving for hundreds of hours on reasonably powerful resources. These long-running planning cycles are prohibitive due to various reasons. One of the reasons is the cost of computation. Even with the advent of SaaS (Software as a service) and PaaS (Platform as a service), the computing cost of such resources becomes very high. Global full-scaled supply chains needed to be replanned on a monthly or weekly basis are now require planning multiple times in a day. The disruption and exception in the extended supply chain are more accurately and frequently made available due to the advent of technology. Shipping agencies are providing updates on possible delays of trucks or ships every 15 minutes. The delay calculation can be based on real-time events (e.g., port congestions at the current docking port) or predicted based on environmental factors (e.g., predicted oceanic conditions). These more accurate inputs make re-planning more reliable. Further, there is pressure from the customer side, who also expects to be intimated of any possible delay or disruption more frequently and accurately. Most of the operational planning runs happen daily and, the planned output is rolled out for production orders and purchase orders in an incremental manner. If an optimal changeover problem exceeds the planning cycle time, the plan is just stale and not execution-ready. With the above practical constraints, most of the planners settle for heuristic-based changeover optimization. However, it has pitfalls – suboptimality, infeasibility, or both. If a formulation for campaign planning is well-formulated and is faster, it has immense business value. To conclude, we provide mathematical modeling for a master production schedule that respects campaign planning constraints and try to attempt few supply chain scenarios for understanding plan output. We study campaign planning problems one tire industry faces and highlight the industrial outlook of the challenges an industry or management can have. However, we only attempted to highlight the benefit of computing an efficient and optimal campaign plan. Implementation and challenges in other supply chain industries that look for the model respecting campaign planning and lot sizing constraints is the work that needs to be done.

Chapter 6

Conclusion and Future Work

We study MILPs and MOLPS, two categories of linear mathematical programs, one specified with integer variables and the other with many objectives. We focus on methodologies to obtain their solution that employs a general method of solving a sequence of LPs. For MILPs, we introduce a new branching technique similar to reliability branching that uses the closeness between LPs solved using the branch and bound method. For MOLPs, we develop a lexicographic method that, similar to the branching technique for MOLPs, exploits the similarity between LPs solved with preference. Apart from studying various strategies and developing methodologies for MILPs and hierarchical MOLPs (h-MOLPs), our contribution to industry problems is to perform a detailed study of the master production schedule (MPS), one of the main components in master planning in manufacturing industries, and some related restrictions associated with it. We study the modeling of MPS as h-MOLP. The challenge of addressing campaign planning in MPS is also studied using the heuristic and the exact formulation.

In Chapter 2, we discuss various variable branching rules and present a new branching procedure that looks more closely at the information collected at different nodes in B&B and tries to use them selectively. The procedure evaluates the similarity between the current node and nodes already explored in the tree to select an appropriate variable to branch on. Towards this end, we define a similarity measure between nodes computed using relevant features of the relaxation, like bounds on variables. Using information from ‘similar’ nodes, we estimate the change in the objective value for each branching candidate, much like reliability branching, to select the variable to branch on. We develop efficient procedures for implementing this scheme and, present computational results on benchmark instances and compared with the default scheme of a solver (CBC). We find that effectively calling strong branching speeds up the LP-based B&B for MILP by 20% and results in a 30% node reduction.

Chapter 3 studies two popular methods used to solve h-MOLPs and their challenges. We obtain results on the lexicographic method for h-MOLPs and propose a new reduced cost-based lexicographic technique. It exploits the structure of the underlying hierarchical model by monitoring the changes in the input parameters and leverages reoptimization when solving the objectives in the hierarchy. We define a similarity measure between intermediate linear programs appearing while solving the model and use it to decide whether we should solve the current linear program from scratch or use the available feasible solution obtained from the previous linear program solve. We show the effectiveness of our rule over the existing method on small-sized benchmark instances. We realize the consistent speedup of 25% over the available default lexicographic method in CPLEX for solving h-MOLPs, modeled for master production schedules.

In Chapter 4, we perform a modeling exercise where the MPS of a dummy potato chip industry is formulated. We discuss various important demand-based business objectives and a mathematical formulation for them. We devise a rule to combine some of the objectives in the lexicographic method. We find that functional knowledge of objectives' preferences helps us combine them without losing the solution quality of the standard lexicographic method.

Chapter 5 starts by posing the campaign planning heuristic as a sequential decision problem (SDP) and use the Cross-entropy method to solve it to obtain a better solution than the existing one. We avoid the unnecessary multiple MPS routines while ensuring optimal plan output. It develops an idea of an 'exact method' that applies campaign planning constraints on MPS. Further, we discuss a case study of campaign planning for one of the tire industries to understand the importance of campaign planning. We run our exact formulation of campaign planning in 12.10 version of the CPLEX solver and compare it with the existing campaign planning technique over h-MOLP modeled for master production schedules over two supply chain scenarios. In the first scenario, which is a small-sized supply chain, we see the performance improvement with seventy times faster planning computation from our method. For the second scenario, the speed up is ten times.

We now highlight some promising future research directions that can extend the work presented in this thesis.

Currently, our similarity-based branching procedure, SimBranch, is implemented only with MIPs where some variables are constrained to be binary. We can extend this idea by exploiting other features in the subproblems in the branch-and-bound procedure for general integer cases and other classes of problems like MINLP, quadratic programs, and CSP. Similarly, we can extend the work of the similarity-based lexicographic technique, SimLex, for the MOPs consisting of integer variables. Unlike reduced cost information for non-basic continuous variables used in our current work's similarity computation, we need to have some other feature that computed similarity and decide whether reoptimization is useful. In addition, we can extend the current work to link our Optimization-based SimBranch and SimLex with machine learning techniques that can exploit the feature vectors.

Another direction can be to use with no extra cost features collected in SimBranch for node selection strategy. We studied two key points we need to consider while selecting the node in the branch-and-bound procedure. 1) it should choose the node pointing to subproblems with the best lower bound (in the case of minimization), and 2) the node setup cost should not be high - the linear program should not change much from one iteration to the next.

Sensitivity analysis may be another research direction toward selective reoptimization in MOPs. Two consecutive LPs solved using the lexicographic method differ in more than two parameters. Sensitivity analysis in multi-objective decision-making is a popular area of study [129, 130]. Sensitivity analysis with simultaneous variations in the model is studied in [131]. To our best knowledge, almost all of them analyze the extent of variations in input parameters by which the solution of the model does not change - they perform the post optimality. Our idea is to check whether two LPs are similar to decide whether to build the solution from scratch or to use the existing solution, which can be combined with the simultaneous sensitivity analysis.

Focusing on our industry work, including lot sizing constraints with the master production schedule and further including it with campaign planning, can be another research direction. It will incorporate the situation where some intermediate buffer items produced or the final items demanded must be lot sized. Considering lot-sizing into MPS need modeling a mathematical program with integer variables. Further, including lot-sizing and MPS with campaign planning make the problem more challenging to model and solve. Another interesting area of research from our industry-based work is to find the sequence of business objectives management can accept in calculating MPS. A weighted-sum approach will combine those hierarchical objectives without losing their hierarchy in the lexicographic technique. Currently, we have such objectives - demand-based objectives. A group of safety stock-based objectives of various items can be an example of such groups that can be tried.

Appendix A

MIP Instances

Table A.1: Running time (t, in seconds), number of nodes processed (n) and number of strong branching iterations (#strong_itrn) by SimBranch and Default-Cbc on all 222 benchmark instances

	SimBranch			Default-Cbc		
Instance	t	n	#strong_itrn	t	n	#strong_itrn
30n20b8	2066.7	39554	4574136	1636.48	81262	963920
50v-10	7200	833564	2168082	7200	1454959	17890026
aflow40b	7200	1132095	1452	7200	556977	45187370
air04	28.78	144	120866	33.78	1282	29776
app1-2	7200	12862	3704174	4140.93	13876	1633646
assign1-5-8	7200	9437195	19272	7200	2509526	246931754
atlanta-ip	7200	2786	907208	7200	3023	1189275
b1c1s1	7200	150376	189426	7200	106076	3389556
bab2	7200	1903	1569398	7200	5526	138918
bab5	7200	26684	9446052	7200	79230	848403
bab6	7200	4596	3446948	7200	4605	131650
beasleyC3	7200	285732	226188	7200	215416	10438111
biella1	628.05	1350	1181269	7200	83754	1308635
bienst2	349.51	72550	10915	442.24	69748	1063511
binkar10_1	62.55	10976	27915	61.09	11514	68773
blp-ar98	7200	72223	15669195	7200	201116	1932681
blp-ic98	4006.47	47156	9096366	7200	128681	6411946
bnatt400	7200	46084	2194501	7200	36771	5169764
bppc4-08	7200	2303032	2577	7200	2248360	27508589
brazil3	7200	5732	3227386	7200	7304	3218996
buildingenergy	7200	352	384419	7200	233	203580
chromaticindex512-7	7200	10248	2309223	7200	68510	749293
cmflsp50-24-8-8	7200	69282	1510384	7200	36388	4151640
CMS750_4	7200	67192	257459	7200	63300	936250
co-100	7200	4298	1669251	7200	2085	84494
cod105	7200	111457	2177796	7200	79823	11417996
comp21-2idx	7200	2798	2290435	7200	3732	1331912
core2536-691	1478.76	4690	2694941	283.6	2292	65599
cost266-UUE	7200	587294	190994	7200	307727	21411168
cov1075	7200	918344	267801	7200	704709	15802997
csched007	7200	1038155	1467124	7200	1635149	17236674
csched008	7200	2394359	674224	7200	2177516	20811656
csched010	7007.23	764318	1330629	7200	1228224	12031128
cvs16r128-89	7200	46899	13700550	7200	68870	10586187
dano3_5	146.39	402	42528	155.7	324	77550
danoint	4997.59	502308	65470	7200	687526	17675634
dws008-01	7200	63354	9186413	7200	117693	1108765
eil33-2	221.23	5128	1515388	105.21	10894	169073
eilA101-2	7200	4952	5182640	7200	15906	329271
eilB101	1250.87	12968	4900827	1814.53	52456	721683
enlight_hard	194.33	120346	46737	233.59	67721	247210
enlight13	7200	494755	97213	7200	510872	427682
exp-1-500-5-5	7200	1381906	9850	7200	672138	53203314
fast0507	7200	19651	13637157	7200	196691	1747175
fastxgemm-n2r6s0t2	7200	288693	44241	7200	216553	5204311
germanrr	7200	77308	8922144	7200	206723	2290545
glass-sc	7200	107265	282356	7200	72952	2872744

glass4	7200	1862286	1930	7200	1176335	42397239
gmu-35-40	124.38	112950	12307	641.98	569884	1693734
gmu-35-50	7200	5565066	37495	7200	5033123	55728038
graph20-20-1rand	7200	18869	2051346	7200	14911	2985262
graphdraw-domain	2397.12	447282	94070	6323.99	856926	26853297
h80x6320d	7200	204832	25314	7200	123712	8143952
ic97_potential	7200	1281952	61101	7200	1370030	8331548
icir97_tension	7200	873119	465637	7200	510390	23914293
iis-100-0-cov	7200	191081	185718	7200	132204	6441808
iis-bupa-cov	7200	104982	157606	7200	85487	3217323
iis-pima-cov	7200	57171	296290	7200	46314	3037337
irp	4.57	40	12981	5.78	120	5514
istanbul-no-cutoff	1624.28	1848	10375	1825.67	1932	221638
lectsched-5-obj	7200	11621	3493839	7200	15220	2824217
leo1	7200	297782	8525081	7200	460363	4734826
leo2	7200	192245	10959569	7200	338839	3320975
lotsize	7200	111746	1906934	7200	305498	4761766
macrophage	255.05	3602	102809	571.3	8514	396148
mad	7200	9065978	84495	7200	5673186	181414257
map10	4627.14	2922	109354	4346.67	2422	356504
map16715-04	7200	1483	91038	7200	2572	448385
map18	1503.9	1688	84455	2173.68	2134	341471
map20	1491.2	2040	65534	1339.08	1602	234490
markshare_4_0	23.57	2195853	491	29.84	1626048	435928
markshare2	7200	171838226	4569	7200	190661114	119487467
mas74	534.2	3645065	21847	417.64	2996964	6226261
mas76	28.36	546805	7122	22.94	324782	521713
mc11	7200	170328	357467	7200	188146	5320395
mcsched	1268.39	58202	2141106	5675.38	262202	22819827
mik-250-1-100-1	273.73	837770	66886	1512.51	2242458	27569929
mik-250-20-75-4	38.36	151240	22436	28.94	124111	288004
milo-v12-6-r2-40-1	7200	213633	223383	7200	210100	2876704
mine-166-5	31.11	1772	42302	60.38	3608	123588
mine-90-10	3339.33	1122608	225083	3493.5	902058	14170010
momentum1	7200	6416	857667	7200	11036	1418417
msc98-ip	7200	706	571906	7200	1280	826488
mspp16	7200	1560	67337	7200	1893	98575
mushroom-best	7200	62121	313210	7200	36913	1858583
mzzv11	97.39	120	51102	82.72	140	33761
n2seq36q	7200	166298	11321696	7200	350979	2814187
n3div36	7200	77745	12779475	7189.3	263950	2997734
n3seq24	7200	35075	10611169	7200	87913	941624
neos-1109824	7200	78468	5175	4941.3	37686	3192172
neos-1337307	7200	83888	7223091	7200	73611	5093874
neos-1396125	753.61	73670	269950	781.8	46594	3025056
neos-1445765	594.4	6246	2738991	3336.82	173706	1722231
neos-1456979	7200	78935	4679817	7200	99059	5234655
neos-1582420	55.36	1054	481292	22.9	1298	108991
neos-2657525-crna	7200	1924971	364927	7200	10179555	70264336
neos-2746589-doon	7200	19231	6787719	7200	119921	1057921
neos-2978193-inde	7200	430102	25236	7200	275004	12294185
neos-3046615-murg	7200	16831515	85997	7200	3667520	2353094
neos-3083819-nubu	6.99	2098	26680	14.42	5966	51680
neos-3216931-puriri	4149.82	6178	2998006	4584.9	7442	2378586
neos-3381206-awhea	7200	555853	265849	7200	723373	4420560
neos-3555904-turama	7200	6836	4348083	7200	9741	4021870
neos-3627168-kasai	7200	1064567	71678	7200	1287477	5820479
neos-3656078-kumeu	7200	1424	910079	7200	1438	1050790
neos-3754480-nidda	7200	7808244	7147	7200	6795519	32224139
neos-4300652-rahue	7200	310	351651	7200	791	913734
neos-4338804-snowy	7200	1739219	261891	7200	1390117	20835744
neos-4387871-tavua	7200	124011	3410945	7200	84801	11175606
neos-4532248-waihi	7200	1004	619774	7200	1018	504577
neos-4647030-tutaki	7200	5359	10939	7200	6073	12791
neos-4722843-widden	7200	651	104449	7200	759	231751
neos-4738912-atrato	315.57	10385	236726	552.21	17189	976335
neos-476283	358.92	246	20274	688.42	1102	51034
neos-4763324-toguru	7200	1073	804486	7200	1445	458939
neos-4954672-berkel	7200	610748	48969	7200	488480	6101484
neos-5052403-cygnet	7200	2570	2389539	7200	3309	94382
neos-5093327-huahum	7200	54395	38683	7200	50483	2240747
neos-5107597-kakapo	7200	128610	685690	7200	73060	5552460
neos-5188808-nattai	7200	17124	48027	7200	21120	1371409
neos-5195221-niemur	7200	436	385154	7200	1017	851375

Continued on next page ⟶

neos-631710	7200	10058	1509899	7200	11071	478849
neos-662469	7200	41330	18003053	7200	209116	1519426
neos-686190	144.85	4298	698841	68.66	4848	219370
neos-848589	7200	655	142805	7200	440	332407
neos-860300	83.19	574	294399	138.42	5248	86993
neos-873061	7200	68691	17157	7200	72272	3450676
neos-911970	7200	1407586	4570	7200	814172	64810519
neos-916792	7200	315484	27282	1944.75	55134	2845557
neos-934278	7200	8300	3844844	7200	11962	5589591
neos-957323	1554.57	2258	1289761	752.58	12902	222443
neos13	3909.65	64266	255810	1618.55	16888	997324
neos17	7200	1045643	200976	7200	3135269	30448850
neos18	308.45	11262	201444	533.44	17302	483377
neos5	1667.73	3502835	18805	7200	1596440	8518782
net12	4375.4	1708	809779	5286.97	2568	730469
netdiversion	1176.09	226	173999	481.43	98	5645
newdano	7200	722081	43293	7200	527561	8034121
nexp-150-20-8-5	7200	139969	207585	7200	93258	3969075
noswot	7200	23261519	29058	7200	20838393	64073523
ns1208400	7200	114707	4896787	7200	251653	3120866
ns1688347	7200	8788	1500223	7200	4633	1363961
ns1830653	933.83	23692	496878	4921.93	112430	5031707
nu25-pr12	360.07	41357	457945	427.52	43614	904309
nursesched-medium-hint03	7200	640	344571	7200	586	31176
opm2-z10-s4	7200	424	390121	7200	654	277731
opm2-z7-s2	380.42	462	253802	669.62	2158	369847
p200x1188c	140.58	8784	218905	884.06	38270	2361162
pg	5.49	202	7461	6.38	160	8339
pg5_34	870.51	72248	1408760	838.95	32978	3153583
pigeon-10	7200	2609640	868240	7200	3109938	18460069
piperout-08	84.93	120	71514	88.24	291	11749
pk1	47.69	303191	8181	54.86	285094	1124744
proteindesign121hz512p9	7200	18447	4767155	7200	59886	1016773
proteindesign122trx11p8	7200	17213	3883868	7200	51870	847627
pw-myciel4	7200	231683	895950	7200	192724	17007743
qiu	98.07	12250	26678	123.88	10646	320188
radiationm18-12-05	7200	61071	2673087	7200	44104	6810637
radiationm40-10-02	7200	3867	2556249	7200	2603	1830401
rail01	7200	827	818794	7200	435	23000
rail507	7200	17036	12100888	7200	171056	1885423
ran14x18-disj-8	7200	1662572	193178	7200	920050	47689977
ran16x16	536.89	123902	98778	489.34	82644	3023779
rd-rplusc-21	7200	35180	983	7200	11268	1054913
reblock115	7200	1854165	484242	7200	1272312	31183214
reblock67	370.34	173234	158676	1137.56	323626	7220591
rmatr100-p10	119.61	2440	151782	133.04	2032	235715
rmatr100-p5	150.54	1544	145616	183.04	1520	258001
rmatr200-p5	7200	7664	253314	7200	5731	1178196
rmine6	1613.58	810650	215438	863.52	206772	4209024
rocI-4-11	7200	135518	275291	7200	164428	5186953
rocII-4-11	5367.99	45122	2723384	7200	32753	1788990
rocII-5-11	7200	20416	5348991	7200	35592	2233138
rococoB10-011000	7200	33024	8081301	7200	51767	470964
rococoC10-001000	565.14	8064	2431158	708.66	15710	187907
roi2alpha3n4	1859.28	8598	1745848	1343.63	10678	77844
roi5alpha10n8	7200	1616	1131286	7200	1722	51381
roll3000	142.94	1533	167547	169.43	1592	265083
s100	7200	6772	5336606	7200	20800	336603
s250r10	1238.66	5740	2472098	385.95	1776	40935
satellites1-25	7200	26777	18050837	1189.34	23446	299649
satellites2-40	7200	12014	8065563	7200	6424	202647
satellites2-60-fs	7200	2729	2388503	3908.15	3526	137085
sct2	7200	1022860	400560	7200	293456	9753748
seymour	7200	70456	582404	7200	54460	4602308
seymour1	546.86	5350	52008	926.69	6566	719961
sing326	7200	5140	2320876	7200	11037	2361827
sing44	7200	2777	1425384	7200	17547	1529814
snp-02-004-104	7200	25467	3440	7200	13059	29953
sp150x300d	208.4	46452	48981	7200	2730562	608276
sp97ar	7200	111242	11157571	7200	219056	2332050
sp98ar	7200	120188	8154551	7200	176136	1814152

Continued on next page ⟶

sp98ic	406.17	14681	1640253	860.11	65396	475548
sp98ir	38.31	2564	195881	35.76	4184	60778
splice1k1	7200	1962	728038	7200	4859	706038
square41	7200	1013	957930	7200	3815	164859
supportcase18	7200	586969	5501778	7200	883773	6451225
supportcase26	7200	2786384	36608	7200	2115546	22836237
supportcase33	3257.66	14408	4552013	3957.19	61120	681782
supportcase40	2935.75	28328	117477	5548.01	36212	5223062
supportcase42	7200	22725	71973	7200	25769	446823
supportcase6	7200	20249	10180458	7200	155926	2537273
supportcase7	625.06	200	55201	881.4	182	141656
swath1	182.53	22578	283252	99.53	9226	298729
swath3	978.07	142632	598139	1361.56	198236	1901465
tanglegram1	7200	509	565169	7200	624	763733
tanglegram2	59.11	158	72564	664.59	2652	661860
tbfp-network	46.78	118	102721	32.47	152	8586
thor50dday	7200	9152	2174209	7200	11065	1079248
timtab1	7200	3199630	44007	7200	3183369	8296692
tr12-30	7200	550998	8539	7200	472705	3398127
traininstance2	7200	639603	7535083	7200	672494	13349619
traininstance6	7200	872512	2785913	7200	725684	21975990
trento1	1713.32	5682	3493318	7200	77721	954160
uccase12	7200	29603	56507	7200	56418	344708
uccase9	7200	590	262619	7200	1166	952447
uct-subprob	7200	55857	2138655	7200	58103	167963
unitcal_7	1233.28	4252	438975	1116.71	4224	642239
var-smallemery -m6j6	1800.89	417014	305486	3516.02	287906	6700991
vpphard	7200	17236	4695406	7200	129797	935027
wachplan	7200	521228	3442528	7200	327650	29997961
zib54-UUE	3967.82	163672	57429	3943.86	123918	4227522

Table A.2: number of nodes enumerated (n), strong branching iterations (#strong_itrn) and percentage gap (gap) of those instances which could not be solved by both the procedures for a given time limit of 7200 seconds

	SimBranch			Default-Cbc		
Instance	n	#strong_itrn	gap	n	#strong_itrn	gap
50v-10	833564	2168082	3.18	1454959	17890026	3.52
aflow40b	1132095	1452	6.03	556977	45187370	5.89
assign1-5-8	9437195	19272	11.59	2509526	246931754	9.42
atlanta-ip	2786	907208	8.90	3023	1189275	8.90
b1c1s1	150376	189426	31.90	106076	3389556	35.80
bab2	1903	1569398	3.97	5526	138918	3.97
bab5	26684	9446052	0.97	79230	848403	1.02
bab6	4596	3446948	2.23	4605	131650	2.23
beasleyC3	285732	226188	11.07	215416	10438111	11.69
blp-ar98	72223	15669195	0.74	201116	1932681	0.72
bnatt400	46084	2194501	100.00	36771	5169764	100.00
bppc4-08	2303032	2577	2.66	2248360	27508589	2.66
brazil3	5732	3227386	4.17	7304	3218996	4.17
buildingenergy	352	384419	0.00	233	203580	0.00
chromaticindex512-7	10248	2309223	25.00	68510	749293	25.00
cmflsp50-24-8-8	69282	1510384	0.70	36388	4151640	0.72
CMS750_4	67192	257459	0.79	63300	936250	0.79
co-100	4298	1669251	57.44	2085	84494	57.44
cod105	111457	2177796	34.25	79823	11417996	34.28
comp21-2idx	2798	2290435	38.36	3732	1331912	38.36
cost266-UUE	587294	190994	6.88	307727	21411168	6.99
cov1075	918344	267801	11.06	704709	15802997	10.97
csched007	1038155	1467124	12.35	1635149	17236674	14.04
csched008	2394359	674224	1.16	2177516	20811656	1.16
cvs16r128-89	46899	13700550	24.26	68870	10586187	25.61
dws008-01	63354	9186413	65.15	117693	1108765	66.18
eilA101-2	4952	5182640	8.18	15906	329271	6.70
enlight13	494755	97213	66.52	510872	427682	65.36
exp-1-500-5-5	1381906	9850	18.17	672138	53203314	16.05
fast0507	19651	13637157	0.67	196691	1747175	0.61
fastxgemm-n2r6s0t2	288693	44241	88.26	216553	5204311	88.26
germanrr	77308	8922144	1.83	206723	2290545	1.93

Continued on next page ⟶

glass-sc	107265	282356	21.23	72952	2872744	21.31
glass4	1862286	1930	31.16	1176335	42397239	29.87
gmu-35-50	5565066	37495	0.00	5033123	55728038	0.00
graph20-20-1rand	18869	2051346	156.76	14911	2985262	156.76
h80x6320d	204832	25314	4.71	123712	8143952	4.27
ic97_potential	1281952	61101	0.75	1370030	8331548	0.79
icir97_tension	873119	465637	0.19	510390	23914293	0.27
iis-100-0-cov	191081	185718	16.93	132204	6441808	17.99
iis-bupa-cov	104982	157606	12.53	85487	3217323	12.93
iis-pima-cov	57171	296290	6.42	46314	3037337	7.56
lectsched-5-obj	11621	3493839	33.33	15220	2824217	33.33
leo1	297782	8525081	1.19	460363	4734826	1.23
leo2	192245	10959569	1.64	338839	3320975	1.84
lotsize	111746	1906934	44.98	305498	4761766	45.35
mad	9065978	84495	100.00	5673186	181414257	100.00
map16715-04	1483	91038	162.12	2572	448385	160.14
markshare2	171838226	4569	100.00	190661114	119487467	100.00
mc11	170328	357467	18.33	188146	5320395	19.11
milo-v12-6-r2-40-1	213633	223383	9.87	210100	2876704	9.69
momentum1	6416	857667	15.72	11036	1418417	16.96
msc98-ip	706	571906	0.18	1280	826488	0.17
mspp16	1560	67337	6.06	1893	98575	6.06
mushroom-best	62121	313210	71.07	36913	1858583	62.03
n2seq36q	166298	11321696	0.38	350979	2814187	0.38
n3seq24	35075	10611169	0.38	87913	941624	0.38
neos-1337307	83888	7223091	0.03	73611	5093874	0.02
neos-1456979	78935	4679817	7.32	99059	5234655	7.32
neos-2657525-crna	1924971	364927	100.00	10179555	70264336	100.00
neos-2746589-doon	19231	6787719	1.12	119921	1057921	1.12
neos-2978193-inde	430102	25236	1.29	275004	12294185	1.29
neos-3046615-murg	16831515	85997	62.94	3667520	2353094	63.25
neos-3381206-awhea	555853	265849	0.19	723373	4420560	0.19
neos-3555904-turama	6836	4348083	19.45	9741	4021870	19.45
neos-3627168-kasai	1064567	71678	0.59	1287477	5820479	0.61
neos-3656078-kumeu	1424	910079	25.32	1438	1050790	25.32
neos-3754480-nidda	7808244	7147	4409.25	6795519	32224139	4222.72
neos-4300652-rahue	310	351651	93.23	791	913734	93.23
neos-4338804-snowy	1739219	261891	1.63	1390117	20835744	1.63
neos-4387871-tavua	124011	3410945	32.52	84801	11175606	32.78
neos-4532248-waihi	1004	619774	91.88	1018	504577	91.88
neos-4647030-tutaki	5359	10939	0.00	6073	12791	0.00
neos-4722843-widden	651	104449	55.78	759	231751	55.78
neos-4763324-toguru	1073	804486	30.36	1445	458939	30.35
neos-4954672-berkel	610748	48969	20.69	488480	6101484	20.38
neos-5052403-cygnet	2570	2389539	1.30	3309	94382	1.31
neos-5093327-huahum	54395	38683	23.08	50483	2240747	23.08
neos-5107597-kakapo	128610	685690	58.73	73060	5552460	62.50
neos-5188808-nattai	17124	48027	82.77	21120	1371409	94.56
neos-5195221-niemur	436	385154	100.00	1017	851375	100.00
neos-631710	10058	1509899	7.27	11071	478849	7.27
neos-662469	41330	18003053	0.01	209116	1519426	0.01
neos-848589	655	142805	2.65	440	332407	2.65
neos-873061	68691	17157	1.17	72272	3450676	1.17
neos-911970	1407586	4570	4.65	814172	64810519	4.62
neos-934278	8300	3844844	0.19	11962	5589591	0.19
neos17	1045643	200976	48.00	3135269	30448850	58.67
newdano	722081	43293	29.85	527561	8034121	28.43
nexp-150-20-8-5	139969	207585	60.99	93258	3969075	60.06
noswot	23261519	29058	4.88	20838393	64073523	4.88
ns1208400	114707	4896787	100.00	251653	3120866	100.00
ns1688347	8788	1500223	81.48	4633	1363961	81.48
nursesched-medium-hint03	640	344571	50.32	586	31176	50.32
opm2-z10-s4	424	390121	41.04	654	277731	41.04
pigeon-10	2609640	868240	11.11	3109938	18460069	11.11
proteindesign121hz512p9	18447	4767155	2.51	59886	1016773	2.58
proteindesign122trx11p8	17213	3883868	0.92	51870	847627	0.92
pw-myciel4	231683	895950	50.00	192724	17007743	10.00
radiationm18-12-05	61071	2673087	0.01	44104	6810637	0.01
radiationm40-10-02	3867	2556249	0.00	2603	1830401	0.00
rail01	827	818794	16.64	435	23000	16.64
rail507	17036	12100888	0.65	171056	1885423	0.63
ran14x18-disj-8	1662572	193178	4.42	920050	47689977	4.56
rd-rplusc-21	35180	983	99.94	11268	1054913	99.94
reblock115	1854165	484242	0.61	1272312	31183214	0.56

Continued on next page ⟶

rmatr200-p5	7664	253314	27.14	5731	1178196	26.14
rocI-4-11	135518	275291	84.06	164428	5186953	84.06
rocII-5-11	20416	5348991	76.68	35592	2233138	77.48
rococoB10-011000	33024	8081301	13.60	51767	470964	16.38
roi5alpha10n8	1616	1131286	40.64	1722	51381	40.64
s100	6772	5336606	0.77	20800	336603	0.18
satellites2-40	12014	8065563	57.89	6424	202647	57.89
sct2	1022860	400560	0.02	293456	9753748	0.02
seymour	70456	582404	1.84	54460	4602308	1.87
sing326	5140	2320876	0.16	11037	2361827	0.16
sing44	2777	1425384	0.14	17547	1529814	0.00
snp-02-004-104	25467	3440	0.07	13059	29953	0.49
sp97ar	111242	11157571	0.65	219056	2332050	0.66
sp98ar	120188	8154551	0.34	176136	1814152	0.38
splice1k1	1962	728038	317.73	4859	706038	317.73
square41	1013	957930	41.07	3815	164859	41.07
supportcase18	586969	5501778	1.69	883773	6451225	1.69
supportcase26	2786384	36608	16.20	2115546	22836237	16.37
supportcase42	22725	71973	0.09	25769	446823	0.09
supportcase6	20249	10180458	9.13	155926	2537273	9.66
tanglegram1	509	565169	99.90	624	763733	99.90
thor50dday	9152	2174209	58.70	11065	1079248	58.70
timtab1	3199630	44007	36.13	3183369	8296692	35.94
tr12-30	550998	8539	21.69	472705	3398127	21.10
traininstance2	639603	7535083	100.00	672494	13349619	100.00
traininstance6	872512	2785913	100.00	725684	21975990	100.00
uccase12	29603	56507	0.00	56418	344708	0.00
uccase9	590	262619	1.52	1166	952447	1.52
uct-subprob	55857	2138655	6.85	58103	167963	8.60
vpphard	17236	4695406	100.00	129797	935027	100.00
wachplan	521228	3442528	12.50	327650	29997961	12.50

Appendix B

Mathematical Modeling and h-MOLP Instances

B.1 Instances

Table B.1: MOLP instances selected from MOPLIB library for our computational experiment

instance	no. of linear constraints	no. of variables	no. of nonzeros in linear constraints	no. of business objectives	no. of nonzeros in objective function
molp_3_100_20_assignment	20	100	200	3	300
molp_4_729_729_bensolvehedron	729	729	729	4	2612
molp_4_900_60_assignment	60	900	1800	4	3600
molp_9_100_60_mpp	100	60	6000	9	540
molp_10_779_10174_entropy	779	10174	43948	10	12668
molp_10_900_60_assignment	60	900	1800	10	9000
molp_12_21_30_dc	21	30	75	12	22
molp_21_31_138_entropy	31	138	546	21	21
molp_22_43_213_entropy	43	213	863	22	22
molp_23_28_218_entropy	28	218	623	23	1592
molp_27_28_218_entropy	28	218	623	27	1860

B.2 Formulation of MPS for Potato Chips Industry

hMOLP1: lexmin $-\,8.17285714\,AMT020201 - 6.74428571\,AMT020101$

$-\,8.17285714\,AMT010201 - 3.88714286\,AMT010101$

subject to $CP01T01 : 0.1666667\,OP03T01 + 0.000666667\,OP04T01 - CP01BD0T01 = 0,$

$CP01T02 : 0.1666667\,OP03T02 + 0.000666667\,OP04T02 - CP01BD0T02 = 0,$

$CP01T03 : 0.1666667\ OP03T03 + 0.000666667\ OP04T03 - CP01BD0T03 = 0,$

$CP01T04 : 0.1666667\ OP03T04 + 0.000666667\ OP04T04 - CP01BD0T04 = 0,$

$CP01T05 : 0.1666667\ OP03T05 + 0.000666667\ OP04T05 - CP01BD0T05 = 0,$

$CP01T06 : 0.1666667\ OP03T06 + 0.000666667\ OP04T06 - CP01BD0T06 = 0,$

$CP01T07 : 0.1666667\ OP03T07 + 0.000666667\ OP04T07 - CP01BD0T07 = 0,$

$BL04T01 : OP03T01 - XBL04T01 = 0,$

$BL04T02 : OP03T02 + XBL04T01 - XBL04T02 = 0,$

$BL04T03 : -AMT020201 + OP03T03 + XBL04T02 - XBL04T03 = 0,$

$BL04T04 : -AMT020101 + OP03T04 + XBL04T03 - XBL04T04 = 0,$

$BL04T05 : OP03T05 + XBL04T04 - XBL04T05 = 0,$

$BL04T06 : OP03T06 + XBL04T05 - XBL04T06 = 0,$

$BL04T07 : OP03T07 + XBL04T06 - XBL04T07 = 0,$

$BL05T01 : OP04T01 - XBL05T01 = 0,$

$BL05T02 : OP04T02 + XBL05T01 - XBL05T02 = 0,$

$BL05T03 : -AMT010201 + OP04T03 + XBL05T02 - XBL05T03 = 0,$

$BL05T04 : OP04T04 + XBL05T03 - XBL05T04 = 0,$

$BL05T05 : OP04T05 + XBL05T04 - XBL05T05 = 0,$

$BL05T06 : -AMT010101 + OP04T06 + XBL05T05 - XBL05T06 = 0,$

$BL05T07 : OP04T07 + XBL05T06 - XBL05T07 = 0,$

bound: $0 \leq AMT020201 \leq 3600,$

$0 \leq AMT020101 \leq 3600,$

$0 \leq AMT010201 \leq 2160,$

$0 \leq AMT010101 \leq 2160,$

$0 \leq CP01BD0T01 \leq 36,$

$0 \leq CP01BD0T02 \leq 36,$

$0 \leq CP01BD0T03 \leq 36,$

$0 \leq CP01BD0T04 \leq 36,$

$0 \leq CP01BD0T05 \leq 36,$

$0 \leq CP01BD0T06 \leq 36,$

$0 \leq CP01BD0T07 \leq 36.$

Appendix C

Modeling of MPS for the Small-industry Problem

LP1: $\text{obj1} := \min -xd_3^1 - xd_3^2 - xd_3^3 - xd_3^4$

subject to $0.002\, op_t^i - c_t^i \le 0, \quad t = 1, 2, 3, \ldots, T$, and $i = 1, 4, 5, 6, 7, 12$,

$0.01666667\, op_t^3 + 0.01\, op_t^2 - c_t^2 \le 0, \quad t = 1, 2, 3,$

$0.01\, op_t^8 + 0.025\, op_t^9 - c_t^7 \le 0, \quad t = 1, 2, 3,$

$0.01666667\, op_t^{10} + 0.01\, op_t^{11} - c_t^8 \le 0, \quad t = 1, 2, 3,$

$0.025\, op_t^{13} + 0.01\, op_t^{14} - c_t^{10} \le 0, \quad t = 1, 2, 3,$

$op_1^1 - op_1^2 - b_1^1 = 0,$

$op_t^1 - op_t^2 + b_{t-1}^1 - b_t^1 = 0, \; t = 2, 3,$

$op_1^6 - op_1^{10} - b_1^2 = 0,$

$op_t^6 - op_t^{10} + b_{t-1}^2 - b_t^2 = 0, \; t = 2, 3,$

$op_1^5 - op_1^{11} - b_1^3 = 0,$

$op_t^5 - op_t^{11} + b_{t-1}^3 - b_t^3 = 0, \; t = 2, 3,$

$op_1^4 - op_1^3 - b_1^4 = 0,$

$op_t^4 - op_t^3 + b_{t-1}^4 - b_t^4 = 0, \; t = 2, 3,$

$op_1^{11} - op_1^{15} - b_1^5 = 0,$

$op_t^{11} + b_{t-1}^5 - b_t^5 = 0, \; t = 2, 3,$

$op_1^3 - op_1^{16} - b_1^6 = 0,$

$op_t^3 + b_{t-1}^6 - b_t^6 = 0, \; t = 2, 3,$

$op_1^2 - op_1^{17} - b_1^7 = 0,$

$$
\begin{aligned}
& op_t^2 + b_{t-1}^7 - b_t^7 = 0,\ t = 2,\ 3, \\
& op_1^{10} - op_1^{18} - b_1^8 = 0, \\
& op_t^{10} + b_{t-1}^8 - b_t^8 = 0,\ t = 2,\ 3, \\
& op_1^{13} - op_1^{19} - b_1^9 = 0, \\
& op_t^{13} + b_{t-1}^9 - b_t^9 = 0,\ t = 2,\ 3, \\
& op_1^{14} - op_1^{20} - b_1^{10} = 0, \\
& op_t^{14} + b_{t-1}^{10} - b_t^{10} = 0,\ t = 2,\ 3, \\
& op_1^9 - op_1^{21} - b_1^{11} = 0, \\
& op_t^9 + b_{t-1}^{11} - b_t^{11} = 0,\ t = 2,\ 3, \\
& op_1^8 - op_1^{22} - b_1^{12} = 0, \\
& op_t^8 + b_{t-1}^{12} - b_t^{12} = 0,\ t = 2,\ 3, \\
& op_1^{16} - b_3^{13} = 0, \\
& op_1^{17} - b_3^{14} = 0, \\
& op_1^{19} - b_3^{15} = 0, \\
& op_1^{20} - b_3^{16} = 0, \\
& op_1^{18} - b_3^{17} - xd_3^1 = 0, \\
& op_1^{15} - b_3^{18} - xd_3^4 = 0, \\
& op_1^{21} - b_3^{19} - xd_3^3 = 0, \\
& op_1^{22} - b_3^{20} - xd_3^2 = 0, \\
& op_1^7 - op_1^{13} - op_1^{14} - b_1^{21} = 0, \\
& op_t^7 - op_t^{13} - op_t^{14} - b_t^{21} + b_{t-1}^{21} = 0,\ t = 2,\ 3, \\
& op_1^{12} - op_1^8 - op_1^9 - b_1^{22} = 0, \\
& op_t^{12} - op_t^8 - op_t^9 - b_t^{22} + b_{t-1}^{22} = 0,\ t = 2,\ 3, \\
\text{bound: } & 0 \le xd_3^i \le 3000, \quad \text{for all demand item } i = 1,\ 2,\ 3,\ 4, \\
& 0 \le c_t^1 \le 12, \quad t = 1,\ 2,\ 3, \\
& 0 \le c_t^2 \le 12, \quad t = 1,\ 2,\ 3, \\
& \qquad\qquad \vdots \\
& 0 \le c_t^{10} \le 12, \quad t = 1,\ 2,\ 3, \\
& op_t^i, b_t^i \ge 0, \quad i = 1,\ 2,\ 3, \ldots,\ 14, \text{ and } t = 1,\ 2\ , 3, \\
& op_1^i, b_1^i \ge 0, \quad i = 15, \ldots,\ 22.
\end{aligned} \tag{C.1}
$$

Bibliography

[1] Richard S Sutton, Andrew G Barto, et al. Introduction to reinforcement learning. 1998.

[2] Robert Fourer and Dominique Orban. Drampl: a meta solver for optimization problem analysis. *Computational Management Science*, 7(4):437–463, 2010.

[3] Christos H Papadimitriou. On the complexity of integer programming. *Journal of the ACM (JACM)*, 28(4):765–768, 1981.

[4] Ailsa H Land and Alison G Doig. An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008*, pages 105–132. Springer, 2010.

[5] Dimitris Bertsimas and Robert Weismantel. *Optimization over integers*, volume 13. Dynamic Ideas Belmont, 2005.

[6] Alexander Schrijver. On the history of combinatorial optimization (till 1960). *Handbooks in operations research and management science*, 12:1–68, 2005.

[7] George B Dantzig and Mukund N Thapa. *Linear programming 1: introduction*. Springer Science & Business Media, 2006.

[8] Francis Ysidro Edgeworth. *Mathematical psychics: An essay on the application of mathematics to the moral sciences*, volume 10. CK Paul, 1881.

[9] Vilfredo Pareto. *Manual of political economy: a critical and variorum edition*. OUP Oxford, 2014.

[10] Joseph G Ecker and IA Kouada. Finding efficient points for linear multiple objective programs. *Mathematical Programming*, 8(1):375–377, 1975.

[11] Stanley Zionts and Jyrki Wallenius. An interactive programming method for solving the multiple criteria problem. *Management science*, 22(6):652–663, 1976.

[12] Hanif D Sherali and Allen L Soyster. Preemptive and nonpreemptive multi-objective programming: Relationship and counterexamples. *Journal of Optimization Theory and Applications*, 39(2):173–186, 1983.

[13] SF Tantawy and RH Sallam. Multiple objective linear programming (molp) problems with the same objective space. *Journal of Algorithms & Computational Technology*, 3(4):573–581, 2009.

[14] Jasbir S Arora. Multiobjective optimum design concepts and methods. *Introduction to optimum design*, pages 657–679, 2012.

[15] Michael Comelli, Michel Gourgand, and David Lemoine. A review of tactical planning models. *Journal of Systems Science and Systems Engineering*, 17(2):204, 2008.

[16] Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27, 1995.

[17] Tobias Achterberg, Timo Berthold, Thorsten Koch, and Kati Wolter. Constraint integer programming: A new approach to integrate cp and mip. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 6–20. Springer, 2008.

[18] David R Morrison, Sheldon H Jacobson, Jason J Sauppe, and Edward C Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016.

[19] Michel Bénichou, Jean-Michel Gauthier, Paul Girodet, Gerard Hentges, Gerard Ribière, and O Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1(1):76–94, 1971.

[20] Gautam Mitra. Investigation of some branch and bound strategies for the solution of mixed integer linear programs. *Mathematical Programming*, 4(1):155–170, 1973.

[21] David L Applegate, Robert E Bixby, Vasek Chvatal, and William J Cook. *The traveling salesman problem: a computational study*. Princeton university press, 2006.

[22] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[23] Juhani Koski. Multicriteria truss optimization. In *Multicriteria Optimization in Engineering and in the Sciences*, pages 263–307. Springer, 1988.

[24] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

[25] Mokhtar S Bazaraa, John J Jarvis, and Hanis D Sherali. *Linear programming and network flows*. John Wiley & Sons, 2010.

[26] Gurobi. Gurobi optimizer reference manual. `http://www.gurobi.com`. May 29, 2020.

[27] CPLEX. IBM ILOG CPLEX Optimization Studio V12.9.0 documentation. `https://www.ibm.com/support/knowledgecenter/en/SSSA5P_12.9.0/ilog.odms.studio.help/Optimization_Studio/topics/COS_home.html`. July 07, 2020.

[28] Martin Bartusch, Rolf H Möhring, and Franz J Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of operations Research*, 16(1):199–240, 1988.

[29] Markus W Schäffter. Scheduling with forbidden sets. *Discrete Applied Mathematics*, 72(1-2):155–166, 1997.

[30] M Bartusch, Rolf H Möhring, and Franz Josef Radermacher. Design aspects of an advanced model-oriented dss for scheduling problems in civil engineering. *Decision Support Systems*, 5(4):321–344, 1989.

[31] Bruno Escoffier, Martin Milanič, and Vangelis Paschos. Simple and fast reoptimizations for the steiner tree problem. *Algorithmic Operations Research*, 4(2):86–94, 2009.

[32] Davide Bilò, Hans-Joachim Böckenhauer, Juraj Hromkovič, Richard Královič, Tobias Mömke, Peter Widmayer, and Anna Zych. Reoptimization of steiner trees. In *Scandinavian Workshop on Algorithm Theory*, pages 258–269. Springer, 2008.

[33] Claudia Archetti, Luca Bertazzi, and M Grazia Speranza. Reoptimizing the traveling salesman problem. *Networks: An International Journal*, 42(3):154–159, 2003.

[34] Hans-Joachim Böckenhauer and Dennis Komm. Reoptimization of the metric deadline tsp. *Journal of Discrete Algorithms*, 8(1):87–100, 2010.

[35] David Eppstein, Zvi Galil, Giuseppe F Italiano, and Amnon Nissenzweig. Sparsification—a technique for speeding up dynamic graph algorithms. *Journal of the ACM (JACM)*, 44(5):669–696, 1997.

[36] Monika R Henzinger and Valerie King. Maintaining minimum spanning trees in dynamic graphs. In *International Colloquium on Automata, Languages, and Programming*, pages 594–604. Springer, 1997.

[37] Hans-Joachim Böckenhauer, Juraj Hromkovič, and Dennis Komm. Reoptimization of hard optimization problems. In *Handbook of Approximation Algorithms and Metaheuristics, Second Edition*, pages 427–454. Chapman and Hall/CRC, 2018.

[38] Giorgio Ausiello, Vincenzo Bonifaci, and Bruno Escoffier. Complexity and approximation in reoptimization. In *Computability in Context: Computation and Logic in the Real World*, pages 101–129. World Scientific, 2011.

[39] Joseph YJ Chow. Activity-based travel scenario analysis with routing problem reoptimization. *Computer-Aided Civil and Infrastructure Engineering*, 29(2):91–106, 2014.

[40] Herbert Meyr. Simultaneous lotsizing and scheduling by combining local search with dual reoptimization. *European Journal of Operational Research*, 120(2):311–326, 2000.

[41] Hadas Shachnai, Gal Tamir, and Tami Tamir. A theory and algorithms for combinatorial reoptimization. In *Latin American Symposium on Theoretical Informatics*, pages 618–630. Springer, 2012.

[42] Nicola Secomandi and Francois Margot. Reoptimization approaches for the vehicle-routing problem with stochastic demands. *Operations research*, 57(1):214–230, 2009.

[43] TK Ralphs and M Güzelsoy. Duality and warm starting in integer programming. In *The Proceedings of the 2006 NSF Design, Service, and Manufacturing Grantees and Research Conference*, 2006.

[44] Elizabeth John and E Alper Yıldırım. Implementation of warm-start strategies in interior-point methods for linear programming in fixed dimension. *Computational Optimization and Applications*, 41(2):151–183, 2008.

[45] Tiago P Filomena and Miguel A Lejeune. Warm-start heuristic for stochastic portfolio optimization with fixed and proportional transaction costs. *Journal of Optimization Theory and Applications*, 161(1):308–329, 2014.

[46] Jacek Gondzio. Warm start of the primal-dual method applied in the cutting-plane scheme. *Mathematical Programming*, 83(1):125–143, 1998.

[47] Hartmut Stadtler, Hartmut Stadtler, Christoph Kilger, Christoph Kilger, Herbert Meyr, and Herbert Meyr. *Supply chain management and advanced planning: concepts, models, software, and case studies*. Springer, 2015.

[48] R Keith Oliver, Michael D Webber, et al. Supply-chain management: logistics catches up with strategy. *Outlook*, 5(1):42–47, 1982.

[49] Paul Pounder, Gavin Bovell, and Shannelle Pilgrim-Worrell. A review of supply chain management and its main external influential factors. In *Supply Chain Forum: An International Journal*, volume 14, pages 42–50. Taylor & Francis, 2013.

[50] Samuel H Huan, Sunil K Sheoran, and Ge Wang. A review and analysis of supply chain operations reference (scor) model. *Supply chain management: An international Journal*, 2004.

[51] Bernhard Fleischmann, Herbert Meyr, and Michael Wagner. Advanced planning. In *Supply chain management and advanced planning*, pages 81–106. Springer, 2005.

[52] Jens Rohde, Herbert Meyr, Michael Wagner, et al. Die supply chain planning matrix. Technical report, Darmstadt Technical University, Department of Business Administration …, 2000.

[53] ASCP. Association for supply chain management. https://www.ascm.org/. June 10, 2022.

[54] Ken Titmus. Master planning the forgotten, but vital, manufacturing supply chain planning process, 2006. Paper Presented at Sapics 28th Annual Conference and Exhibition, Feb 2, 2021.

[55] Paul Higgins and Jim Browne. Master production scheduling: a concurrent planning approach. *Production Planning & Control*, 3(1):2–18, 1992.

[56] G Guillén, FD Mele, MJ Bagajewicz, A Espuna, and L Puigjaner. Multiobjective supply chain design under uncertainty. *Chemical Engineering Science*, 60(6):1535–1553, 2005.

[57] Fulya Altiparmak, Mitsuo Gen, Lin Lin, and Turan Paksoy. A genetic algorithm approach for multi-objective optimization of supply chain networks. *Computers & industrial engineering*, 51(1):196–215, 2006.

[58] Songsong Liu and Lazaros G Papageorgiou. Multiobjective optimisation of production, distribution and capacity planning of global supply chains in the process industry. *Omega*, 41(2):369–382, 2013.

[59] Tadeusz Sawik. A lexicographic approach to bi-objective scheduling of single-period orders in make-to-order manufacturing. *European Journal of Operational Research*, 180(3):1060–1075, 2007.

[60] Mohamad Sayed Al-Ashhab, Taiser Attia, Shadi Mohammad Munshi, et al. Multi-objective production planning using lexicographic procedure. *American Journal of Operations Research*, 7(03):174, 2017.

[61] GE Vieira and F Favaretto. A new and practical heuristic for master production scheduling creation. *International Journal of Production Research*, 44(18-19):3607–3625, 2006.

[62] C-C Chern and J-S Hsieh. A heuristic algorithm for master planning that satisfies multiple objectives. *Computers & Operations Research*, 34(11):3491–3513, 2007.

[63] Zhengjia Wu, Cheng Zhang, and Xiaoqin Zhu. An ant colony algorithm for master production scheduling optimization. In *Proceedings of the 2012 IEEE 16th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 775–779. IEEE, 2012.

[64] Marcio M Soares and Guilherme E Vieira. A new multi-objective optimization method for master production scheduling problems based on genetic algorithm. *The International Journal of Advanced Manufacturing Technology*, 41(5-6):549–567, 2009.

[65] Lazaros G Papageorgiou and Constantinos C Pantelides. Optimal campaign planning/scheduling of multipurpose batch/semicontinuous plants. 1. mathematical formulation. *Industrial & engineering chemistry research*, 35(2):488–509, 1996.

[66] Peter Loos and Thomas Allweyer. Application of production planning and scheduling in the process industries. *Computers in Industry*, 36(3):199–208, 1998.

[67] VCB Camargo, Franklina Maria Bragion Toledo, and Bernardo Almada-Lobo. Three time-based scale formulations for the two-stage lot sizing and scheduling in process industries. *Journal of the Operational Research Society*, 63(11):1613–1630, 2012.

[68] Josef Kallrath. Planning and scheduling in the process industry. *OR spectrum*, 24(3):219–250, 2002.

[69] Georgios P Georgiadis, Apostolos P Elekidis, and Michael C Georgiadis. Optimization-based scheduling for the process industries: From theory to real-life industrial applications. *Processes*, 7(7):438, 2019.

[70] Marcus Brandenburg and Franz-Josef Tölle. Milp-based campaign scheduling in a specialty chemicals plant: a case study. In *Supply Chain Planning*, pages 1–26. Springer, 2009.

[71] Klaus Neumann, Christoph Schwindt, and Norbert Trautmann. Advanced production scheduling for batch plants in process industries. *OR spectrum*, 24(3):251–279, 2002.

[72] Kumar Rajaram and Uday S Karmarkar. Campaign planning and scheduling for multiproduct batch operations with applications to the food-processing industry. *Manufacturing & Service Operations Management*, 6(3):253–269, 2004.

[73] Christopher Suerie. Campaign planning in time-indexed model formulations. *International Journal of Production Research*, 43(1):49–66, 2005.

[74] Andreas Drexl and Knut Haase. Proportional lotsizing and scheduling. *International Journal of Production Economics*, 40(1):73–87, 1995.

[75] Narasimha B Kamath, Devender Chauhan, Deba Kalyan Mohanty, and Dinesh Damodaran. System and method of solving supply chain campaign planning problems involving major and minor setups, February 24 2015. US Patent 8,965,548.

[76] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

[77] Shie Mannor, Reuven Y Rubinstein, and Yohai Gat. The cross entropy method for fast policy search. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 512–519, 2003.

[78] Xiu Ning and Pingke Li. A cross-entropy approach to the single row facility layout problem. *International Journal of Production Research*, pages 1–14, 2017.

[79] G Alon, Dirk P Kroese, Tal Raviv, and Reuven Y Rubinstein. Application of the cross-entropy method to the buffer allocation problem in a simulation-based environment. *Annals of Operations Research*, 134(1):137–151, 2005.

[80] Gerardo Beruvides, Ramón Quiza, and Rodolfo E Haber. Multi-objective optimization based on an improved cross-entropy method. a case study of a micro-scale manufacturing process. *Information Sciences*, 334:161–173, 2016.

[81] Hossein Jahandideh, Kumar Rajaram, and Kevin McCardle. Production campaign planning under learning and decay. 2016.

[82] Tobias Achterberg. Constraint integer programming. 2009.

[83] Denis Naddef. Polyhedral theory and branch-and-cut algorithms for the symmetric tsp. In *The traveling salesman problem and its variations*, pages 29–116. Springer, 2007.

[84] Ralf Borndörfer, Carlos E Ferreira, and Alexander Martin. Decomposing matrices into blocks. *SIAM Journal on optimization*, 9(1):236–269, 1998.

[85] Jeff T Linderoth and Martin WP Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11(2):173–187, 1999.

[86] David Applegate, Robert Bixby, William Cook, and Vasek Chvátal. *On the solution of travelling salesman problems*. Universität Bonn. Institut für Ökonometrie und Operations Research, 1998.

[87] Gilbert Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, 1992.

[88] J Forrest and R Lougee-Heimerl. CBC (coin-or branch-and-cut) solver. `https://projects.coin-or.org/Cbc`. April 10, 2020.

[89] Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, et al. The scip optimization suite 7.0. 2020.

[90] W. Glankwamdee and Jeff T. Linderoth. Lookahead branching for mixed integer programming. In *ICS 2011*, 2011.

[91] Tobias Achterberg and Timo Berthold. Hybrid branching. *Integration of AI and OR techniques in constraint programming for combinatorial optimization problems*, pages 309–311, 2009.

[92] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535, 2001.

[93] Gregor Hendel. Enhancing mip branching decisions by using the sample variance of pseudo costs. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 199–214. Springer, 2015.

[94] Chu Min Li and Anbulagan Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the 15th international joint conference on Artifical intelligence-Volume 1*, pages 366–371, 1997.

[95] Tobias Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[96] Ashutosh Mahajan. Presolving mixed-integer linear programs. *Wiley Encyclopedia of Operations Research and Management Science. John Wiley & Sons, Inc*, page 44, 2010.

[97] George L Nemhauser, Martin WP Savelsbergh, and Gabriele C Sigismondi. Minto, a mixed integer optimizer. *Operations Research Letters*, 15(1):47–58, 1994.

[98] Matteo Fischetti and Michele Monaci. Backdoor branching. In *IPCO*, pages 183–191. Springer, 2011.

[99] Fatma Kılınç Karzan, George L Nemhauser, and Martin WP Savelsbergh. Information-based branching schemes for binary linear mixed integer problems. *Mathematical Programming Computation*, 1(4):249–293, 2009.

[100] IBM ILOG CPLEX. 12.2: Using the cplex callable library. *Information available at http://www-01. ibm. com/software/integration/optimization/cplex-optimizer*.

[101] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Miplib 2003. *Operations Research Letters*, 34(4):361–372, 2006.

[102] Thorsten Koch, Tobias Achterberg, Erling Andersen, Oliver Bastert, Timo Berthold, Robert E Bixby, Emilie Danna, Gerald Gamrath, Ambros M Gleixner, Stefan Heinz, et al. Miplib 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011.

[103] COR@L. Mip instances. http://coral.ie.lehigh.edu/mip-instances/. June 10, 2022.

[104] Matteo Fischetti and Michele Monaci. Branching on nonchimerical fractionalities. *Operations Research Letters*, 40(3):159–164, 2012.

[105] Timo Berthold and Domenico Salvagnin. Cloud branching. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 28–43. Springer, 2013.

[106] Gerald Gamrath, Tobias Fischer, Tristan Gally, Ambros M Gleixner, Gregor Hendel, Thorsten Koch, Stephen J Maher, Matthias Miltenberger, Benjamin Müller, Marc E Pfetsch, et al. The scip optimization suite 3.2. *ZIB Report*, pages 15–60, 2016.

[107] Pierre Le Bodic and George L Nemhauser. An abstract model for branching and its application to mixed integer programming. *arXiv preprint arXiv:1511.01818*, 2015.

[108] Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[109] Hang Li. Learning to rank for information retrieval and natural language processing. *Synthesis lectures on human language technologies*, 7(3):1–121, 2014.

[110] Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195, 2017.

[111] Norman J Driebeek. An algorithm for the solution of mixed integer programming problems. *Management Science*, 12(7):576–587, 1966.

[112] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[113] Gerald Gamrath, Anna Melchiori, Timo Berthold, Ambros M Gleixner, and Domenico Salvagnin. Branching on multi-aggregated variables. In *CPAIOR*, pages 141–156, 2015.

[114] Jagat Patel and John W Chinneck. Active-constraint variable ordering for faster feasibility of mixed integer linear programs. *Mathematical Programming*, 110(3):445–474, 2007.

[115] Rui Xu and Donald Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678, 2005.

[116] A Gleixner, G Hendel, G Gamrath, T Achterberg, M Bastubbe, T Berthold, PM Christophel, K Jarck, T Koch, J Linderoth, et al. Miplib 2017, 2018.

[117] Stefan Gollowitzer, Luis Gouveia, and Ivana Ljubić. Enhanced formulations and branch-and-cut for the two level network design problem with transition facilities. *European Journal of Operational Research*, 225(2):211–222, 2013.

[118] Tobias Achterberg, Ashish Sabharwal, and Horst Samulowitz. Stronger inference through implied literals from conflicts and knapsack covers. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 1–11. Springer, 2013.

[119] James Renegar. Incorporating condition measures into the complexity theory of linear programming. *SIAM Journal on Optimization*, 5(3):506–524, 1995.

[120] James Renegar. Some perturbation theory for linear programming. Technical report, Cornell University Operations Research and Industrial Engineering, 1993.

[121] Diagnosing ill conditioning. `https://www.ibm.com/support/pages/diagnosing-ill-conditioning`. Accessed: 2022-02-18.

[122] Gurobi Optimization. Gurobi optimizer version 7.0. 2, 2017.

[123] Andreas Lohne and Sebastian Schenker. A problem library for multi-objective linear, multi-objective.

[124] Juhani Koski. Defectiveness of weighting method in multicriterion optimization of structures. *Communications in applied numerical methods*, 1(6):333–337, 1985.

[125] Indraneel Das and John E Dennis. A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems. *Structural optimization*, 14(1):63–69, 1997.

[126] Numeric difficulties. https://www-eio.upc.edu/lceio/manuals/cplex-11/html/usrcplex/solveLP17.html. Accessed: 2022-03-20.

[127] Cirulli Gabriele. 2048, 2014. Accessed: 2021-09-20.

[128] S Kalyanakrishnan and Devanand. Policy encoding as 2-ply search with evaluation function, 2015. Accessed: 2022-09-07.

[129] David Rios Insua. Sensitivity analysis in multi-objective decision making. In *Sensitivity Analysis in Multi-objective Decision Making*, pages 74–126. Springer, 1990.

[130] Roberto Calandra, Jan Peters, and MP Deisenrothy. Pareto front modeling for sensitivity analysis in multi-objective bayesian optimization. In *NIPS Workshop on Bayesian Optimization*, volume 5, 2014.

[131] David Rios Insua and Simon French. A framework for sensitivity analysis in discrete multi-objective decision-making. *European journal of operational research*, 54(2):176–190, 1991.

List of Publications

Conferences and Posters

1. Devanand, R., et al. “Mathematical Modeling of Master Production Schedule with Campaign Planning Constraints.” 2021 *IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*. IEEE, 2021.

2. Devanand, Ashutosh Mahajan, N. Hemachandra “Similarity Based Lexicographic Method for Hierarchical Multiobjective Linear Programs.” 2022 *Poster Presentation at Mathematics of Data Science (MDS) 2022*. SIAM Conference, 2022.

3. Devanand, Ashutosh Mahajan, N. Hemachandra. “Similarity among nodes for the branching decisions in Branch and Bound algorithm.” 2019 *Seminar presentation, Operations Research Society of India (ORSI) at Indian Institute of Management Ahmedabad (IIMA), India, 2019*. ORSI, 2019.

4. Devanand, Ashutosh Mahajan. “Bandit based branching scheme in branch and bound algorithm.” 2019 *Poster presentation, Mumbai, 2016*. Optimization Summit, JDA 2016.

Patents

Issued

1. Devanand, R. “System and Method for Automatic Parameter Tuning of Campaign Planning with Hierarchical Linear Programming Objectives.” U.S. Patent Application No. 17/728,808.

Filed

1. Devanand R., Tushar Shekhar. “Dynamic Switching in Hierarchical LPOPT Solve.” U.S. Patent Application No. 17/858,727.

2. Devanand R., Tushar Shekhar. "System and Method of Auxiliary Model-Supported Combination of Hierarchical Objectives." U.S. Patent Application No. 63/178,884.

3. Devanand R., Tushar Shekhar. "Objective Crunching (ObCrunch) in Hierarchical Optimization of Supply Chain." U.S. Patent Application No. 62/741,516.

4. Devanand R. et al. "Fair-Share Band Optimization using a mixture of Heterogeneous Gaussian." U.S. Patent Application No. 63/051,647.

Acknowledgements

First and foremost, many thanks and immense love to my Divine Mother for her blessings that gave me wisdom and strength to handle challenges in completing my Ph.D. journey, mixed with academic and industry endeavors.

I would like to thank my supervisors, Prof. Ashutosh Mahajan and Prof. N Hemachandra, for all their help and advice with this Ph.D.

I also thank my research progress committee members for their time and advice. I am thankful to Prof. V. Kavitha for her guidance at the initial stage of my Ph.D.

I especially like to thank my external supervisor and manager, Tushar Shekhar, for his consistent support, invaluable advice, and help managing my industry and Ph.D. work.

Additionally, I would like to thank my senior colleagues and friends, Dr. Umakanta Pattanayak, Dr. Parmod Kumar, and Dr. Prashant Palkar, for their research writing guidance. I thank office staff members Abasaheb Molavane and Pramod Pawar for their timely help in office-related work.

I am deeply grateful to my family members for their support, appreciation, and encouragement. I am especially grateful to my parents for being supportive during my Ph.D. and always.

Finally, my biggest thanks to my wife, Jyoti, who has been by my side throughout this journey. I could not have undertaken this journey without her.

Devanand
IIT Bombay
8 May 2023