Modern Methodologies for Practical Discrete Optimization

Ryo Kimura

May 2, 2019

Tepper School of Business
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Willem-Jan van Hoeve
John Hooker
Benjamin Moseley
J. Christopher Beck

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in Operations Research

Contents

Chapter 1

Introduction

Modern discrete optimization problems, especially those motivated by practice, continue to grow in complexity and scale. The development of modern methodologies to address such problems is of paramount importance, and even more so given that advanced analytical tools become increasingly widespread among businesses. This dissertation consists of three projects, commonly aligned in the pursuit of solving and analyzing larger, more complex optimization problems involving discrete variables.

In the first project (Chapter 2), we consider the home healthcare problem, inspired by a real-world hospice care firm's operational need to perform weekly updates to a rolling visitation schedule of patients by home health aides. The need to jointly perform assignment and routing of aides combined with scheduling constraints makes this problem especially challenging. We propose an exact method based on logic-based Benders decomposition (LBBD) that allows the assignment and routing aspects of the problem to be treated independently and show that it is superior to a monolithic MILP model for all but a few instances. We also find that a variation of the standard LBBD method, called Branch-and-Check (B&C), performs better for the instances tested.

In the second project (Chapter 3), we propose a generic framework for robust scheduling problems where the uncertainty set has a "combinatorial" structure that can be efficiently queried. This contrasts with many robust scheduling problems where the uncertainty set is assumed to be polyhedral, or more generally, convex. The idea is to avoid including the whole uncertainty set in the model by dynamically generating the scenarios to only include those that correspond to the "worst-case", which are often fewer in number. We apply our framework to the robust job shop scheduling problem with machine breakdowns and the robust parallel machine scheduling problem with machine breakdowns and show that we can indeed compute a robust optimal solution by only considering a fraction of the entire uncertainty set.

In the third project (Chapter 4), we consider post-optimality analysis of mixed integer linear programming (MILP) problems, where the goal is to provide a syste-

matic method of analyzing the set of near-optimal solutions. Following a line of work initiated by Hadzic and Hooker in 2006 and extended by Serra and Hooker in 2017, we use the notion of sound reduction and sound decision diagrams as a compact and transparent representation of the set of near-optimal solutions. We also propose a novel method for handling continuous variables with decision diagrams and report preliminary results on some MIPLIB instances to investigate the practicality of our procedure.

Chapter 2

A Logic-based Benders Approach for Home Healthcare Scheduling

2.1 Introduction

Home healthcare is one of the world's most rapidly growing industries, due primarily to cost advantages and aging populations. The number of home healthcare aides in the United States, for example, has doubled in the last decade [Spa16]. Home care is not only less expensive than institutional care but offers other advantages. It allows patients to be treated in the comfortable and familiar surroundings of home, which are less stressful than an institutional environment. It reduces the risk of acquiring drug-resistant infections that may spread in hospitals and nursing homes. The increasing availability of portable equipment and online consultation makes home care feasible for a growing range of conditions. Hospice care, which provides palliative rather than curative treatment, is particularly suited for the home. It may consist of a variety of services, including assistance in everyday tasks, nursing care, psychological counseling, physical therapy, religious/spiritual support, and bereavement services for the family.

The cost-effectiveness of home healthcare depends critically on the efficient assignment, scheduling and routing of healthcare aides, whom we call *aides* for short. Aides typically start their work shift at home or a central office, travel directly from one patient to the next, and return to home or office at the end of the shift. Aides must be qualified to service patients to whom they are assigned, and the schedule must observe a number of constraints imposed by the availability of aides, patient needs, work rules, and legal and regulatory requirements. These include time windows for each patient visit and each aide's departure from and return to home base.

We propose an exact method for solving the home healthcare problem that relies on *logic-based Benders decomposition* (LBBD). While many heuristic methods have been proposed for the problem, an exact method is particularly useful when it is necessary to determine what level and type of staffing are adequate to meet existing

or projected patient needs. By maximizing the number of patients that can be served by a given set of aides, one can determine with certainty whether these aides are adequate, or additional aides must be hired and trained. Maximizing the population served also tends to result in less travel time and idle time for aides. The method can be modified to accommodate other objectives as well.

Logic-based Benders decomposition [Hoo00; HO03a] is well suited for this application because the problem naturally decomposes into an assignment task and a scheduling task. The assignment portion of the problem becomes the Benders master problem, leaving the routing and scheduling for the Benders subproblem, which further decouples into a separate problem for each aide. While classical Benders decomposition requires that the subproblem be a linear or nonlinear programming problem [Ben62; Geo72], LBBD generalizes the classical method to accommodate an arbitrary subproblem, such as the scheduling subproblem posed by home healthcare. A variant of LBBD, *branch and check (B&C)*, has the same characteristics, but solves the master problem once rather than repeatedly as in standard LBBD (S-LBBD; [Hoo00; Tho01]). It can be advantageous when the master problem is much harder to solve than the subproblem. We apply both standard LBBD and branch and check to the home healthcare problem.

A logic-based Benders approach has two additional advantages. The subproblem decouples into small scheduling problems that remain roughly the same size as the overall problem grows in size, allowing the algorithm to scale up to real-world applications. In addition, the master problem and subproblem can be solved with methods that are best suited for each. We solve the master problem with mixed integer linear programming (MILP), which is well suited for computing optimal assignments, while we solve the subproblem with the powerful scheduling algorithms in a constraint programming (CP) solver.

Our research was occasioned by a project undertaken for a major home hospice care organization. In this and in many other contexts, a weekly schedule is required, in which each patient is visited a specified number of times each week. The task is to determine which aide serves each patient, on which days of the week, and at what time of day. We therefore formulate a model that schedules patient visits over a given time horizon, with multiple visits per patient if so mandated by the patient care plan. Patients may also require visits from two or more different types of aides. The model can also accommodate a limited number of temporal dependencies between visits, as when patients require two or more aides to be present simultaneously for a joint task. If this kind of teamwork is the norm, however, the scheduling problem no longer decouples, and LBBD may not be an efficient solution method. We tested LBBD on problem instances that do not include such temporal dependencies.

Due to the nature of hospice (where eligible patients have a life expectancy of six months or less if the illness runs its normal course), the patient population is very dynamic. The problem presented to us was to update an existing aide schedule in response to projected changes in the patient population, as this allows the organiza-

tion to anticipate staffing needs. We therefore focus primarily on the computation of a rolling schedule, a task that arises in many other applications as well. This means that when newly admitted patients replace some existing patients in the population, we find aides and visit times for the new patients while allowing the visit times of patients in service to be rescheduled. To maintain continuity of service – a key contributor to quality of service and patient satisfaction – we require that existing patients be served by the same aides on the same days as before. Other types of continuity constraints are easily incorporated into the model.

LBBD is especially well suited for computation of a rolling schedule, because the structure of the decomposition makes the problem much easier to solve when a subset of patients is replaced, even though the visit times are scheduled for the entire population. However, to test LBBD on a problem with very different characteristics, we also applied it to a Danish home care scheduling problem originally studied by [Ras+12]. This application requires solving the problem from scratch rather than on a rolling basis.

We found that LBBD can solve instances of realistic size to optimality, with solution times ranging from a few seconds to a few minutes on nearly all instances, depending on the number of new patients. Branch and check proved to be significantly faster than standard LBBD on the hospice care instances, as might be expected, because the master problem is much harder to solve than the subproblem. Branch and check is also far superior to MILP except on some instances with narrow time windows. Both forms of LBBD are dramatically faster than MILP on the Rasmussen instances. The master problem is easier to solve than the subproblem in some of these instances, and for these, standard LBBD tends to outperform branch and check.

The paper is organized as follows. After a review of previous work in Section 2.2 below, we formulate the home healthcare problem in Section 2.3, along with options for modifying the model. Section 2.4 provides a brief description of LBBD and its application to the home care problem. Section 2.5 states the scheduling subproblem and indicates how Benders cuts are generated and strengthened. Section 2.6 states the master problem and indicates how the Benders model can be altered to accommodate different objective functions. Section 2.7 indicates how branch and check differs from standard LBBD. A key element in the success of LBBD is the inclusion of a subproblem relaxation in the master problem, and we describe three possible relaxations in Section 2.8. Computational results are reported in Section 2.9, which is followed by conclusions and suggestions for future research.

2.2 Previous Work

Due to the difficulty of solving life-sized home healthcare delivery problems, nearly all existing methods are heuristic algorithms. Recent studies have used tabu search [HL09; RH16], pattern or column generation [All+13; CS15], variable neighborhood search [TH11; MMB14], variable neighborhood search combined with scatter

search and other heuristics [Hie+15], constraint programming combined with heuristics [NSS12; Ren+12], an inexact Benders method [CH12], and separate solution of the rostering and scheduling components of the problem [Yal+14].

There are relatively few exact methods. [Red+12] formulated the home healthcare problem with an MILP model but solved only small instances (15 patients). [Cha+09] used a specialized branch-and-bound algorithm to schedule home chemotherapy, but again tested it only on a very small instance (8 patients).

[Ras+12] scaled up to problem instances of realistic size by solving an MILP model of the problem with column generation and a specialized branching scheme. However, only some of the smaller instances (20–80 patients) were solved to optimality within an hour. The remainder were solved after grouping visits into clusters, so as to reduce the number of visits in the model (clustering can, of course, be used in LBBD if desired). This sacrifices optimality, but the authors report that the solutions are optimal or close to optimal on smaller instances that could also be solved optimally.

The results of Rasmussen et al. show that a column generation method is a viable approach to exact solution of the home care problem, at least for smaller real-world instances. A direct comparison with the results we report is difficult, due to differences in the problem solved. They compute a schedule for one day and one visit per patient, while we schedule multiple visits per patient over a time horizon of several days. On the other hand, we reschedule on a rolling basis, and their problem instances include temporal dependencies between visits.

Benders decomposition was introduced by [Ben62] and extended to accommodate nonlinear programming subproblems by [Geo72]. Logic-based Benders decomposition was developed by [Hoo95; Hoo00] and [HO03a]. The computational advantages of LBBD have since been demonstrated in a wide range of applications, partially surveyed by [Hoo12] and [CÇH15]. Guidelines for applying LBBD can be found in these references and [Hoo07]. [CF06] developed a method based on *combinatorial Benders cuts* that is closely related to LBBD and applies to the specific case of an MILP subproblem. [Rah+17] provide an excellent survey of recent developments in Benders decomposition, including LBBD.

Branch and check was introduced by [Hoo00] and first applied by [Tho01], who coined the term "branch and check." The method has received much less attention than standard LBBD, but [Sad04; Sad08] uses it to minimize the weighted number of late jobs on a single machine, and [LV16] use it for vehicle routing on a congested network. [Bec10] compares performance with standard LBBD on several different problems, as well as presenting a variation of branch and check that avoids solving the subproblems under certain circumstances.

This paper is based on methodology presented in a conference paper by [HH16] but goes significantly beyond it. It modifies the decomposition to allow the option of requiring that a patient's visits occur at the same time each day. It further develops the time window relaxation described in the earlier paper, experiments with assignment and multicommodity flow relaxations, and compares standard LBBD with

branch and check. It is also based on a new implementation that uses SCIP and Gecode as MILP and CP solvers, respectively, rather than commercial solvers.

2.3 The Model

We define the home healthcare problem over d days. Each patient j must be visited on v_j days during this time period by an assigned aide having a set Q_j of qualifications. Each visit has a duration of p_j time units and must take place within a time window $[r_j, d_j]$.

We will let binary variable $y_{ijk} = 1$ if aide i is assigned to visit patient j on day k. If there are restrictions on which days patients can be visited, we indicate this with the generic constraint $y \in K$. For example, in a weekly schedule ($d = 7$), patient j may require two visits that must be scheduled on Tuesday and Thursday or Tuesday and Friday. There may be separation constraints to ensure that the visits are spaced evenly throughout the time horizon, particularly when the schedule is cyclic. For example, in a schedule with $v_j = 2$, we may require that the visits be separated by at least two days in the cycle, so that visits on Monday and Saturday would be infeasible.

Each aide i has a set Q'_i of qualifications. On any given day k, aide i begins at a starting location b_i, travels to the home of each assigned patient, and returns to the terminal location b'_i (normally $b_i = b'_i$). The travel time between aide/patient locations j and j' is $t_{jj'}$ time units, based on an optimal route that is calculated in advance. Aide i must leave location b_i during the time window $[r_{b_i}, d_{b_i}]$ and return to location b'_i during $[r_{b'_i}, d_{b'_i}]$. In addition, aide i cannot be on duty more than U_i time units during the scheduling horizon. We will suppose that the aide "clocks in" on arrival at the first patient of the day and "clocks out" on departure from the last patient, but this can be altered if desired.

The remaining variables of the model are as follows. We let binary variable $\delta_j = 1$ if patient j is assigned an aide, and binary variable $x_{ij} = 1$ if aide i is assigned to patient j. We also let integer variable $\pi_{ik\nu}$ denote the νth patient visited by aide i on day k, and real variable s_j denote the time that the visit to patient j starts on each day it occurs. We are therefore supposing that visits to patient j occur at the same time, because this simplifies notation and reflects the real-world situation we modeled. This assumption can easily be relaxed by adding a day index k to the variables s_j and modifying the model in the obvious way.

The problem can be stated as follows:

$$\max \sum_j \delta_j \tag{2.1}$$

$$\sum_i x_{ij} = \delta_j, \ \sum_{i,k} y_{ijk} = v_j \delta_j, \ \ \forall j \tag{2.2}$$

$$y_{ijk} \le x_{ij}, \ \ \forall i, j, k \tag{2.3}$$

$$x_{ij} = 0, \ \ \forall i, j \text{ with } Q_j \not\subseteq Q'_i \tag{2.4}$$

$$y_{ib_ik} = y_{ib'_ik} = 1, \ \ \forall i, k \tag{2.5}$$

$$y \in K \tag{2.6}$$

$$\delta_j, x_{ij}, y_{ijk} \in \{0, 1\}, \ \ \forall i, j, k \tag{2.7}$$

$$n_{ik} = \sum_j y_{ijk}, \ \ \text{all-different}\{\pi_{ik\nu} \mid \nu = 1, \ldots, n_{ik}\}, \ \forall i, k \tag{2.8}$$

$$\pi_{ik\nu} \in \{j \mid y_{ijk} = 1\}, \ \ \forall i, k, \text{ and } \nu = 1, \ldots, n_{ik} \tag{2.9}$$

$$\pi_{ik1} = b_i, \ \pi_{ikn_{ik}} = b'_i, \ \ \forall i, k \tag{2.10}$$

$$r_j \le s_j \le d_j - p_j, \ \ \forall i, j \tag{2.11}$$

$$s_{\pi_{ik\nu}} + p_{\pi_{ik\nu}} + t_{\pi_{ik\nu}\pi_{ik,\nu+1}} \le s_{\pi_{ik,\nu+1}}, \ \ \forall i, k, \text{ and } \nu = 1, \ldots, n_{ik} - 1 \tag{2.12}$$

$$\sum_k \left(s_{\pi_{ik,n_{ik}-1}} + p_{\pi_{ik,n_{ik}-1}} - s_{\pi_{ik2}} \right) \le U_i, \ \ \forall i \tag{2.13}$$

$$s_j \in \mathbb{R}, \text{ for all } j; \ \ \pi_{ikk'} \in \mathbb{Z}, \ \forall i, k, k' \tag{2.14}$$

The objective (2.1) is to maximize the number of patients served. Other objectives are possible, as discussed in Section 2.6. Constraint (2.2) defines δ_j and ensures that patients are visited the required number of times by their assigned aide. Constraint (2.3) says that patients are only visited by aides who are assigned to them. Constraint (2.4) prevents patients from being served by aides without the proper qualifications. Constraint (2.5) ensures that aides visit their starting and ending locations. Constraint (2.6) enforces restrictions on which days visits may be scheduled.

The remainder of the model schedules the aides. This part of the model will appear in the subproblem, which will be solved by constraint programming (CP). Several of the constraints have a form that is peculiar to CP models, which typically contain "global constraints," or high-level constraints that convey information to the solver about special structure in the problem. Constraint (2.8) defines n_{ik}, the number of patients visited by aide i on day k. It also uses the *all-different* global constraint to require the variables $\pi_{ik\nu}$ to take on distinct values. Constraint (2.9) ensures that the patients who are sequenced for a given aide on a given day are in fact assigned to that aide. Constraint (2.10) requires aides to visit their starting location first and ending location last. Note that the variable π has another variable n_{ik} as one of its indices, a standard feature of CP models. Constraint (2.11) ensures visits occur within the required time windows. Constraint (2.12) ensures there is enough time to

travel between locations, and constraint (2.13) enforces the maximum work time for aides. These two constraints likewise contain variable indices.

When a patient requires visits from two or more aides, the model can represent the patient as two or more distinct patients with different requirements. If there are *temporal dependencies* between the visits, they must be enforced by constraints on the start times. For example, if two aides must be present at the same time to perform a task together, we regard the patient as two patients j and j' and add the constraint $s_j = s_{j'}$. If one aide must pick up where the other left off, we can add the constraint $s_j + p_j = s_{j'}$. If the two visits should not overlap, we can give them nonoverlapping time windows, in which case no temporal constraints are necessary. Alternatively, if we want the windows to overlap but not the visits, we can impose a nonoverlapping constraint for the visits – a standard option in CP solvers.

As patients are frequently admitted or discharged from service, it is often desirable to modify the schedule to include the newly admitted patients and remove the discharged patients while rescheduling patients remaining in service as little as possible. The schedule may also be adjusted at the beginning of the day to reflect unavailability of aides or other contingencies. Such updates are easily accommodated by adding constraints to the above model. Suppose that patient j is currently scheduled to be serviced by aide i at time t on days k for $k \in K_j$. To retain this arrangement, we merely set $y_{ijk} = 1$ for $k \in K_j$ in the model and modify the time window $[r_j, d_j]$ to $[t, t + p_j]$. To allow flexibility in the time of day, we leave the time window unchanged. To fix the aide assignment but not the day of the week, we set $x_{ij} = 1$ in the model and leave y_{ijk} unfixed. We can also require that only certain aides take on new patients (or patients whose aides are unavailable). We need only add the constraint $x_{ij} = 0$ for each of the remaining aides i and all new patients j.

2.4 Logic-Based Benders Decomposition

Logic-based Benders decomposition (LBBD) applies to optimization problems of the form $\min\{f(x, y) \mid C(x, y), C(x)\}$, where $C(x, y)$ is a constraint set containing variables x and y, and $C(x)$ a constraint set containing only x. Fixing x to a value $\bar{x}$ that satisfies $C(x)$ defines the *subproblem* $\min\{f(\bar{x}, y) \mid C(\bar{x}, y)\}$. In many applications, including the present one, the subproblem decouples into smaller problems that can be solved separately.

The subproblem is solved to obtain an optimal value v^*, which indicates that cost cannot be less than v^* when x is fixed to $\bar{x}$. We therefore have the bound $f(\bar{x}, y) \geq v^*$ for any y. The solution of the subproblem is analyzed to obtain a *Benders cut*, which is a more general bound $f(x, y) \geq \beta_{\bar{x}}(x)$ that applies for any value of x. The Benders cut is added to a *master problem*, which is solved to obtain the next value $\bar{x}$ to which x is fixed. The kth master problem is

$$\min \left\{ v \;\middle|\; C(x);\;\; v \geq \beta_{x^i}(x),\; i = 1, \ldots, k-1 \right\}$$

where $x^1, \ldots, x^{k-1}$ are the solutions of the first $k-1$ master problems. The optimal value v_k of the master problem is a lower bound on the optimal value of the original problem, and each $\beta_{x^i}(x^i)$ is an upper bound. The algorithm terminates when $v_k = \min\{\beta_{x^i}(x^i) \mid i = 1, \ldots, k-1\}$.

In principle, a Benders cut is found by examining the proof that v^* is optimal in the subproblem. The proof can be regarded as a solution of the *inference dual* of the subproblem. The same proof may yield a useful bound $\beta_{\bar{x}}(x)$ for values of x other than $\bar{x}$. In the special case of a linear programming problem, the inference dual is the linear programming dual. The proof takes the form of dual multipliers, which form the basis for a classical Benders cut. These concepts are discussed further in [Hoo00; Hoo07; Hoo12; HO03a].

In the present application, the objective function depends only on the master problem variables x, so that the problem has the form $\min\{f(x) \mid C(x,y), C(x)\}$. The subproblem becomes a feasibility problem, which may simply be written $C(\bar{x}, y)$. When the subproblem is feasible, the Benders algorithm terminates with an optimal solution. When it is infeasible, the proof of infeasibility may establish infeasibility for values of x other than $\bar{x}$, giving rise to a Benders cut in the form of a constraint $B_{\bar{x}}(x)$ that must be satisfied by any feasible x. One can always use a simple *no-good cut* $x \neq \bar{x}$, but it is desirable to find stronger cuts. The master problem now has the form $\min\{f(x) \mid C(x);\ B_{x^i}(x),\ i = 1, \ldots, k-1\}$.

When the proof of infeasibility is not directly accessible from the solver, a Benders cut must be inferred in some other manner. One approach is to tease out the nature of the infeasibility proof by checking heuristically if the subproblem remains infeasible when some of the premises $x_j = \bar{x}_j$ are dropped. For example, if the subproblem remains infeasible when x_j is fixed to $\bar{x}_j$ only for $j = 1, \ldots, q$, we have the Benders cut $(x_1, \ldots, x_q) \neq (\bar{x}_1, \ldots, \bar{x}_q)$, which excludes more solutions than $x \neq \bar{x}$. This strategy has proved successful in several contexts and will be used here [CÇH15; Hoo05; Hoo06; Hoo07]. A second approach is to deduce from the structure of the subproblem an *analytical Benders cut* that strengthens the no-good cut. Analytical cuts have been used in a wide variety of applications, such as [TBB07; Hoo07; FB09; PT09; ÇH13].

We decompose the home healthcare problem by assigning aides and visit days to patients in the master problem and visit times in the subproblem. Because the objective function depends only on master problem variables, the subproblem becomes a feasibility problem. In previous work, [HH16] decoupled the subproblem into a separate scheduling problem for each aide and each day. However, it is often useful in practice to require a patient's visits on different days to occur at the same time. This couples the daily scheduling problems for each aide, but we nonetheless obtained better computational results than in the earlier paper.

When there are temporal dependencies between visits, the scheduling problems for the aides performing the visits must be coupled. Since each solution of the master problem may assign different aides to patients, the subproblem may decouple diffe-

rently in each Benders iteration. LBBD is most effective when relatively few visits are subject to temporal dependencies, because this allows most of the aides to be scheduled separately.

2.5 Subproblem

The subproblem normally decouples into a separate scheduling problem for each aide. Each scheduling problem checks whether there is a schedule that observes the time windows while taking account of visit durations, travel times, and simultaneity constraints. If not, a Benders cut is generated as described below.

The subproblem formulation consists of the scheduling constraints (2.8)–(2.13) after the daily assignment variables y_{ijk} are fixed to the values $\bar{y}_{ijk}$ they receive in the solution of the previous master problem. The scheduling problem S_i for each aide i is

$$\begin{aligned}
&\text{all-different}\left\{\pi_{k\nu} \mid \nu = 1,\dots,\bar{n}_k\right\}, \quad \forall k \\
&\pi_{k1} = b_i, \ \pi_{k\bar{n}_k} = b'_i, \quad \forall k \\
&r_j \le s_j \le d_j - p_j, \quad \forall j \in \bigcup_k P_{ik} \\
&s_{\pi_{k\nu}} + p_{\pi_{k\nu}} + t_{\pi_{k\nu}\pi_{k,\nu+1}} \le s_{\pi_{k,\nu+1}}, \quad \forall k \text{ and } \nu = 1,\dots,\bar{n}_k - 1 \\
&\sum_k \left(s_{\pi_{k,\bar{n}_k-1}} + p_{\pi_{k,\bar{n}_k-1}} - s_{\pi_{k2}} \right) \le U_i, \quad \forall i \\
&\pi_{k\nu} \in P_{ik}, \quad \nu = 1,\dots,\bar{n}_k
\end{aligned}$$

where $P_{ik} = \{j \mid \bar{y}_{ijk} = 1\}$ and $\bar{n}_k = |P_{ik}|$. If the scheduling problem for two or more aides must be coupled, the variables $\pi_{k\nu}$ become $\pi_{ik\nu}$ as in the main model, and the above constraints are repeated for each of the aides i that must be coupled. Constraints are added to reflect temporal dependencies as described earlier.

The number of variables in the subproblem is limited by the fact that an aide can service only a limited number of patients in a day, say L, regardless of the overall size of the problem instance. Suppose that there are m aides and n patients, and there is no coupling of aides. Then there are at most n variables s_j and at most dL variables $\pi_{k\nu}$ in each of the m scheduling problems, which are solved separately. The number of variables in the subproblem therefore increases linearly with the number of patients.

If scheduling problem S_i is infeasible, we initially generate a no-good cut $\sum_k \sum_{j \in P_{ik}} (1 - y_{ijk}) \ge 1$ that prevents the same set of patients from being assigned to aide i on their corresponding days in subsequent assignments. To strengthen the cut, we re-solve S_i for subsets of P_{ik} using the following heuristic. For each k, we initially set $\bar{P}_{ik} = P_{ik}$, and for each $j \in \bar{P}_{ik}$ we do the following: remove j from $\bar{P}_{ik}$, re-solve S_i, and restore j to $\bar{P}_{ik}$ if the modified S_i is feasible. By replacing P_{ik} with $\bar{P}_{ik}$ in the no-good cut, we obtain a Benders cut that results in significantly better performance.

2.6 Master Problem

The basic master problem consists of constraints (2.1)–(2.7) of the original problem and the Benders cuts generated in all previous iterations, as described above. Because the problem is solved by MILP, the constraints (2.6) on days assignments must be encoded as linear inequality constraints. This is usually not difficult and will be illustrated in Section 2.9.

The master problem contains mnd variables y_{ijk}, mn variables x_{ij}, and n variables δ_j. The number of variables therefore increases linearly with the number of patients, as in the subproblem. Furthermore, when a rolling schedule is computed, many of the variables y_{ijk} and x_{ij} effectively drop out of the problem because they are fixed to 0 or 1.

We augment the master problem with a relaxation of the subproblem, because computational experience in [CÇH15] and elsewhere indicates that including such a relaxation is crucial to obtaining good performance. Normally, the relaxation would contain only master problem variables y_{ijk}, x_{ij} and δ_j rather than variables in the subproblem. This is quite different from a classical relaxation, which contains variables from the problem being relaxed. We present a *time window relaxation*, described in Section 2.8.1 below, that contains only the variables y_{ijk}. A number of time window relaxations for other types of problems are described in [Hoo12].

In an effort to find stronger subproblem relaxations, we also experimented with relaxations that contain variables from the subproblem. In particular, we used the classical assignment relaxation and a multicommodity flow relaxation, described in Sections 2.8.2 and 2.8.3, respectively. The solution values of the subproblem variables are discarded after the master problem is solved, and new solution values obtained when the subproblem is solved.

The decomposition can be modified to accommodate other objective functions, including those defined in terms of subproblem variables. In the latter case, the subproblem becomes an optimization problem, and the Benders cuts become inequalities as described in Section 2.4. Cuts of this kind can be constructed in analogy with the no-good cuts used here. Suppose, for example, that the hourly wage for aide i is c_i, and we wish to minimize total wages. We can convert U_i in the subproblem to a variable, and the scheduling problem for aide i now has an objective of minimizing $c_i U_i$. If z_i^* is the minimum cost found for aide i's schedule, the Benders cut consists of the inequalities

$$z_i \geq z_i^* \sum_k \sum_{j \in P_{ik}} (1 - y_{ijk}), \quad \forall i$$

and the master problem has the objective function $\sum_i z_i$, where z_i is the cost of aide i. This cut imposes the lower bound z_i^* on the cost of aide i if the same patients are assigned to the aide on the same days. The cut can be strengthened heuristically by re-solving the scheduling problems. Heuristics and subproblem relaxations for various objective functions are described, for example, in [Hoo06; Hoo07; Hoo12].

2.7 Branch and Check

Branch and check can be useful when the master problem is much harder to solve than the subproblem. It solves the master problem only once with a branch-and-bound procedure, rather than repeatedly as in standard LBBD. Each time a feasible solution is found at a node of the branching tree, the current values of the master problem variables are sent to the subproblem, and the resulting subproblem is solved. Feasible solutions generated by primal heuristics may also be sent to the subproblem. If the subproblem is infeasible, one or more Benders cuts are generated, and they are enforced throughout the remainder of the branching process.

We will find that the scheduling subproblem generally solves much more rapidly than the master problem in the hospice care instances, which suggests that branch and check may be preferable to standard LBBD for these instances. In fact, branch and check could benefit from relaxations stronger than the time window relaxation, since it may be advantageous to invest more time in solving a master problem that better reflects the original problem. We test these hypotheses in the computational experiments (Section 2.9.3).

2.8 Subproblem Relaxation

We now describe the three subproblem relaxations we investigated for use in the standard LBBD and branch and check algorithms.

2.8.1 Time Window Relaxation

The time window relaxation generalizes a similar relaxation used in [HH16]. It is based on the idea that the total duration of visits and travel assigned to an aide must be no greater than the length of a time interval into which the visits must fit.

For each aide i and day k, define a set $\{[r_{b_i}, \alpha_{ik\ell}] \mid \ell \in L_{ik}\}$ of *backward intervals* that begin with the start of the aide's shift, and a set $\{[\beta_{ik\ell}, d_{b'_i}] \mid \ell \in L'_{ik}\}$ of *forward intervals* that end with the termination of the shift. The time window relaxation requires that the visits that are assigned to aide i on day k, and whose time windows lie inside a given backward interval, must have a total duration that fits in that interval. There is a similar requirement for forward time intervals. In the case of backward intervals, the minimum travel time from the previous visit is included in the visit duration, and in the case of forward intervals, the minimum travel time to the next visit is included.

To state the relaxation more precisely, let $J[t, t']$ be the set of patients whose time windows lie in the interval $[t, t']$, so that $J[t, t'] = \{j \mid [r_j, d_j] \subseteq [t, t']\}$. Let the *backward augmented duration* p'_{ijk} for a patient j, aide i and day k be the visit duration p_j plus the minimum travel time from the previous visit, which may be the aide's origin base. The *forward augmented duration* p''_{ijk} is p_j plus the minimum travel

time to the next visit, which may be the aide's terminal base. So we have

$$p'_{ijk} = p_j + \min\left\{t_{b_i j}, \min_{j' \in J_{ik}} \{t_{j'j}\}\right\}, \quad p''_{ijk} = p_j + \min\left\{\min_{j' \in J_{ik}} \{t_{jj'}\}, t_{jb'_i}\right\}$$

where J_{ik} is the set of patients that are already assigned aide i on day k, or that have not yet been assigned to an aide. Thus the backward augmented duration is a lower bound on the time required to reach and carry out a visit, and similarly for the forward augmented duration.

We now observe that the sum of the backward augmented durations of visits in $J[r_{b_i}, \alpha_{ik\ell}]$ must be at most the width of the backward interval $[r_{b_i}, \alpha_{ik\ell}]$, and similar for any forward interval:

$$\sum_{j \in J[r_{b_i}, \alpha_{ik\ell}]} p'_{ijk} y_{ijk} \leq \alpha_{ik\ell} - r_{b_i}, \quad \ell \in L_{ik}; \qquad \sum_{j \in J[\beta_{ik\ell}, d_{b'_i}]} p''_{ijk} y_{ijk} \leq d_{b'_i} - \beta_{ik\ell}, \quad \ell \in L'_{ik} \tag{2.15}$$

This is because the visits and travel to each visit must fit between the beginning of the aide's shift and the end of the backward interval, and similarly for a forward interval. Inequalities (2.15), collected over all aides i and days k, comprise a time window relaxation.

To obtain tighter inequalities (2.15), the backward and forward intervals should be chosen to have a large *density*. That is, the visits that can take place within them should have a large total duration relative to the width of the interval. To accomplish this, we need only consider the backward intervals $[r_{b_i}, d_j]$ and the forward intervals $[r_j, d_{b'_i}]$ for all patients j. The corresponding densities are

$$\rho_j = \frac{1}{d_j - r_{b_i}} \sum_{j' \in J[r_{b_i}, d_j]} p'_{ij'k}, \qquad \rho'_j = \frac{1}{d_{b'_i} - r_j} \sum_{j' \in J[r_j, d_{b'_i}]} p''_{ij'k}$$

respectively. We now let the set L_{ik} of backward intervals contain those intervals $[r_{b_i}, d_j]$ for which ρ_j is sufficiently large, and similarly for the set L'_{ik} of forward intervals.

The inequalities (2.15) are still fairly weak when scheduling all patients from scratch, because the shortest travel time from the last (or next) visit is a weak bound on the actual travel time. However, they are more effective when scheduling on a rolling basis, because the shortest travel time is computed only over patients who are already assigned aide i on day k or are unassigned.

Additionally, we add the following relaxation of the "no overtime" constraint, which states that the combined duration of all patients visited by an aide cannot exceed the aide's work limit:

$$\sum_j v_j p_j x_{ij} \leq U_i, \quad \forall i \tag{2.16}$$

2.8.2 Assignment Relaxation

The second relaxation we considered is based on the assignment relaxation of the traveling salesman problem. We define a new binary variable $w_{ijj'k} = 1$ if aide i visits patient j immediately prior to patient j' on day k. We add the constraints

$$w_{ijb'_ik} + \sum_{j' \neq j} w_{ijj'k} = w_{ib_ijk} + \sum_{j' \neq j} w_{ij'jk} = y_{ijk}, \quad \forall i, j, k \tag{2.17}$$

$$w_{ib_ijk} + \sum_{j' \neq j} w_{ij'jk} = w_{ijb'_ik} + \sum_{j' \neq j} w_{ijj'k}, \quad \forall i, j, k \tag{2.18}$$

plus similar constraints in which j and/or j' is a home base. In addition, we include a simple feasibility constraint based on the total hours spent traveling/working in a day

$$\sum_j \left(t_{b_ij} w_{ib_ijk} + t_{jb'_i} w_{ijb'_ik} + p_j y_{ijk} + \sum_{j' \neq j} t_{jj'} w_{ijj'k} \right) \leq d_{b'_i} - r_{b_i}, \quad \forall i, k \tag{2.19}$$

a strengthened version of the inequalities (2.15) from the time window relaxation

$$\begin{aligned} &\sum_{j \in J_{i\ell}} \left(t_{b_i,j} w_{ib_ijk} + p_j y_{ijk} + \sum_{\substack{j' \in J_{i\ell} \\ j' \neq j}} t_{j'j} w_{ij'jk} \right) \leq \alpha_{i\ell} - r_{b_i}, \quad \ell \in L_i \\ &\sum_{j \in J'_{i\ell}} \left(p_j y_{ijk} + t_{jb'_i} w_{ijb'_ik} + \sum_{\substack{j' \in J'_{i\ell} \\ j' \neq j}} t_{jj'} w_{ijj'k} + \right) \leq d_{b'_i} - \beta_{i\ell}, \quad \ell \in L'_i \end{aligned} \tag{2.20}$$

as well as a strengthened version of the inequalities (2.16)

$$\sum_{j,k} p_j y_{ijk} + \sum_{j' \neq j} t_{jj'} w_{ijj'k} \leq U_i, \quad \forall i \tag{2.21}$$

2.8.3 Multicommodity flow relaxation

The third relaxation is based on the well-known multicommodity flow model for the vehicle routing problem with time windows [CL06]. In addition to including all variables and constraints of the assignment relaxation, we include the variables s_{ijk} and the constraints

$$s_{ij'k} \geq s_{ijk} + p_j + t_{jj'} - M_{jj'}(1 - w_{ijj'k}), \quad \forall i, j, j', k \tag{2.22}$$

$$r_j \leq s_{ijk} \leq d_j - p_j, \quad \forall i, j, k \tag{2.23}$$

plus similar constraints in which j and/or j' is a home base. Here $M_{jj'} = \max\{0, d_j + p_j + t_{jj'} - r_{j'}\}$. The inequalities (2.19) and (2.20) are also strengthened slightly, by replacing r_{b_i} with s_{ib_ik} and $d_{b'_i}$ with $s_{ib'_ik}$.

If we constrain the variables $w_{ijj'k}$ to be binary (rather than continuous), the multicommodity flow relaxation becomes an MILP formulation of the original problem.

2.9 Computational Results

We tested standard LBBD (S-LBBD) and branch and check (B&C) on three datasets. One consists of real-world data provided by a major hospice care organization, one is obtained by modifying these data, and one consists of Rasmussen instances. We solved instances in the first two datasets on a rolling basis, and the instances in the third from scratch.

2.9.1 Hospice Care Instances

The task presented to us was to update an existing schedule so as to accommodate projected changes in the patient population. In particular, management wished to determine whether a given staff was adequate to serve the new population while meeting all requirements. We therefore maximized the number of patients that can be served by a specified work force with specified qualifications. The aide assignments and scheduled days of the week were fixed for patients in service, but the time of day could be rescheduled.

The organization maintains a weekly schedule for each region it serves; some regions provide service seven days per week while others offer aide visits on weekdays only. In our dataset, visits are scheduled on weekdays only. The patients required multiple visits per week, in accordance with their plan of care; per patient request, these were all scheduled at exactly the same time of day. When there are two visits per week, consecutive visits should be separated by at least two days, and when there are three visits per week, these visits should be separated by at least one day. This was enforced in the master problem by replacing the generic constraint (2.6) with

$$y_{ijk} + y_{ij,k+\tau} \leq 1, \quad \forall i, j \text{ with } v_j \in \{2, 3\}, \ \forall \tau, k \text{ with } 1 \leq \tau \leq 4 - v_j, \ 1 \leq k \leq 5$$

When formulating the time window relaxation, we noted that almost all the time windows either span most of the morning or most of the afternoon. It was therefore natural to use one backward interval ending at noon, and one forward interval beginning at noon, for each aide i. Thus we set $L_i = L'_i = \{1\}$ and $\alpha_{i1} = \beta_{i1} =$ noon for each i. This choice of α and β turns out to yield the highest-density non-trivial backward and forward interval for almost every aide i, where density is defined as in Section 2.8.1, and a nontrivial interval is one that includes and excludes at least one visit.

To obtain an initial schedule, we ran a greedy heuristic on an 80-patient population using 20 aides. Since the heuristic could only schedule 48 patients, we ran the standard LBBD algorithm on 60 of these patients, including 40 pre-scheduled by the greedy heuristic and 20 treated as new patients. They collectively required 270 visits, since the patients required between 2 to 5 visits per week. LBBD scheduled all of the new patients using 18 aides. The resulting 60-patient schedule was used as a starting point for computational tests. It is better than a heuristic schedule but worse than an optimal one, as one might expect when scheduling on a rolling basis.

We ran the tests for different rates of patient turnover in the 60-patient population. One instance was generated for each number $n = 8, \ldots, 25$ of new patients, where the new patients are assumed to be the last n patients in the list of 60. We designated 8 of the 18 aides as available to cover the new patients (along with their pre-assigned patients), because a minimum of 9 aides were required in nearly every instance. This allowed us to test computational performance near the phase transition for the problem. We set a maximum time limit of one hour.

2.9.2 Implementation

We implemented the algorithms using SCIP version 3.2.1 [Ach09] and the CP solver Gecode version 4.4.0 [Gec16]. The master problem was solved by SCIP, and the scheduling subproblems by Gecode. The SCIP presolver removes variables in the master problem that are fixed to 0 or 1 by preassignments. The scheduling problems were formulated with a combination of Hamiltonian path constraints, unary resource constraints, and element constraints. The problems were solved with branch-and-bound search, first branching on sequence variables and then on start-time variables. We implemented S-LBBD via a custom dialog (for SCIP), which made calls to SCIP and Gecode to solve the master problem and solve the subproblem/generate Benders cuts, respectively.

We implemented B&C by incorporating into the master problem an additional constraint that enforced feasibility of the subproblems. We implemented a custom constraint handler for this constraint, which made calls to Gecode to determine feasibility of the subproblem and generated Benders cuts accordingly. We generated cuts for feasible solutions found by primal heuristics, as well as for those obtained in the branching process, because this proved to accelerate solution significantly. The solvers were run in Arch Linux on a laptop with an Intel Core i5 processor and 7.75 GB RAM.

We formulated an MILP model for the problem by modifying the well-known multicommodity flow model for the vehicle routing problem with time windows [Des+88; DL91; Cor+07]. The model consists of (2.1)–(2.7), (2.17)–(2.18), and (2.22)–(2.23). Because there are mn^2d variables $w_{ijj'k}$, the number of variables increases quadratically with the number of patients. This remains the case for a rolling schedule, because all patient visits can be resequenced even when many staff and day assignments are fixed. Thus while preassignments in a rolling schedule make the Benders master problem significantly smaller, they have relatively little effect on the size of the MILP model, in which most of the variables are sequencing variables $w_{ijj'k}$.

SCIP uses reliability branching and pseudocosts by default [Ach09]. However, we turned off reliability branching for the B&C master problem, because in this context, SCIP 3.2.1 occasionally attempts to branch on a variable whose value has become fixed, throwing an error. This is likely due to the pseudocost calculations becoming invalid over time as B&C generates global cuts. The removal of reliability

Table 2.1: Solution times for 60-patient hospice care instances requiring 270 visits

New patients	New visits	Patients covered	MILP Time (s)	S-LBBD Iters	S-LBBD Time (s)	B&C Time (s)
8	40	60	43.6	7	3.17	**0.63**
9	45	59	41.0	13	5.99	**0.71**
10	50	59	46.6	7	3.27	**0.74**
11	55	59	53.3	11	5.63	**0.70**
12	60	59	53.2	12	6.49	**1.30**
13	65	59	63.0	21	12.3	**1.11**
14	70	58	113	84	72.3	**9.28**
15	75	58	223	86	77.0	**9.78**
16	80	58	844	91	98.5	**43.5**
17	85	59	1591	93	106	**31.1**
18	90	58	3017	116	202	**62.0**
19	95	58	1189	119	388	**90.0**
20	100	57	1016	124	1251	**600**
21	105	58	923	168	1272	**380**
22	110	58	*	217	951	**523**
23	115	58		264	*	**2092**
24	120	*				

*Computation time exceeded one hour.

branching for B&C appears otherwise to be of little consequence, because the results are very similar with and without reliability branching on instances where no error is generated.

2.9.3 Results for Hospice Care Instances

Computational results for the hospice care instances appear in Table 2.1. The table shows the number of new patients in each problem instance, as well as the number of patients covered in the optimal solution. Computation times are indicated for all three methods, as well as the number of Benders iterations for S-LBBD. Although we tested the Benders methods using all three relaxations, Table 2.1 shows results for the time-window relaxation only, because it proved to be by far the most effective. Computation times for MILP are those obtained by SCIP.

Both S-LBBD and B&C are faster than MILP on nearly every instance, and B&C is consistently superior to S-LBBD. In fact, B&C is almost always between one and two orders of magnitude faster than MILP. These results indicate that a Benders method can scale up to problem instances of realistic size. Patient records indicate that a 5–8% turnover per week is typical in practice. Therefore B&C allows staff planning as much as 6 weeks in advance for 60 patients collectively requiring 270 visits, with computation times ranging from less than a second to ten minutes,

Table 2.2: Percent of solution time devoted to subproblem

	S-LBBD		B&C	
Instances	Avg	Max	Avg	Max
Original 60-patient instances	0.1	0.2	1.4	3.9
Narrow time windows	0.1	0.1	2.8	6.0
Fewer visits per patient	0.0	0.1	1.7	3.5
Rasmussen, weighted objective	0.4	0.8	6.3	13.6
Rasmussen, covering objective	1.2	1.5	85.6	99.7

depending on the number of new patients. This is adequate for many if not most hospice care situations.

As indicated by Table 2.2, solution of the subproblem consumes less that one percent of the S-LBBD solution time for the hospice care instances. The superior performance of B&C over S-LBBD is therefore consistent with our hypothesis that B&C may benefit from fast solution of the subproblem.

We also hypothesized that B&C may benefit from including tighter relaxations in the master problem, such as the assignment and multicommodity flow relaxations discussed earlier. Table refta:bNc compares the performance of these two relaxations with the time window relaxation. The tighter relaxations lead to much worse performance, refuting the hypothesis. This might be explained by the fact that the tighter relaxations result in many fewer Benders cuts, and in fact none at all for the multicommodity flow relaxation. Since a cut is generated at each feasible node of the search tree at which the subproblem is infeasible, this indicates that the search discovers fewer feasible nodes when the relaxation is tighter. This is presumably because the tighter bound allows backtracking at a higher level in the tree. Because there are fewer cuts, less information is obtained from the subproblem, and in fact no information in the case of the multicommodity flow relaxation. Evidently, the reduced information flow results in poorer performance.

Given these results, one might question whether even the time window relaxation is helpful, especially when patient time windows span half a day as in the test instances. Table refta:relax reveals that the time window relaxation yields a significant, if not dramatic, reduction in computation time. It should therefore be included in the master problem. Table refta:relax also shows the advantage of generating cuts from feasible solutions obtained by primal heuristics, rather than solely from those obtained in the branching process.

2.9.4 Modified Hospice Care Instances

To clarify further the effect of problem structure on the performance of S-LBBD and B&C, we solved two modifications of the hospice care problem.

The first modification uses much narrower patient time windows. We replaced

Table 2.3: Solution time (s) for modified hospice care instances

New patients	Patients covered	Narrow time windows MILP	S-LBBD	B&C	New patients	Patients covered	Fewer visits per week MILP	S-LBBD	B&C
8	60	53.1	10.1	**1.37**	12	58	58.9	17.0	**0.91**
9	59	40.0	11.3	**1.13**	13	58	58.6	20.5	**1.05**
10	59	40.5	17.9	**1.44**	14	58	71.3	30.6	**1.33**
11	59	41.8	22.6	**1.75**	15	58	123	80.0	**1.32**
12	59	43.4	28.8	**1.11**	16	58	167	259	**1.94**
13	59	42.3	28.3	**1.41**	17	58	253	521	**1.79**
14	59	45.2	62.8	**2.97**	18	58	3357	472	**3.45**
15	59	47.0	69.0	**5.25**	19	58	2364	710	**3.23**
16	59	59.5	96.5	**3.67**	20	59	1518	519	**5.36**
17	59	106	233	**11.0**	21	59	1811	759	**4.52**
18	58	127	349	**69.1**	22	59	3636	717	**5.29**
19	58	**137**	425	164	23	60	*	767	**3.87**
20	57	**153**	557	160	24	60		1990	**3.48**
21	57	**171**	993	437	25	60		2040	**91.6**
22	57	**254**	997	1818	26	60		2577	**4.57**
23	58	**524**	*	*	27	60		2693	**376**
24	58	**903**			28	60		4200	**3834**
25	58	**2369**			29	60		*	*
26	*	*							

*Computation time exceeded one hour.

the original time windows with time windows centered around each patient's visit as scheduled in the initial heuristic solution. We then set the length of the time window to be twice that of the visit duration.

The second modification is inspired by the fact that over 80% of patients require five visits per week in the original dataset. This was the situation as presented by the company, but one might ask how performance differs when there are fewer visits per week. We therefore changed the number of required visits per week for each patient to a uniformly drawn random number from 1 to 5, with the duration of each visit is kept the same. We ran the greedy heuristic to produce a new initial schedule.

The results appear in Table 2.3. For the instances with narrow time windows, the pure MILP formulation actually outscales S-LBBD and B&C. This is perhaps not surprising, because the time windows are so narrow that their position already determines the schedule to a great extent, and because scheduling is the more difficult task for MILP.

The instances with fewer visits per week are less constrained and therefore further from the phase transition. To correct for this, we reduced the number of available aides to 6, which is enough to cover all but one patient in the optimal solutions. B&C is far superior to MILP on these instances, and it remains faster than S-LBBD as

well.

These results suggest that the advantage of Benders methods is robust, except when time windows become narrow enough to severely constrain the schedule.

2.9.5 Rasmussen Instances

[Ras+12] provided us four real-world instances obtained from two Danish municipalities. The task is to assign a given set of crews, each containing members with specific skills, to a population of patients who require certain skills. The municipalities did not provide temporal dependencies with the instances, as they were handled on an ad-hoc basis, but Rasmussen et al. added a set of dependencies to test their algorithm adequately. We did not include them because our method is designed for problems without temporal dependencies. Rasmussen et al. solved the problem over a time horizon of a single day, while we solved an equivalent 5-day problem that requires that each patient be visited every day at the same time.

The original objective of the Rasmussen instances is to minimize a weighted sum of travel cost, matching costs, and number of uncovered patients. The weights are adjusted so that as many patients as possible are covered, after which matching costs are minimized, followed by travel costs. The matching costs are indicated by giving each crew-patient pair a cost (positive for an undesirable match and negative for a desirable one). We solved each instance twice: once while minimizing a weighted sum of the matching cost and number of uncovered patients, and again while maximizing the number of patients covered.

For the time window relaxation, we explicitly found the best forward interval and best backward interval for each problem instance during the pre-processing phase via an exhaustive search over all reasonable breakpoints (i.e., the ends and starts of task/patient time windows).

The number of patients in the four instances hh, ll1, ll2, and ll3 are 150, 107, 60, and 61, respectively. We found that MILP could not come close to solving any of these, and the MILP model for hh was in fact too large to load into the solver. To allow a meaningful comparison with MILP, we reduced the number of patients to 30 in each instance, keeping the number of crews the same. At this point, MILP could solve two of the instances with the weighted objective within an hour, albeit none with the covering objective. As Table 2.4 indicates, the Benders methods are dramatically superior to MILP, easily solving all 8 instance-objective combinations. Interestingly, standard LBBD is usually faster than B&C when the covering objective is used. This is again consistent with the hypothesis that B&C is preferable to S-LBBD only when the subproblem solves rapidly, because in these instances, the subproblem consumes a much larger fraction of solution time than in other instances (Table 2.2).

Table 2.4: Solution time (s) for modified Rasmussen instances

			Weighted objective			Covering objective		
Instance	Patients	Crews	MILP	S-LBBD	B&C	MILP	S-LBBD	B&C
hh	30	15	*	3.16	**1.41**	*	**23.3**	441
ll1	30	8	*	1.74	**0.43**	*	108	**1.41**
ll2	30	7	2868	1.56	**0.32**	*	**1.38**	6.45
ll3	30	6	1398	2.16	**0.30**	*	**3.07**	5.98

*Computation time exceeded one hour.

2.10 Conclusions and Future Work

We developed an exact solution method for the home healthcare problem using logic-based Benders decomposition (LBBD). We formulated the problem so as to maximize the number of patients served by a given staff with given qualifications, while taking into account patient requirements, travel time, and scheduling constraints. We create a schedule spanning several days during which patients may receive multiple visits from the same healthcare aide, or visits from multiple aides. Unlike most competing methods developed for this problem, LBBD computes an optimal schedule and therefore allows planners to determine with certainty whether a given work force can serve a given patient population. We tested both standard LBBD and a variant of LBBD (branch and check) that solves the master problem only once.

Based on computational tests in a real-world setting, we conclude that LBBD, and in particular branch and check, can solve a problem of realistic size when scheduling on a rolling basis and there are no temporal dependencies between visits. By contrast, mixed integer/linear programming does not scale up, due to growth in the size of the model, except when time windows are so narrow that their position largely determines the schedule. LBBD has the advantage that the scheduling component of the problem breaks down into small scheduling problems that remain roughly constant in size as the overall problem size increases. In addition, LBBD is particularly well suited to computing a rolling schedule, because the structure of the decomposition makes the problem much easier to solve when only a subset of the patient population is replaced. In one real-world context, however, LBBD easily solved instances from scratch that were intractable for MILP.

Branch and check is faster than standard LBBD on most of the instances tested. However, when the subproblem is harder to solve than the master problem, standard LBBD tends to be faster. Branch and check also benefits from Benders cuts generated from feasible solutions found by primal heuristics. An issue for future research is whether such cuts could accelerate standard LBBD.

Even though branch and check solves the master problem only once, it does not benefit from adding variables to the master problem to create a tighter relaxation of the subproblem. The reason for this appears to be that an overly tight relaxa-

tion results in the creation of fewer Benders cuts during the branching process, and therefore too little information flow from the scheduling subproblem to the master problem. This observation could have implications for future applications of branch and check.

Chapter 3

Robust Scheduling with Combinatorial Uncertainty Sets

3.1 Introduction

Many operational problems take place in an uncertain environment. For example, in machine scheduling problems we need to allocate jobs to machines over time, where job durations, machine availability, and demand may all be uncertain. Similarly, in routing problems we need to determine a sequence of locations to visit and service, where travel time and service time may be uncertain. There are various ways to handle the uncertainty when designing solutions to such problems. In the context of combinatorial optimization, two common approaches are stochastic programming and robust optimization. Stochastic programming dates back to the 1950s [Dan55], and utilizes random variables with known or estimated distributions to represent the uncertainty. Oftentimes, recourse models are added to evaluate the expected outcome of the solution once a set of deterministic variables has been fixed. Stochastic programming can therefore be used to find solutions that are optimal in expectation.

Robust optimization, on the other hand, aims to find solutions that offer protection against (disruptive) uncertain events. For example, in the context of machine scheduling a robust solution may be desired to avoid a recourse action that would require a substantial re-scheduling of jobs in case of an event such as a machine breakdown.

In robust optimization, the uncertain data is represented by an *uncertainty set*, which characterizes all possible realizations [BN02], or a subset thereof. For example, we may create an uncertainty set whose possible realizations correspond to a 95% service level. The goal is to find a feasible solution that achieves the best possible objective value with respect to the worst-case realization in the uncertainty set. Much of the robust optimization literature has focused on well-behaved uncertainty sets, such as polyhedral uncertainty sets in the context of linear programming models. In such situations, it is known that the original (LP) model can be transformed to a

robust version that is only a polynomial factor larger [BEN09]. As a consequence, such robust models are often efficiently solvable in practice.

When the uncertainty set is *combinatorial* however, it is no longer possible (in general) to derive a robust model of polynomial size. Practitioners therefore often resort to heuristic robust approaches that may provide useful solutions, but for which the quality cannot be assessed. Our goal in this work is to develop an alternative, exact, approach to robust optimization with a combinatorial uncertainty set. Such an exact approach can, e.g., validate the performance of other heuristic methods. Moreover, we show in this work how we can deploy our exact method as a systematic heuristic with provable bounds on its performance.

One immediate approach to solving robust optimization problems is to explicitly add all possible realizations of the uncertainty set (or scenarios) to the model. Naturally, this is only practical for problems with a small number of scenarios. Instead, our approach is based on a scenario-generation process. The underlying hypothesis is that only a relatively small number of scenarios are required to find a provably optimal solution, even if the uncertainty set contains a large number (perhaps exponentially many) of scenarios. We use a logic-based Benders method [HO03b] in which the master problem finds the best solution with respect to a subset of scenarios, and the subproblems identify new worst-case scenarios to be added to the master's subset. We terminate when no scenario exists that deteriorates the current best solution.

As a case study, we consider the job-shop scheduling problem in which the machines are subject to uncertain breakdowns. The goal is to find a fixed sequence of activities for each machine that provides the optimal solution with respect to the worst-case machine breakdowns in the uncertainty set. As optimization criteria we consider the minimum makespan (the completion time of the latest activity in the schedule), and the sum of the weighted completion times, which are among the most common objectives in scheduling.

Our experimental results do confirm that a fraction of scenarios, about 16% (and in some cases much fewer) suffices to find the optimal robust solution and prove it is optimal. Furthermore, we provide an analysis of the potential gains in practice. When the objective is to minimize makespan the benefits of the robust solution relative to the non-robust optimal solution are limited. For the sum of the weighted completion times, however, the robust solutions can be up to 15% better than the non-robust solutions in terms of worst-case objective.

3.2 Related Work

3.2.1 Robust Scheduling

Robust scheduling has been studied since the 1990s. Buttazzo and Stankovic [BS93] developed a robust variant of the earliest-deadline-first policy for preemptible dynamic scheduling. Daniels and Kouvelis [DK95] formalized worst-case robust single machine

scheduling for various levels of robustness and gave some of the first hardness results for several robust scheduling problems. Both of these papers considered robustness w.r.t. processing time uncertainty.

Since then, many solution methods have been proposed for solving robust scheduling problems, with the most attention focused on the case of a single machine with processing time uncertainty. We can classify these methods into three broad categories: (1) exact methods, which seek to find and prove the optimal solution to the problem, (2) approximation methods, which seek to find good approximations to an often intractable model of the problem, and (3) heuristic methods, which seek to quickly find high quality solutions at the cost of abandoning formal guarantees on optimality.

One of the most common exact methods utilized in the literature are branch-and-bound methods. Most incorporate some bounding procedure into a branch-and-bound tree which generates partial schedules. In fact, one of the first exact algorithms for solving robust single machine problems was a branch-and-bound scheme given by Daniels and Kouvelis [DK95]. A more sophisticated branch-and-bound procedure utilizing the concept of dominance is proposed by Briand, La, and Erschler [BLE07]. Among more recent methods, Bertsimas and Sim develop a general robust optimization framework for solving discrete optimization and network flow problems in [BS03] and [BS04], which is then applied to robust project scheduling by Lin, Janak, and Floudas [LJF04] and [JLF07].

Approximation methods first formulate an accurate but intractable model using a fairly powerful modeling framework, then find tractable approximations to it. Integer linear approximations to mixed-integer nonlinear programs seem to be particularly useful: Pishevar and Tavakkoi-Moghaddam use this approach to solve a β-robust parallel machine scheduling problem [PT14], while Anglani, Grieco, Guerriero, and Musmanno [Ang+05] use it for the robust parallel machines with sequence-dependent setup costs.

Heuristic methods are quite popular for their ability to quickly find reasonably good solutions, often stemming from a key insight. They come in many forms, though local search techniques are quite common. Lu, Lin, and Ying utilize a simulated annealing heuristic to solve the robust single machine with uncertain processing times and setup times [LLY14], while Singh and Mahapatra utilize quantum particle swarm for flexible job shop scheduling with random machine breakdowns [SM13]. On the other hand, not all heuristic methods fall under the local search paradigm; Yang and Yu consider a heuristic dynamic programming approach for robust single machine with interval uncertainty processing times [YY02].

Our methodology contributes to the literature of exact methods, by applying scenario generation to mitigate the computational difficulty of handling robustness with respect to combinatorial uncertainty sets.

3.2.2 Robust Job-shop Problem

Leon, Wu, and Storer [LWS94] were one of the first to extend the genetic algorithm framework, which had been used to solve non-robust job-shop problems [YN97], to the robust case with machine breakdowns, considering single disruptions and introducing various robustness measures. They then formulate a genetic algorithm which seeks to maximize average slack time, and consider its effectiveness against both machine breakdowns and processing time uncertainty. Not long after, Mehta and Uzsoy [MU98] consider a similar problem with maximum lateness objective, and develop various heuristics involving the insertion of idle time between jobs, given the optimal non-robust schedule.

Following Wu et al., genetic algorithms remain the dominant framework for solving the robust job-shop problem with machine breakdowns. Jensen [Jen03] formulates a genetic algorithm with robustness measures and rescheduling strategies based on the N1 neighborhood (the set of schedules obtainable by a single swap of consecutive jobs from the base schedule). Lei [Lei11] tailors the genetic algorithm for the case where the breakdown sizes and frequency are distributed according to an exponential distribution.

While heuristics based on genetic algorithms are by far the most common, they are not the only solution method that has been investigated. Shafia, Aghaee, and Jamili [SAJ11] provide one of the few MIP models for problems of this type, while a tabu search approach is investigated by Nababan and Salim Sitompul [NS11] and Goren, Sabuncuoglu, and Koc [GSK12].

Several authors have also considered the robust flexible job shop problem with machine breakdowns [AE11] [HS13] [SM13] [XXC13]. Here too, genetic algorithms have been dominant, although some authors have utilized other metaheuristics such as tabu search, simulated annealing, and particle swarm optimization.

Our work differs from the existing literature by introducing a generic CP-based solution method which, unlike most heuristic methods, is able to prove the optimality of a solution while mitigating some of the scaling issues which have troubled prior exact approaches.

3.3 Robust Scheduling

Stated most generally, we consider problems of the form

$$(\mathrm{P}) = \min_{x \in S} \left\{ \max_{q \in \mathcal{Q}} f(x; q) \right\} = \left\{ \begin{array}{ll} \min & v \\ \text{s.t.} & x \in X \\ & v \geq f(x; q) \ \forall q \in \mathcal{Q} \end{array} \right\}$$

where f is an arbitrary objective function, X is a discrete set, and $\mathcal{Q}$ is a set of scenarios. If $\mathcal{Q}$ has a "nice" structure (e.g., $\mathcal{Q}$ is convex, or even better, polyhedral), the above problem may be transfomed into an efficiently solvable form via duality

theory (see [BS04]). However, if $\mathcal{Q}$ is *finite* and *discrete*, no such explicit reformulation may exist. For example, $\mathcal{Q}$ may represent a set of machine breakdowns where the breakdown patterns of different machines are correlated. Nevertheless, even in such cases the set of scenarios $\mathcal{Q}$ often has a combinatorial structure, in the sense that for a fixed $\overline{x} \in X$, we can solve $\max_{q \in \mathcal{Q}} f(\overline{x}; q)$ "reasonably efficiently", even though the problem may be (technically speaking) NP-hard. This is especially the case for robust scheduling problems.

This suggests the use of decomposition methods to take advantage of the structure of $\mathcal{Q}$. In particular, given any subset $Q \subseteq \mathcal{Q}$, the problem

$$(\text{MP}) = \left\{ \begin{array}{ll} \min & v \\ \text{s.t.} & x \in X \\ & v \geq f(x; q) \ \forall q \in Q \end{array} \right\}$$

provides a lower bound to the optimal solution of (P). Furthermore, if we find that the optimal solution x^* to (MP) with objective value v^* satisfies $v^* \geq \max_{q \in \mathcal{Q}} f(x^*; q)$, then x^* actually achieves robustness to the *whole* of $\mathcal{Q}$, not just Q. If we can find a small Q with this property, we can solve the original robust optimization problem much more efficiently. Of course, finding such a Q is difficult in general; however, we can try to systematically construct it via scenario generation.

3.3.1 Scenario Generation

We propose the *scenario generation algorithm* (SGA) as a method of iteratively generating a small number of "representative scenarios" Q that allow us to solve the original robust problem optimally. The idea is simple; start with $Q = \emptyset$ and solve (MP) to obtain an optimal solution x^* with objective value v^*. Then solve the following *scenario generation subproblem*

$$(\text{SP}) = \max\{f(x^*; q) \mid q \in \mathcal{Q}\}$$

and compare its optimal value w^* with v^*. If $v^* = w^*$ then x^* is robust to the original uncertainty set $\mathcal{Q}$ and we are done. Otherwise, we add the optimal solution of (SP) (i.e., the *worst-case scenario* relative to x^*) to Q and re-solve (MP). Since $\mathcal{Q}$ is finite, eventually $Q = \mathcal{Q}$ in which case (MP) $\equiv$ (P). However, the hope is that only a small number of iterations are required in order to achieve robustness to all of $\mathcal{Q}$. Algorithm 1 gives a more detailed description of SGA.

3.3.2 Heuristic Scenario Generation

SGA requires solving the subproblem to optimality every time we want to generate a new scenario. However, this is not strictly necessary for correctness of the algorithm. Indeed, we may improve the overall efficiency of the algorithm if we only search

Algorithm 1 Scenario Generation Algorithm (SGA)

$i \leftarrow 0, Q^0 \leftarrow \emptyset$
while not reached time/iteration limit **do**
  Solve master problem (MP^i) w.r.t. scenarios Q^i
    $\hookrightarrow$ $x^i :=$ optimal solution, $v^i :=$ objective value
  Solve subproblem (SP^i)
    $\hookrightarrow$ $q^i :=$ worst-case scenario w.r.t. x^i
  if $f(x^i, q^i) - v^i = 0$ **then**
    Terminate with optimal solution x^i
  else
    $Q^{i+1} \leftarrow Q^i \cup \{q^i\}$
    $i \leftarrow i + 1$
  end if
end while
Terminated by limit, report best solution found so far

for optimally bad scenarios some of the time, and we are satisfied with only finding "sufficiently bad" scenarios most of the time, especially at the beginning of the search.

This idea yields a *heuristic* scenario generation algorithm (HSGA), which is identical to SGA but with the following alternative scenario generation scheme: Given algorithm parameters $\alpha > 0$ and $\beta > 0$, initialize $\alpha^1 = \alpha$. Then, after we solve the master problem on iteration i yielding solution x^i with objective value v^i, we first attempt to find a scenario that achieves a *minimum degradation factor* of $1 + \alpha^i$, i.e., a scenario q^i such that $\text{Obj}(x^i, q^i) \geq (1 + \alpha^i)v^i$. If we find such a solution, we add it to Q^i, keep $\alpha^{i+1} = \alpha^i$ and re-solve the master problem. If we are unable to find such a solution, we solve the subproblem optimally as before and find the worst-case scenario and add it to Q^i, decrease the minimum degradation to $\alpha^{i+1} = \alpha^i/\beta$, and re-solve the master problem. Algorithm 2 gives a more detailed description of HSGA.

3.4 Example: Robust Job-Shop Scheduling

3.4.1 Problem Statement

To clarify the exposition, we first describe the standard job-shop problem, and then our robust variant. In the standard job-shop problem, we are given a set of n jobs $\mathcal{J}$ and m machines $\mathcal{M}$, where each job j consists of a sequence of operations o_{ij} characterized (1) the machine that processes it (machine i), and (2) its duration d_{ij}. Our goal is to find a schedule (i.e., start times s_{ij} for each operation) with the optimal objective value, where the schedule must satisfy the following constraints:

1. (capacity): each machine can run at most one operation at a time

Algorithm 2 Heuristic SGA (Params: α, β)

$i \leftarrow 0, Q^0 \leftarrow \emptyset, \alpha^1 \leftarrow \alpha$
while not reached time/iteration limit **do**
  Solve master problem (MP^i) w.r.t. scenarios Q^i
    $\hookrightarrow x^i :=$ optimal solution, $v^i :=$ objective value
  Search for scenario $\bar{q}^i$ s.t. Obj$(x^i, \bar{q}^i) \geq (1 + \alpha^i)v^i$
  if $\bar{q}^i$ exists **then**
    $\hookrightarrow q^i := \bar{q}^i$
    $\alpha^{i+1} \leftarrow \alpha^i$
  else no such $\bar{q}^i$ exists
    Solve subproblem (SP^i)
      $\hookrightarrow q^i :=$ worst-case scenario w.r.t. x^i
    $\alpha^{i+1} \leftarrow \alpha^i / \beta$
  end if
  if $f(x^i, q^i) - v^i = 0$ **then**
    Terminate with optimal solution x^i
  else
    $Q^{i+1} \leftarrow Q^i \cup \{q^i\}$
    $i \leftarrow i + 1$
  end if
end while
Terminated by limit, report best solution found so far

2. (precedence): if operation o_{ij} comes before operation $o_{i'j}$ in the sequence associated with job j, then we must finish o_{ij} before we can start $o_{i'j}$; we shall refer to these precedence constraints by the set $\mathcal{P}$

Typically we aim to minimize the maximum completion time over all jobs (also called the makespan, or CMAX) or the sum of weighted completion times (SWCT), where the completion time of a job j is the time at which the last operation of j finishes processing. For simplicity, we assume that each job requires exactly one operation from each of the m machines.

Our problem is a robust variant of this standard job-shop problem. Here, in addition to the standard data we are given a set of possible *processing delays* δ_{ij} for each operation, where $\delta_{ij} \in [0, D_{ij}]$ where D_{ij} is the maximum possible delay of operation o_{ij}. We assume that at most k of the δ_{ij}'s are nonzero, and at most one of the δ_{ij}'s corresponding to operations on the same machine is nonzero. This allows us to represent, for example, machine breakdowns with restart semantics by setting $D_{ij} = D + d_{ij} - 1$, where D is the (fixed, known) duration of a single breakdown.

It is worth elaborating on what is meant by "robustness" here. More specifically, a solution to the robust job-shop problem is a *sequencing* Z of operations on each machine, which implicity defines a set of schedules $\{s(Z, q) : q \in \mathcal{Q}\}$, where $\mathcal{Q}$ is

the set of all feasible delay scenarios and $s(Z, q)$ is the unique *semi-active schedule* that respects the specified sequence Z (i.e., no operation can be started earlier while still satisfying all precedence relations implied by Z), the job precedences $\mathcal{P}$, and the delays in q. This corresponds to a rescheduling policy where each operation is scheduled "as early as possible" given all delayed operations are started immediately after the delay (i.e., AOR rescheduling, [AS97]). Thus Z is declared optimal if: for any other sequence of operations Z', there is a delay scenario $q' \in \mathcal{Q}$ such that $\text{Obj}(s(Z', q')) \geq \max_{q \in \mathcal{Q}} \text{Obj}(s(Z, q))$ where $\text{Obj}(s)$ is the objective value of the schedule s. The solution concept used here is equivalent to that of a *partial order schedule* presented by [Pol+07], interpreting the job-shop problem as a special case of the resource-constrained project scheduling problem.

3.4.2 Special case: Makespan with 1 delay (CMAX1)

If our objective is makespan and we assume $k = 1$ delay, then the problem can be simplified as follows: given any feasible schedule Z, let $\text{slack}_j(Z)$ denote the "slack time" of operation j w.r.t. Z, i.e., the maximum amount of time j can be delayed without increasing the makespan of Z. Then delaying operation j by δ_j increases the makespan by exactly $\max\{0, \delta_j - \text{slack}_j(Z)\}$. Since operation j can be delayed by at most $D + d_j - 1$ (by assumption), the worst case delay for Z is given by $\delta_{j^*} = D + d_{j^*} - 1$, where j^* maximizes $d_j - \text{slack}_j(Z)$. Consequently, the problem can be solved by adding a term v to the objective function, together with constraints $v \geq D + d_j - 1 - \text{slack}_j$ for every operation j and constraints calculating the value of slack_j for every operation j.

Note that in case of $k \geq 2$ delays, or when our objective is sum of weighted completion times, this no longer applies since the effect (on the objective value) of delaying operation j can depend on how the delay "propagates" to other operations and/or how the operations are ordered on each machine.

3.4.3 Model

We use constraint programming models to represent our problems. In particular, we make use of the interval variables, sequence variables, and scheduling constraints provided IBM CP Optimizer ([Lab09]). We give a brief formal definition of these constructs defined by IBM CP Optimizer, and then describe a CP model for the robust job-shop problem that uses these constructs.

Intuitively, an *interval variable* represents an object that takes up some interval of time. Formally, we define the set *CPOInterval* as

$$\text{CPOInterval} = \{(\ell, u) \in \mathbb{Z}^2 \mid 0 \leq \ell \leq u\}$$

in which case an interval variable I is a variable that takes values in the set CPOInterval. We write **startOf**(I) and **endOf**(I) to refer to the first and second components of I respectively; in addition, we define **lengthOf**$(I) := \text{endOf}(I) - \text{startOf}(I)$.

An interval variable may also be *optional*, which intuitively means that it may or may not be instantiated in the solution. Formally, we define the set *OptionalCPOInterval* := CPOInterval$\cup\{\emptyset\}$, where $\emptyset$ is an "empty interval" with lengthOf($\emptyset$) = 0. We then define the expression **presenceOf**(I) to be 0 is $I = \emptyset$ and 1 otherwise.

Intuitively, a *sequence variable* over an *ordered* set (i.e., an array) of interval variables $\mathcal{I}$ represents a total ordering of the intervals in $\mathcal{I}$. Formally, we define the set *CPOSequence($\mathcal{I}$)* by

$$\text{CPOSequence}(\mathcal{I}) = \{\sigma : \mathcal{I} \to \{1, \ldots, |\mathcal{I}|\} \mid \sigma \text{ is bijective}\}$$

in which case a sequence variable $S = S(\mathcal{I})$ is a variable that takes values in the set CPOSequence($\mathcal{I}$). We write σ_S to refer to the bijective mapping on $\mathcal{I}$ specified by S.

We also define the following constraints:

1. Let I, I' be intervals. The constraint **endBeforeStart**(I, I') requires that endOf(I) $\leq$ startOf(I')

2. Let $\mathcal{I}$ be an array of intervals and $S = S(\mathcal{I})$ a sequence variable over $\mathcal{I}$. The constraint **noOverlap**(S) requires that (i) the intervals ordered by S are non-overlapping and (ii) the ordering imposed by S is consistent with the temporal ordering of $\mathcal{I}$. That is, for every pair of intervals $I, I' \in \mathcal{I}$, (i) either endOf(I) $\leq$ startOf(I') or endOf(I') $\leq$ startOf(I), and (ii) endOf(I) $\leq$ startOf(I') if and only if $\sigma_S(I) < \sigma_S(I')$

3. Let $\mathcal{I}$ and $\mathcal{I}'$ be arrays of intervals of the same cardinality k, and let $S = S(\mathcal{I})$ and $S' = S'(\mathcal{I}')$ be sequence variables over $\mathcal{I}$ and $\mathcal{I}'$ respectively. The constraint **sameSequence**(S, S') requires that the ordering imposed by S and S' be the same modulo the "trivial" mapping between $\mathcal{I} = \{I_1, \ldots, I_k\}$ and $\mathcal{I}' = \{I'_1, \ldots, I'_k\}$, i.e., for any pair (i, j) with $1 \leq i < j \leq k$, $\sigma_S(I_i) < \sigma_S(I_j)$ if and only if $\sigma_{S'}(I'_i) < \sigma_{S'}(I'_j)$

We now describe the CP model for the robust-job-shop problem. Intuitively, our model defines a separate standard job-shop model for each scenario, then links the solutions of different scenarios together with the sameSequence constraint.

Let $\mathcal{Q}$ denote the set of possible delay scenarios. For each delay scenario $q \in \mathcal{Q}$, let o_{ijq} be the interval variable representing the operation that job j runs on machine i realized in scenario q, and let S_{iq} be the sequence variable corresponding to machine i and scenario q. Let C_{jq} represent the completion of time job j, i.e., the time at which the last operation of job j completes its processing, in scenario Q. For a machine i', let $\mathcal{O}_q(i') = \{o_{ijq} \mid i = i'\}$ be the set of operations that run on machine i' in scenario q. Let $\mathcal{P}_q$ denote the precedence constraints on operations imposed by the job sequences, applied to the operations of scenario q. Then the following is our model for the robust

job-shop problem:

$$
\begin{aligned}
\min \quad & v && && (3.1)\\
\text{s.t.} \quad & v \geq \text{Obj}(C_{1q}, \ldots, C_{nq}) && \forall q \in \mathcal{Q} && (3.2)\\
& C_{jq} \geq \text{endOf}(o_{ijq}) && \forall i \in \mathcal{M}, \forall j \in \mathcal{J}, \forall q \in \mathcal{Q} && (3.3)\\
& \text{sameSequence}(S_{i1}, S_{iq}) && \forall i \in \mathcal{M}, q \in \mathcal{Q} && (3.4)\\
& \text{noOverlap}(S_{iq}) && \forall i \in \mathcal{M}, \forall q \in \mathcal{Q} && (3.5)\\
& \text{endBeforeStart}(o, o') && \forall (o, o') \in \mathcal{P}_q, \forall q \in \mathcal{Q} && (3.6)\\
& \text{lengthOf}(o_{ijq}) = d_{ij} && \forall i \in \mathcal{M}, \forall j \in \mathcal{J}, \forall q \in \mathcal{Q} && (3.7)\\
& C_{jq} \geq 0, C_{jq} \in \mathbb{Z} && \forall j \in \mathcal{J}, \forall q \in \mathcal{Q} && (3.8)\\
& o_{ijq} \in \text{CPOInterval} && \forall i \in \mathcal{M},\ \forall j \in \mathcal{J}, \forall q \in \mathcal{Q} && (3.9)\\
& S_{iq} \in \text{CPOSequence}(\mathcal{O}_q(i)) && \forall i \in \mathcal{M}, \forall q \in \mathcal{Q} && (3.10)
\end{aligned}
$$

Constraints (3.1) and (3.2) express our objective: minimizing the worst case objective value over all breakdown scenarios in $\mathcal{Q}$, while (3.4) ensures we utilize the same sequence of operations for all scenarios. The constraints (3.5) and (3.6) impose constraints associated with the standard job-shop problem, except each operation o only interacts with other operations within the same scenario, and the sequencing of the operations are communicated through S_{iq}.

3.4.4 Scenario Generation

Since the size of the model given in the last section (and in particular, the number of scenarios) grows with both the number of machines m and the number of jobs n, the number of required scenarios can quickly become impractically large. The idea is therefore to generate a subset of the scenarios representing the "worst cases" the solution must consider. The hope is that we only need to generate a small number of scenarios to arrive at an optimal solution to the original problem. This approach may be viewed as an instance of logic-based Benders [HO03b] (in fact, in the context of minmax regret, it is equivalent to the generation of "regret cuts", first introduced in [ML98]).

In particular, our master problem

$$
\begin{aligned}
(\mathrm{MP}^i) = \min \quad & v & \\
\text{s.t.} \quad & v \geq \mathrm{Obj}(C_{1q}, \ldots, C_{nq}) & \forall q \in Q^i \\
& C_{jq} \geq \mathrm{endOf}(o_{ijq}) & \forall i \in \mathcal{M}, \forall j \in \mathcal{J}, \forall q \in Q^i \\
& \mathrm{sameSequence}(S_{i1}, S_{iq}) & \forall i \in \mathcal{M}, q \in Q^i \\
& \mathrm{noOverlap}(S_{iq}) & \forall i \in \mathcal{M}, \forall q \in Q^i \\
& \mathrm{endBeforeStart}(o, o') & \forall (o, o') \in \mathcal{P}_q, \forall q \in Q^i \\
& \mathrm{lengthOf}(o_{ijq}) = d_{ij} & \forall i \in \mathcal{M}, \forall j \in \mathcal{J}, \forall q \in Q^i \\
& C_{jq} \geq 0, C_{jq} \in \mathbb{Z} & \forall j \in \mathcal{J}, \forall q \in Q^i \\
& o_{ijq} \in \mathrm{CPOInterval} & \forall i \in \mathcal{M},\ \forall j \in \mathcal{J}, \forall q \in Q^i \\
& S_{iq} \in \mathrm{CPOSequence}(\mathcal{O}_q(i)) & \forall i \in \mathcal{M}, \forall q \in Q^i
\end{aligned}
$$

at iteration i is a relaxation of the full master problem where we only include a subset $Q^i \subseteq \mathcal{Q}$ of all possible delay scenarios. Thus, (MP^i) finds the schedule with the lowest worst-case objective, where we only consider the scenarios in Q^i. Given a solution Z^i from (MP^i), our subproblem

$$
\begin{aligned}
(\mathrm{SP}^i) = \max \quad & \mathrm{Obj}(C_1, \ldots, C_n) & \\
\text{s.t.} \quad & C_j \geq \mathrm{endOf}(o_{ij}) & \forall i \in \mathcal{M}, \forall j \in \mathcal{J} \\
& \mathrm{noOverlap}(S_i) & \forall i \in \mathcal{M} \\
& \mathrm{endOf}(\delta_{i\sigma_{Z^i}(\ell)}) = \mathrm{startOf}(o_{i\sigma_{Z^i}(\ell+1)}) & \forall \ell = 1, \ldots, |\mathcal{J}| - 1 \\
& \mathrm{endOf}(o_{ij}) = \mathrm{startOf}(\delta_{ij}) & \forall i \in \mathcal{M}, \forall j \in \mathcal{J} \\
& \sum_{i \in \mathcal{M}, j \in \mathcal{J}} \mathrm{presenceOf}(\delta_{ij}) \leq k & \\
& \mathrm{lengthOf}(o_{ij}) = d_{ij} & \forall i \in \mathcal{M}, \forall j \in \mathcal{J} \\
& \mathrm{lengthOf}(\delta_{ij}) = D_{ij} & \forall i \in \mathcal{M}, \forall j \in \mathcal{J} \\
& C_j \geq 0, C_j \in \mathbb{Z} & \forall j \in \mathcal{J} \\
& o_{ij} \in \mathrm{CPOInterval} & \forall i \in \mathcal{M}, \forall j \in \mathcal{J} \\
& \delta_{ij} \in \mathrm{OptionalCPOInterval} & \forall i \in \mathcal{M}, \forall j \in \mathcal{J} \\
& S_i \in \mathrm{CPOSequence}(\mathcal{O}'(i)) & \forall in \mathcal{M}
\end{aligned}
$$

then finds the worst-case scenario $q^i \in \mathcal{Q}$ w.r.t. Z^i, i.e., the scenario that degrades the objective value of Z^i by the maximum possible amount. This worst-case scenario q^i is then added to Q^i to form the subset for the next iteration Q^{i+1}.

The algorithm terminates when we either reach a prespecified time or iteration limit, or we find the worst-case breakdown q^i causes no degradation of the objective value. In the latter case, we may conclude the current solution is optimal. (See algorithm 1.)

Notice that in the statement of our algorithm, we only assume that (a) the set $\mathcal{Q}$ can be characterized as a set of "scenarios" and (b) there is some method that can (efficiently) find the "worst" scenario in $\mathcal{Q}$ relative to some feasible solution to the master problem. Thus, although the uncertainty set for our particular problem is rather simple, the algorithm is equipped to handle much more general uncertainty sets. Thus our analysis will be focused less on computational efficiency (which cannot be expected considering its generality) and more on the behavior of its upper/lower bounds as the number of iterations increases.

3.4.5 Upper/Lower Bounds and Termination

The lower bounds at each iteration increase monotonically, since they are solutions to increasingly stronger relaxations of the original problem. However, the upper bounds at each iteration fluctuate non-monotonically, since they are derived from the worst-case breakdown scenario specific to the solution found at that iteration, and thus do not apply to solutions found at any other iteration. Nevertheless, since at each iteration we obtain a feasible solution and its corresponding (worst-case) objective value, taking the minimum over all solutions found so far gives a valid upper bound.

Note that if we continue to add new delays from $\mathcal{Q}$ to Q^i (if we do not, then by the definition of Z^i and Q^i we are done) and set no time/iteration limits, eventually $Q^i = \mathcal{Q}$ and the master problem will become equivalent to the original problem, and the algorithm will terminate. Of course, the hope is that the algorithm will terminate much sooner, since $|\mathcal{Q}|$ is often quite large, and the complexity of the master problem increases exponentially with the number of scenarios.

Finally, note that solving (SP^i) to optimality is not strictly required for the SG algorithm; it is still valid even if we only require that each q^i be a feasible solution to (SP^i), since $Q^i \subseteq \mathcal{Q}$ is still true at each iteration. We therefore introduce the following heuristic: in iteration 0, we solve (SP^0) to optimality; in iteration 1, we initialize a "minimum degradation level" $\alpha^1 \geq 0$; then when solving (SP^i) for $i \geq 1$, if we encounter a scenario q such that $\text{Obj}(\vec{s}(Z^i, q)) \geq (1 + \alpha^i) \cdot v^i$, we stop solving ($\text{SP}^i$), return q as our solution, and set $\alpha^{i+1} \leftarrow \alpha^i$; if we prove that no such scenario q exists, we return q that maximizes $\text{Obj}(\vec{s}(Z^i, q))$, and set $\alpha^{i+1} \leftarrow \alpha^i/\beta$, where $\beta > 1$ is some reducing factor. The effect of this heuristic and parameters α^1 and β are investigated in our experiments.

3.5 Experimental Results for Robust Job-Shop

Our experiments serve three purposes. First, we wish to analyze the behavior of our scenario-generation algorithm in terms of 1) number of iterations and bound convergence, 2) sensitivity with respect to the objective and the number of breakdowns, and 3) comparison with the non-robust optimal solutions in terms of worst-case and best-case behavior. For this we use a large set of randomly generated instances each

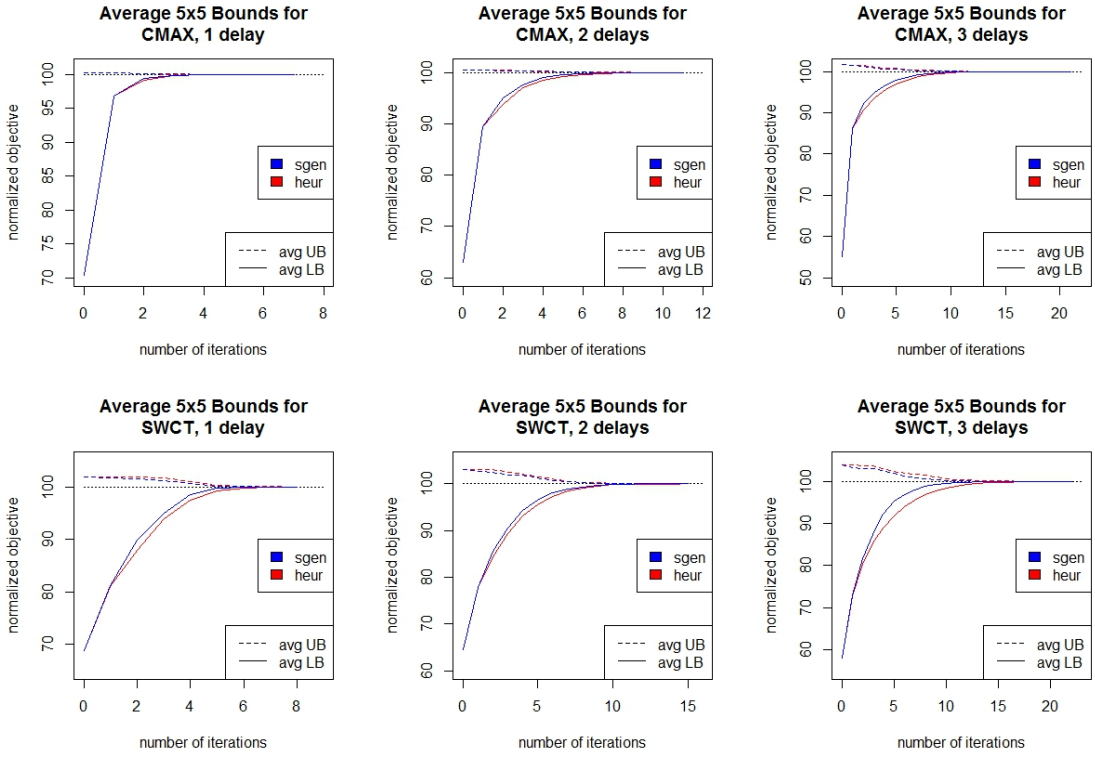


Figure 3.1: Average Upper Bounds (dashed line) and Lower Bounds (solid line) for CMAX/SWCT objectives with 3 delays, normalized to the optimal robust objective value (dotted line); sgen (blue) corresponds to SG without the heuristic, and heur (red) corresponds to SG with the heuristic (with $\alpha^1 = 0.2, \beta = 2$)

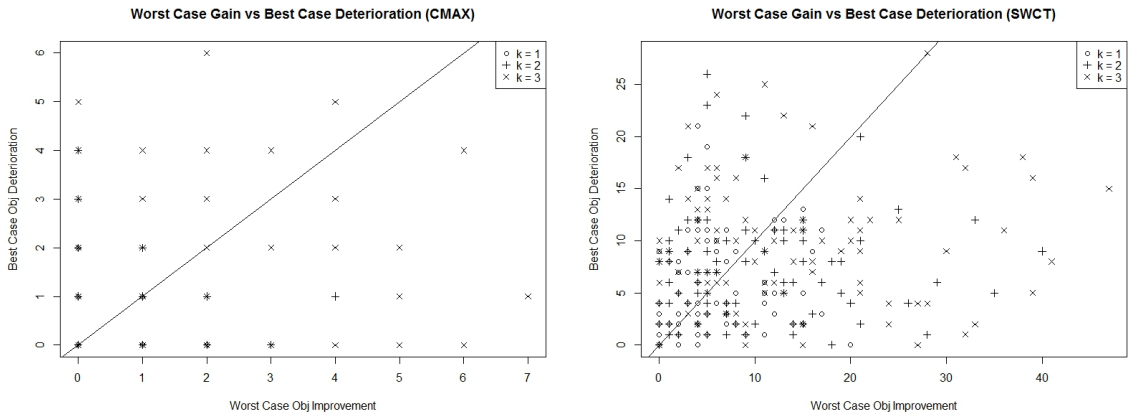
of which can be solved relatively quickly. Second, we evaluate our approach on the Lawrence test suite that consist of larger instances, to assess the current scalability of our method. Third, we run the heuristic with different parameter values for α^1 and β to see its effect on the solving time and number of iterations required.

3.5.1 Effect of Multiple Delays/Objectives on Schedule Robustness

We first test our methodology on 100 randomly generated 5x5 instances (5 jobs, 5 machines), where the durations of operations were drawn from a uniform distribution on $\{0, 1, \ldots, 10\}$. We considered $k = 1, 2, 3$ possible delays with a duration of $D = 5$, with both makespan (CMAX) and sum of weighted completion times (SWCT) objectives. For these experiments, we set $\alpha^1 = 0.2$ and $\beta = 2$ for our heuristic parameters. The impact of different values for α is β will be discussed in a later section.

Figure 3.1 shows the average bounding behavior over the 100 instances, with the

blue/red lines corresponding to the scenario generation (SG) algorithm without/with the heuristic[1] respectively. We see that on average, obtaining bounds for the CMAX objective is easier than obtaining bounds for the SWCT objective, and both become more difficult to compute as the number of delays k increases. Furthermore, the bounds obtained by the algorithm with the heuristic are very similar to those obtained by the algorithm without it. Thus, only partially solving the subproblem on each iteration can be an effective way to reduce the computation time while still obtaining reasonable bounds on the objective.



a. Worst-case gain vs. best-case deterioriation for CMAX

b. Worst-case gain vs. best-case deterioriation for SWCT

Figure 3.2: Comparing Robust and Non-robust Optimal Solutions with respect to Worst-case gain versus Best-case deterioration for CMAX (a), resp. SWCT (b). For each figure we report instances with $k = 1, 2, 3$ breakdowns.

Figure 3.2.a and 3.2.b show the relationship between $\overline{z_{det}}$, the worst case objective value for the optimal non-robust (i.e., deterministic) solution (calculated with the non-robust CP model) and $\overline{z_{rob}}$, the worst case objective value for the optimal robust solution (calculated with our methodology). Here each point represents a single instance; the points below the diagonal are those instances where $\overline{z_{rob}} < \overline{z_{det}}$. Figure 3.2.c and 3.2.d compare z_{det} and z_{rob}, the best case objective value (i.e., when there are no breakdowns) for the non-robust and robust optimal solutions respectively. Figure 3.2.e and 3.2.f compare the worst-case gain ($\overline{z_{det}} - \overline{z_{rob}}$) with the best-case deterioration ($z_{rob} - z_{det}$) of the objective. Note that $z_{det} \leq z_{rob} \leq \overline{z_{rob}} \leq \overline{z_{det}}$.

The most interesting experiment involves the gain that a robust solution can provide, relative to its cost. For each instance, we compare two solutions: One is

[1]Note that SG with the heuristic is still an exact procedure, since it solves every (restricted) master problem to optimality; however, since it does not solve every subproblem to optimality, when calculating upper bounds, we must only take the minimum over solutions whose subproblem was solved to optimality.

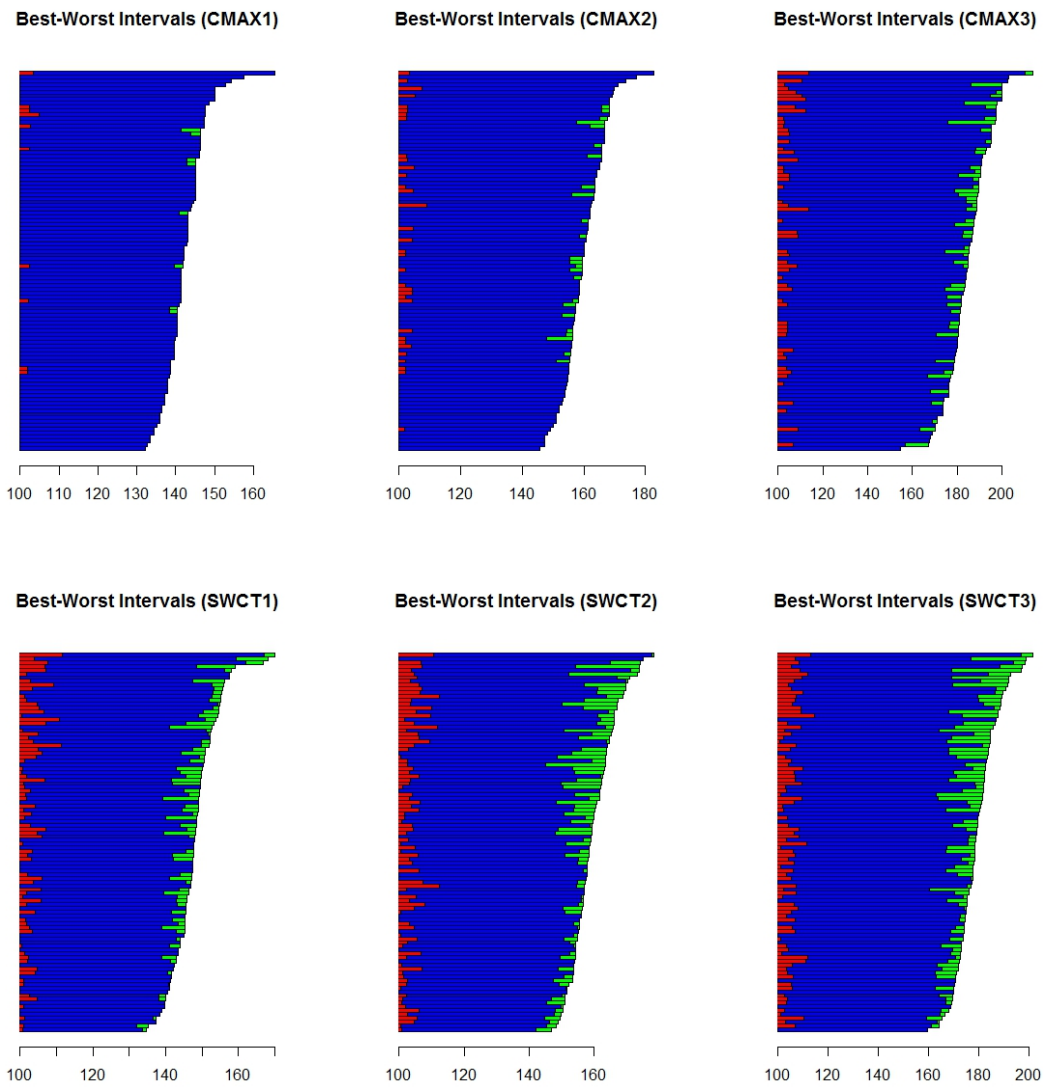


Figure 3.3: Best-worst graphs for all instances. CMAXk, resp. SWCTk, represents the CMAX, resp. SWCT, objective with k delays, where $k = 1, 2, 3$. For each instance the depicted interval represents the difference between the best-case and worst-case objective. The red interval on the left represents the best-case deterioration while the green interval on the right represents the worst-case gain of the robust solution w.r.t. the non-robust solution.

provided by the conventional optimal deterministic CP model (with objective z_{det}), while the other is provided by our robust CP model (with objective z_{rob}). We evaluate the maximum gain that the robust solution can provide in a worst-case scenario (i.e., $z_{\text{det}} - z_{\text{rob}}$) versus the maximum deterioration in the best-case (no breakdowns)

scenario (i.e., $z_{\text{rob}} - z_{\text{det}}$). The scatter plots in Figure 3.2.a and 3.2.b show this comparison for each instance (for $k = 1, 2, 3$), with the CMAX and SWCT objectives. The points below the diagonal are those instances where the worst-case gain is higher than the best-case deterioration. For example, for some instances with the SWCT objective, the robust solution may provide a gain up to value 40 while the deterioration may only be of value 5. The opposite is also possible, as indicated in the figure.

Figure 3.2 demonstrates that 1) the SWCT objective is much more sensitive to uncertainty than the CMAX objective, and 2) there is a clear trade-off between the worst-case protection and best-case deterioration offered by robust solutions. To make the latter trade-off more explicit, Figure 3.3 represents each instance as an interval, where the whole interval indicates the gap between z_{det} and $\overline{z_{det}}$, while the blue region indicates the gap between z_{rob} and $\overline{z_{rob}}$. Note all values are normalized to the deterministic objective value. We denote by CMAX1, CMAX2, and CMAX3 the problem with CMAX objective and $k = 1, 2, 3$ breakdowns, respectively; similarly for SWCT1, SWCT2, and SWCT3. We observe that for CMAX, the robust and non-robust solutions have similar best case/worst case behavior, suggesting that the simplicity of the makespan objective "washes out" the robustness. By contrast, for SWCT there is more variation between the robust and non-robust solutions. Naturally, for some instances the worst-case gain is lower than the best-case deterioration. However, the figure also demonstrates that *the robust solution may provide much better worst-case behavior with limited best-case objective deterioration.*

3.5.2 Larger Instances

We next evaluate our methodology on the Lawrence test suite [Law84], a standard set of JSP problem instances provided by the OR-library. We remark that despite their relatively small size, finding exact robust solutions for these instances is very challenging. For these tests, we used a delay duration of $D = 10$, again with CMAX and SWCT objectives. In addition, we set a time limit of 1 hour to solve each problem specification.

The results for CMAX objective are shown in Table 3.1; the three columns show how many instances were solved to optimality, obtained bounds, or were unable to obtain any bounds (due to not being able to complete the first iteration within the time limit). We see that the problem becomes intractable even with 20 jobs on 10 machines when the number of breakdowns increases. With the SWCT objective, we were not able to solve any of the instances within the specified time limit, and we could only compute trivial bounds (from the deterministic optimum) for instances la01–05 and la16–la20.

instances	size	CMAX, $k=1$			CMAX, $k=2$			CMAX, $k=3$		
		S	B	NB	S	B	NB	S	B	NB
la01-05	10x5	5	0	0	4	1	0	1	4	0
la06-10	15x5	5	0	0	4	1	0	0	5	0
la11-15	20x5	5	0	0	4	1	0	0	5	0
la16-20	10x10	5	0	0	0	5	0	0	5	0
la21-25	15x10	2	3	0	0	5	0	0	0	5
la26-30	20x10	3	2	0	0	3	2	0	0	5
la31-35	30x10	5	0	0	0	0	5	0	0	5
la36-40	15x15	1	4	0	0	5	0	0	0	5

Table 3.1: Results with CMAX Objective, and $k = 1, 2, 3$ breakdowns.
S = solved to optimality, B = obtained bounds, NB = no bounds obtained

3.5.3 Heuristic Parameters

We also explored the effect of changing the parameters of our heuristic. We tested $\alpha^1 = 0.01, 0.125, 0.25, 0.5, 1.0$ for our initial minimum degradation level and $\beta = 1.25, 1.5, 2.0, 4.0, 8.0$ for the factor we divide by whenever we solve the subproblem to optimality. For each pair of parameters (α^1, β), we solved the first 10 of the 5x5 randomly generated instances from our first experiment using either CMAX or SWCT objectives and $k = 3$ breakdowns.

Table 3.2 contains the results, showing the average solving time and average number of iterations. We see that larger values of α^1 and smaller values of β both tend to decrease the number of iterations, up to a certain point. On the other hand, the solving time can either increase or decrease depending on the type of objective, though it appears to be affected more by α^1 than by β. In particular, CMAX benefits from running many iterations with α^1 close to 0, while SWCT does best when $\alpha^1 \approx 0.5$ and $\beta \approx 3$.

3.6 Example: Robust Unrelated Parallel Machine Scheduling

3.6.1 Problem Statement

Similarly to the job shop, let us first describe the standard unrelated parallel machine scheduling problem, and then our robust variant. In the standard problem, we are given a set of n jobs $\mathcal{J}$ and m machines $\mathcal{M}$, where for each machine i and job j we have a *duration* d_{ij} indicating how long it takes to run job j on machine i. Each job also has a *release time* r_j indicating that no machine can start working on job j until after time r_j. Our goal is to find a schedule (i.e., determine J_m, the set of jobs to run on machine m, and s_j, the start time of job j) with the optimal objective value,

Table 3.2: Effect of Heuristic Parameters (over 10 instances)

CMAX, $k = 3$		Avg Solve Time					Avg Num Iterations				
		Reducing Factor (β)					Reducing Factor (β)				
		1.25	1.5	2.0	4.0	8.0	1.25	1.5	2.0	4.0	8.0
Initial	0.01	35.59	35.60	35.58	35.61	35.61	10.5	10.5	10.5	10.5	10.5
De-	0.125	70.64	64.60	58.47	53.00	51.90	6.9	6.9	7.0	7.7	8.5
grd	0.25	86.82	76.39	68.70	60.56	52.56	5.9	5.9	6.1	6.8	6.7
Level	0.5	108.41	94.79	78.08	64.64	54.92	6.3	6.3	5.6	6.2	6.9
(α^1)	1.0	113.53	109.79	89.81	73.28	63.46	6.4	6.4	6.3	6.4	7.7

SWCT, $k = 3$		Avg Solve Time					Avg Num Iterations				
		Reducing Factor (β)					Reducing Factor (β)				
		1.25	1.5	2.0	4.0	8.0	1.25	1.5	2.0	4.0	8.0
Initial	0.01	835.60	836.20	835.29	835.86	835.89	19.5	19.5	19.5	19.5	19.5
De-	0.125	406.35	396.05	381.72	419.18	447.28	13.4	13.4	13.8	13.8	14.0
grd	0.25	338.50	320.35	304.04	277.10	347.97	11.9	11.7	12.1	12.4	13.3
Level	0.5	274.52	266.10	256.23	293.88	357.45	9.9	9.9	10.8	12.3	13.0
(α^1)	1.0	289.95	281.84	256.60	246.55	353.03	9.8	9.8	10.8	11.4	13.3

where the schedule satisfies the constraint that each machine can run at most one job at a time and all jobs are started at or after their release time. Again, we consider both CMAX (the maximum completion time over all jobs) and SWCT (the sum of weighted complection times) as useful metrics.

Our problem is a robust version of this standard unrelated parallel machine scheduling problem. Again, for the robust problem we are also given a set of possible *processing delays* $\delta_{ij} \in [0, D_{ij}]$, where D_{ij} is the maximum possible delay of job j on machine i. We assume that at most k of the δ_{ij}'s are nonzero , and at most k of the δ_{ij}'s corresponding to the same machine is nonzero. We use the same notion of robustness as with the robust job shop problem, except a solution Z specifies the assignment of jobs to machines, and potentially sequencing if the objective is SWCT.

3.6.2 Model

As before, we use constraint programming to model our problem.

Let $\mathcal{Q}$ denote the set of possible delay scenarios. For each delay scenario $q \in \mathcal{Q}$, let o_{ijq} be the interval variable representing running job j on machine i in scenario q, and let S_{iq} be the sequence variable corresponding to machine i and scenario q. Let C_{jq} represent the completion of time job j, i.e., the time at which job j completes its processing, in scenario Q. For a machine i', let $\mathcal{O}_q(i') = \{o_{ijq} \mid i = i'\}$ be the set of operations that run on machine i' in scenario q. Then the following is our model for

the robust unrelated parallel machine scheduling problem:

$$
\begin{aligned}
\min \quad & v && && (3.11)\\
\text{s.t.} \quad & v \geq \text{Obj}(C_{1q}, \ldots, C_{nq}) && \forall q \in \mathcal{Q} && (3.12)\\
& C_{jq} \geq \text{endOf}(o_{ijq}) && \forall i \in \mathcal{M}, \forall j \in \mathcal{J}, \forall q \in \mathcal{Q} && (3.13)\\
& \text{sameSequence}(S_{i1}, S_{iq}) && \forall i \in \mathcal{M}, q \in \mathcal{Q} && (3.14)\\
& \text{noOverlap}(S_{iq}) && \forall i \in \mathcal{M}, \forall q \in \mathcal{Q} && (3.15)\\
& \sum_{i \in \mathcal{M}} \text{presenceOf}(o_{ijq}) = 1, && \forall j \in \mathcal{J}, \forall q \in \mathcal{Q} && (3.16)\\
& \text{lengthOf}(o_{ijq}) = d_{ij} && \forall i \in \mathcal{M}, \forall j \in \mathcal{J}, \forall q \in \mathcal{Q} && (3.17)\\
& \text{startOf}(o_{ijq}) \geq r_j && \forall i \in \mathcal{M}, \forall j \in \mathcal{J}, \forall q \in \mathcal{Q} && (3.18)\\
& C_{jq} \geq 0, C_{jq} \in \mathbb{Z} && \forall j \in \mathcal{J}, \forall q \in \mathcal{Q} && (3.19)\\
& o_{ijq} \in \text{OptionalCPOInterval} && \forall i \in \mathcal{M},\ \forall j \in \mathcal{J}, \forall q \in \mathcal{Q} && (3.20)\\
& S_{iq} \in \text{CPOSequence}(\mathcal{O}_q(i)) && \forall i \in \mathcal{M}, \forall q \in \mathcal{Q} && (3.21)
\end{aligned}
$$

The model is very similar to the one for robust-job shop. Constraints (3.11) and (3.12) express our objective: minimizing the worst case objective value over all breakdown scenarios in $\mathcal{Q}$, while (3.14) ensures we utilize the same assignment (and possibly sequence) of jobs for all scenarios. The constraints (3.15) and (3.16) impose constraints associated with the standard unrelated parallel machine scheduling problem, except each job o only interacts with other jobs within the same scenario, and the sequencing of the operations are communicated through S_{iq}. Note that each interval o_{ijq} is now an *optional* interval since (3.16) forces the job to be assigned to exactly one machine.

3.6.3 Scenario Generation

Following the same methodology as before, our solution algorithm has exactly the same structure as with robust job-shop, except the master problem is now

$$
\begin{aligned}
(\mathrm{MP}^i) = \min \quad & v & \\
\text{s.t.} \quad & v \geq \mathrm{Obj}(C_{1q}, \ldots, C_{nq}) & \forall q \in \mathcal{Q} \\
& C_{jq} \geq \mathrm{endOf}(o_{ijq}) & \forall i \in \mathcal{M}, \forall j \in \mathcal{J}, \forall q \in Q^i \\
& \mathrm{sameSequence}(S_{i1}, S_{iq}) & \forall i \in \mathcal{M}, q \in Q^i \\
& \mathrm{noOverlap}(S_{iq}) & \forall i \in \mathcal{M}, \forall q \in Q^i \\
& \sum_{i \in \mathcal{M}} \mathrm{presenceOf}(o_{ijq}) = 1 & \forall j \in \mathcal{J}, \forall q \in Q^i \\
& \mathrm{lengthOf}(o_{ijq}) = d_{ij} & \forall i \in \mathcal{M}, \forall j \in \mathcal{J}, \forall q \in Q^i \\
& \mathrm{startOf}(o_{ijq}) \geq r_j & \forall i \in \mathcal{M}, \forall j \in \mathcal{J}, \forall q \in Q^i \\
& C_{jq} \geq 0, C_{jq} \in \mathbb{Z} & \forall j \in \mathcal{J}, \forall q \in Q^i \\
& o_{ijq} \in \mathrm{OptionalCPOInterval} & \forall i \in \mathcal{M},\ \forall j \in \mathcal{J}, \forall q \in Q^i \\
& S_{iq} \in \mathrm{CPOSequence}(\mathcal{O}_q(i)) & \forall i \in \mathcal{M}, \forall q \in Q^i
\end{aligned}
$$

and our subproblem is

$$
\begin{aligned}
(\mathrm{SP}^i) = \max \quad & \mathrm{Obj}(C_1, \ldots, C_n) & \\
\text{s.t.} \quad & C_j \geq \mathrm{endOf}(o_{ij}) & \forall i \in \mathcal{M}, \forall j \in \mathcal{J} \\
& \mathrm{noOverlap}(S_i) & \forall i \in \mathcal{M} \\
& \mathrm{presenceOf}(o_{ij}) = \mathcal{A}_{Z^i}(o_{ij}) & \forall i \in \mathcal{M}, \forall j \in \mathcal{J} \\
& \mathrm{endOf}(\delta_{i\sigma_{Z^i}(\ell)}) = \mathrm{startOf}(o_{i\sigma_{Z^i}(\ell+1)}) & \forall \ell = 1, \ldots, |\mathcal{J}| - 1 \\
& \mathrm{endOf}(o_{ij}) = \mathrm{startOf}(\delta_{ij}) & \forall i \in \mathcal{M}, \forall j \in \mathcal{J} \\
& \mathrm{presenceOf}(\delta_{ij}) \leq \mathrm{presenceOf}(o_{ij}) & \forall i \in \mathcal{M}, \forall j \in \mathcal{J} \\
& \sum_{i \in \mathcal{M}, j \in \mathcal{J}} \mathrm{presenceOf}(\delta_{ij}) \leq k & \\
& \mathrm{lengthOf}(o_{ij}) = d_{ij} & \forall i \in \mathcal{M}, \forall j \in \mathcal{J} \\
& \mathrm{lengthOf}(\delta_{ij}) = D_{ij} & \forall i \in \mathcal{M}, \forall j \in \mathcal{J} \\
& C_j \geq 0, C_j \in \mathbb{Z} & \forall j \in \mathcal{J} \\
& o_{ij} \in \mathrm{CPOInterval} & \forall i \in \mathcal{M}, \forall j \in \mathcal{J} \\
& \delta_{ij} \in \mathrm{OptionalCPOInterval} & \forall i \in \mathcal{M}, \forall j \in \mathcal{J} \\
& S_i \in \mathrm{CPOSequence}(\mathcal{O}'(i)) & \forall i \in \mathcal{M}
\end{aligned}
$$

3.6.4 Computational Results

While we do not have computational results for this model at this time, preliminary testing indicates similar trends as with robust-job shop.

3.7 Conclusion and Discussion

We introduced an exact solution method, based on scenario generation, for constraint-based scheduling problems with combinatorial uncertainty sets. As specific case studies, we considered the robust job-shop scheduling problem and robust unrelated parallel machine scheduling problem with machine breakdowns. Our method provides provable bounds on the worst-case objective value, and our experimental analysis suggests that nontrivial bounds can be obtained even after a single iteration. Naturally, robust solutions provide protection against unfavorable scenarios, at the potential cost of objective efficiency when no such scenario occurs. Thus, if proving some performance guarantee of the schedule is of paramount importance, our method can be used to produce such robust solutions.

Even if we only need a small subset of scenarios to find and prove optimal robust solutions in many cases, the computation time may become prohibitively large when the number of scenarios increases. Our robust CP method would benefit from more effective bounding procedures which extend beyond single iterations, both for providing theoretical guarantees and practical power for iterative procedures. While a generic scalable approach to robust scheduling based on constraint programming will likely require more sophisticated tools than the ones we have presented here, the fact that such a generic approach to robust scheduling is even possible highlights the benefits of using constraint programming as a promising framework to develop such an approach.

Chapter 4

Post-Optimality Analysis of Mixed Integer Linear Programming Problems Using Decision Diagrams

4.1 Introduction

Mathematical optimization is used in a wide range of fields to make decisions, allocate resources, and maximize performance for various problems. Depending on the types of decisions made, the constraints of the problem and the objective function in consideration, several different classes of optimization models may be used. Integer linear programing (ILP) and mixed integer linear programming (MILP) models are especially popular for their ability to express discrete decisions and/or logical constraints. The rich mathematical structure arising from the interaction between integrality and linear inequality constraints provides fertile ground for deep insights, enabling theoretical advances that have translated to high quality solvers that can find an optimal solution to most moderately sized models within a few hours.

However, in practice a single optimal solution to a MILP model may not be sufficient for resolving the original problem. Often the model does not exactly represent reality due to other considerations and restrictions which are not (and often cannot) be incorporated into the model, or parameters whose exact values cannot be determined. Even if the model is an exact representation of reality, a single solution by itself yields very little insight into what factors critically influence the final decision, which may be just as important. What is needed is a methodology to gain more insight into the nature of the optimal soltuion beyond being able to generate one. We refer to this as *post-optimality analysis*.

To solve this problem, [HS17] suggest using decision diagrams to compactly encode multiple optimal and near-optimal solutions. By considering multiple solutions that are just as good or almost as good as an optimal one, this gives decision makers a chance to incorporate considerations which could not be represented by the model.

Furthermore, decision diagrams allow one to effficiently make certain queries about the set of solutions it represents. Finally, decision diagrams provide a compact way to encode a large number of solutions; in fact, [HS17] consider a special kind of diagram called a *sound decision diagram* to gain even greater compression.

In this work, we extend the framework of [HS17] from ILP models to MILP models, which are more common in practice. In the process of extending the notion of sound reduction to the MILP case, we also propose two broad approaches for representing solutions with both discrete and continuous parts within a decision diagram, an inherently discrete object. Finally, we perform computational testing on the most scalable approach, and show that a sound decision diagram can be compiled with a reasonable amount of extra work in comparison to generating the solutions using a MIP solver.

4.2 Related Work

Prior work that is related to our problem can be split into three categories: post-optimality for linear programming (LP) problems, and post-optimality for integer linear programming (ILP) and mixed integer linear programming (MILP) problems, and miscellaneous.

4.2.1 Postoptimality for LP

To our knowledge, no previous study specifically addresses the question of how to represent the set of near-optimal solutions of an LP problem in a compact and transparent fashion. There is one study [Lee+00] that gives a sequential algorithm for generating all optimal solutions of an LP problem in the context of metabolic engineering. However, they do not consider solution representation and only perform computational experiments for a single model.

That said, the set of near-optimal solutions of an LP problem is always a (convex) polyhedron, so questions of representation and solution generation can be essentially reduced to corresponding questions about polyhedra. Here, the long line of research on *vertex enumeration* becomes relevant.

For general polyhedra, there are two major approaches: *incremental* and *graph-traversal*. The classical *double description method* [Mot+53; Che65; FP95] is a prime example of the incremental approach; it iteratively builds a *double description pair* (i.e., a pair of matrices (A, R) such that $Ax = 0 \iff \exists \lambda \geq 0$ s.t. $x = R\lambda$) of the polyhedron by considering the inequalities that describe the polyhedron one by one. By contrast, the *reverse search method* [Bal61; Dye83; AF92; Avi00] is a graph-traversal approach; it starts with a feasible vertex and performs pivoting operations to explore neighboring vertices, much like the simplex algorithm. A theoretical and computational comparison by [ABS97] of the two approaches shows that incremental methods are often immune to degeneracy but scale poorly, whereas graph-traversal methods

tend to scale well but have trouble with degeneracy. Alternative approaches based on backtracking [FLM97; BL98] as well as the the dual problem of *facet enumeration* [Swa85; FR94; Bar+96; BFM98; Jos03] have also been studied.

Interestingly, the complexity of vertex enumeration for general polytopes is still an open problem, while for general polyhedra [Kha+08] proves that it is NP-hard. Vertex enumeration for 0/1-polytopes can be done in polynomial time and space [BL98] (in fact, it is strongly $\mathcal{P}$-*enumerable* [FLM97]), while for 0/1-polyhedra no output-polynomial time algorithm exists unless $P = NP$ [Bor+11]. As always, there exist many special cases that can be done in polynomial time [Pro94; ADP03; Bor+09; Mur09]

We consider the question of representing the set of near-optimal solutions to LP problems. In particular, we propose representing basic feasible solutions based on which inequalities are part of the basis, and incorporate the cost of these basic solutions via a weighted decision diagram.

4.2.2 Postoptimality for ILP and MILP

To our knowledge, no previous study (other than direct predecessors of this work) addresses the issue of how to represent near-optimal solutions of ILP problems in a compact and transparent fashion.

A few papers have proposed methods for generating a diverse set of multiple solutions. [GLW00] utilizes scatter search to generate near-optimal solutions for 0–1 MILPs that roughly maximizes Hamming distance among different solutions. However, since it is a heuristic method, it does not obtain an exhaustive set of solutions for any given optimality tolerance. Diverse solutions of a MILP problem have also been obtained by solving a sequence of MILP models, beginning with the given problem, in which each seeks a solution different from the previous ones. [Gre+08] explicitly compares this approach to a much larger model that obtains multiple solutions simultaneously. However, neither method is scalable, as there may be a large number of near-optimal solutions.

Other papers focus on ways to utilize the branch-and-bound tree to efficiently generate multiple optimal solutions. The *one-tree method* of [Dan+07] generates a collection of optimal or near-optimal solutions of a MILP problem by extending a branching tree that is used to solve the problem. While possible, the collection is not intended to be exhausive, and there is no indication of how to represent the collection compactly or query more easily. [AHK08] presents an extension of branch-and-bound called *branch-and-count* to enumerate all *feasible* solutions of an ILP problem, based on the identification of *unrestricted subtrees* of the branching tree. These are subtrees in which all values of the unfixed variables are feasible. While the method can be extended to MILP, the paper only tests pure integer programs, and does not explicitly take the objective value into account, only feasibility.

The one-tree method is used by CPLEX [IBM19] as part of its "solution pool"

feature which was introduced in CPLEX 11.0 [10]. Branch-and-count is used by SCIP [Gle+18] for solution counting, which was introduced for pure ILPs in SCIP 1.1 [08] and for mixed ILPs in SCIP 2.0 [11]. By contrast, post-optimality software based on decision diagrams operates independently of the solution method. It also differs by organizing an exhaustive set of near-optimal solutions in a decision diagram that is convenient for post-optimality analysis, as opposed to heuristically considering quality and/or diversity.

Integer programming sensitivity analysis has been investigated for some time, as for example in [Bow72; GN77; Wol81; HK84; SW85; DH00]; the mixed integer programming case has been studied in [Roo74; Wil89; Cre95; Pac04; GR07]. However, our main interest here is in probing the near-optimal solution set that results from the *original* problem data, rather than analyzing parameter sensitivity by investigating *perturbations* of the problem data.

Decision diagrams were first proposed for ILP post-optimality analysis in [HH06], and the concept of a sound diagram was introduced in [HH07]. [HS17] proves several properties of sound diagrams, introduces the sound reduction operation, and proves that sound reduction yields a sound diagram of minimum size. It also presents algorithms for generating sound-reduced diagrams for ILP problems and conducting post-optimality analysis on these diagrams, as well as reporting computational tests on the representational efficiency of the diagrams. Binary decision diagrams have also been used in [BE07] to implement a fast vertex enumeration method for the special case of 0-1 ILP problems.

The present work extends [HS17] by allowing variables to be continuous. We propose several ways to construct sound diagrams for MILP problems, and show that a modified version of sound reduction can still be done. We also implement the most promising approach and report results on the representational efficiency of the diagrams.

4.2.3 Miscellaneous

Our problem has some similarities with that of *product configuration*, which considers the process of specifying a product consisting of a set of components, where different components can only be combined in certain predefined ways. In particular, in both cases we want a systematic method to analyze the set of near-optimal/valid solutions to some fixed optimization/configuration model.

One approach used in product configuration is to represent the set of valid configurations in a form that allows various queries to be performed very quickly. The process of computing such a representation is known as *knowledge compilation*. While initial papers use deterministic finite automata (DFA) [AFM02] and binary decision diagrams (BDD) [Had+04; AHP10] as their target representation, many others including AND/OR multivalued decision diagrams (AND/OR MDD) [MMD07], Tree-of-BDDs (ToB) [Sub05], and boolean satisfiability (SAT) instances [SKK03; Jan10]

have also been considered. However, all of them are restricted to configuration models with only discrete variables. While there are a few papers that consider product configuration models with continuous variables [Ald+03; GF03; XHK05], none of them consider the task of knowledge compilation.

A completely different line of work initiated by [Hoe+99] considers the use of decision diagrams in the context of stochastic dynamic programming. Here the value function of a markov decision process (MDP) is represented as an *algebraic decision diagram* (ADD) [Bah+97], a generalization of a BDD that is designed to represent a pseudo-boolean function. By using an ADD that compactly captures both the recursive and disjunctive structure of the value function, value iteration can be performed very efficiently. [SDB11] extends this to continuous state MDPs by generalizing ADDs to *extended algebraic decision diagrams* (XADD) where each node represents a polynomial inequality/equality/disequality, and its two outgoing arcs corresponding to whether the condition is satisfied or not. [ZSF12] further extend this approach to MDPs with continuous state and action spaces by restricting the conditions to non-negated linear inequalities and possibly negated boolean variables. However, the ADD/XADD approach does not readily extend to our setting since most MILPs do not have a compact recursive formulation.

Most recently, in the relatively new area of decision diagram methods for optimization, [Lin17] proposes an extension of MDD-based branch-and-bound [Ber13] that incorporates Benders cuts into the compilation of relaxed and restricted diagrams. This allows the method to handle MILP problems by treating the continuous part as the Benders subproblem. In this case, feasibility cuts pose no issues, but objective cuts require the use of *cost tuples*, which associates a tuple to every node of the MDD to track the intermediate righthand side value of all objective cuts. In particular, the cost tuples are incompatible with reduced diagrams since the righthand side values arising from different prefixes may not be identical. Nevertheless, cost tuples can still be used during top-down compilation to eliminate suboptimal solutions, and extensive computational results are reported for variations of the maximum independent set problem and the adapted market share split problem.

4.3 Decision Diagrams

A *decision diagram* D is a directed multigraph that represents an indicator function $h(x_1, \ldots, x_n)$ in which each x_j has a finite domain S_j. The node set of D is partitioned into subsets or *layers* $U_1, \ldots, U_{n+1}$, with U_1 containing only the *root node* r. Every arc a of D is directed from a node $u \in U_j$ to a node in layer U_{j+1}. Each arc a leaving $u \in U_j$ has a *label* $x_j(a) \in S_i$ that represents the assignment of value $x_j(a)$ to variable x_j. The arcs leaving u must have distinct labels. If at most two arcs leave each node, D is a *Binary Decision Diagram* (BDD), and otherwise it is a *Multivalued Decision Diagram* (MDD).

Decision diagrams classically have two terminal nodes in layer U_{n+1}, representing

x_1	x_2	x_3	$h(x)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

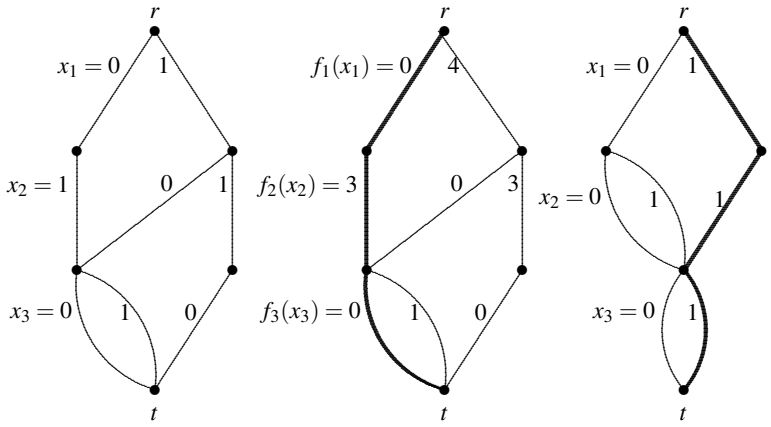


Figure 4.1: (a) Decision diagram representing the Boolean function on the left. Arc labels are shown. (b) Decision diagram representing problem (4.1). Arc weights are shown. The heavy path represents the optimal solution. (c) Sound relaxation of the same decision diagram. The heavy path corresponds to an additional path with weight exceeding those of paths from the other diagram.

true and false outcomes for the indicator function. For our purposes, it suffices to have a single terminal node t for true. A path p in D from r to t represents an assignment of values to $x = (x_1, \ldots, x_n)$, which we will denote by $x(p)$. The diagram represents function h if its r–t paths represent precisely the values of x for which $h(x) = 1$. For example, the binary decision diagram of Fig. 4.1(a) represents the Boolean function h shown.

Decision diagrams can represent the solutions of any optimization problem with finite-domain variables and a separable objective function; that is, any problem of the form

$$\min \left\{ \sum_{j=1}^{n} f_j(x_j) \;\middle|\; x \in S \right\} \tag{P}$$

where $S \subseteq S_1 \times \cdots \times S_n$ and S_j is finite for all j. If we define $h(x)$ to be 1 if and only if $x \in S$, then decision diagram D *represents* problem P when D represents $h(x)$, and each arc a leaving a node in layer U_j is given weight $f_j(x_j(a))$. For example, the diagram of Fig. 4.1(b) represents the optimization problem

$$\min \left\{ 4x_1 + 3x_2 + x_3 \;\middle|\; 2x_1 + 2x_2 + x3 \geq 2, \;\; x_1 + x_2 + x_3 \leq 2, \;\; x \in \{0,1\}^3 \right\} \tag{4.1}$$

The weight of any r–t path p in D is the cost $\sum_i f_j(x_j(p))$ of the corresponding solution $x(p)$. Furthermore, any shortest r–t path p defines an optimal solution $x(p)$ of P. This is illustrated by the heavy arcs in Fig. 4.1(b).

It is convenient to let $\mathrm{Sol}(D)$ be the set of solutions represented by diagram D. We also define for each node $u \in U_j$ the set $\mathrm{Pre}_D(u)$ of *prefixes* of u, which are the

assignments to $(x_1, \ldots, x_{j-1})$ that correspond to r-u paths in D. We similarly define $\text{Suf}_D(u)$ to be the set of *suffixes* of u, or assignments to $(x_j, \ldots, x_n)$ corresponding to u-t paths.

Many different decision diagrams can represent a given feasible set S. Yet for a given variable ordering, there is a unique *reduced* diagram that represents P [Bry86]. A diagram D is reduced if no two nodes in a layer have the same suffixes. That is, for every layer U_j and every pair of distinct nodes $u, v \in U_j$, $\text{Suf}_D(u) \neq \text{Suf}_D(v)$. For example, the diagram of Fig. 4.1(a) is reduced for the variable ordering x_1, x_2, x_3. The reduced diagram can be viewed as a compact representation of the complete branching tree for P.

The following sequence of results is proven in [HS17].

Lemma 1. *Given distinct nodes $u, v \in U_j$ in a decision diagram D, we have* $\text{Pre}_D(u) \cap \text{Pre}_D(v) = \emptyset$.

Lemma 2. *Let diagram D have layers U_j and diagram D' have layers U'_j for $j = 1, \ldots, n+1$, and suppose* $\text{Sol}(D) = \text{Sol}(D')$. *Then for any $u \in U_j$, there is a node $v \in U'_j$ for which* $\text{Suf}_D(u) = \text{Suf}_{D'}(v)$.

Corollary 1. *For a given variable ordering, the reduced diagram D representing P is the unique diagram upon isomorphism representing P with the minimum number of nodes as well as with the minimum number of arcs.*

Our primary interest is in representing near-optimal solutions of P. Let $P(\Delta)$ be the set of feasible solutions with cost within Δ of the optimal cost z^*, which we refer to as Δ-*optimal* solutions. Thus

$$\text{P}(\Delta) = \left\{ x \in S \;\middle|\; \sum_j f_j(x_j) \leq z^* + \Delta \right\}$$

A diagram D *exactly represents* $\text{P}(\Delta)$ if its r–t paths correspond exactly to the Δ-optimal solutions of P; that is, $\text{Sol}(D) = \text{P}(\Delta)$.

4.4 Sound Decision Diagrams for ILP

A decision diagram is sometimes smaller when it contains more paths, which implies that certain relaxations of the solution set can be encoded more concisely. As defined by [HH07], sound decision diagrams take advantage of this by allowing paths that correspond to "costly" solutions (i.e., solutions that are worse than Δ-optimal) to be included in order to obtain a more compact diagram.

More formally, we say that a solution is Δ-*costly* if its cost is greater than $z^* + \Delta$, and Δ-*cheap* otherwise. A Δ-optimal solution is then a solution that is both Δ-cheap and feasible; conversely, a Δ-*fake* solution is a solution that is both Δ-cheap and infeasible. Given these definitions, we say that a diagram D is *sound* for $\text{P}(\Delta)$ if

1. every Δ-optimal solution of P is represented by an r–t path of D, and

2. no r–t path of D represents a Δ-fake solution; i.e., every r–t path of D either represents a Δ-optimal solution or a Δ-costly solution.

A *proper* sound diagram contains at least one path that is not Δ-optimal.

Fig. 4.1(c) shows how a sound relaxation can be smaller than an exact diagram. All r–t paths in Fig. 4.1(b) weight from 3 to 7, whereas the heavy path in Fig. 4.1(c) weights 8.

One can easily check if a prefix is relevant in a sound relaxation. This is done by pre-computing the shortest path from each node to the terminal and using that value to check if some path from r can be extended to a near-optimal solution. In the example above, consider $z^* = 3$ and $\Delta = 4$. Since the first two arcs in the heavy path weight 7, only the arc for $x_3 = 0$ can be used next.

The following basic properties of sound diagrams were first stated by [HH07]:

Lemma 3. *If decision diagram D is sound for* $\mathrm{P}(\Delta)$ *and* $0 \leq \Delta' \leq \Delta$*, then D is sound for* $\mathrm{P}(\Delta')$*.*

Lemma 4. *When $\Delta \to \infty$, the only sound diagrams for* $\mathrm{P}(\Delta)$ *are those that exactly represent P.*

Since we are interested in small sound diagrams, we wish to remove any arcs and nodes that are not needed to represent Δ-optimal solutions. We therefore focus on *minimal* sound diagrams for $\mathrm{P}(\Delta)$, which are those in which every node lies on some r–t path with weight at most $z^* + \Delta$, and similarly for every arc. Minimality is a necessary condition for a sound diagram to have minimum size.

Interestingly, soundness is not useful when one is only concerned with optimal solutions. In that case, introducing suboptimal paths into the diagram cannot result in a smaller sound diagram. This is a consequence of the following [HS17], which plays a central role in what comes after:

Theorem 1. *No proper sound diagram for* $\mathrm{P}(0)$ *is minimal.*

Proof. Suppose to the contrary that diagram D is a minimal sound diagram for $\mathrm{P}(0)$ and contains a suboptimal r–t path p. For any given node u in p, let $\pi(u)$ be the portion of p from r to u and $\sigma(u)$ the portion from u to t (Fig. 4.2). Select the node u^* in p that maximizes the number of arcs in $\pi(u^*)$ subject to the condition that $\pi(u^*)$ is part of some shortest r–t path in D. Let p^* be such a shortest path, where p^* consists of $\pi(u^*)$ and σ^*. We note that $u^* \notin U_{n+1}$, since otherwise p would be a shortest r–t path, and thus u^* is succeeded in path p by node u' through arc a. Furthermore, $u^* \notin U_1$ since otherwise arc a would prevent D from being minimal. Now since D is minimal, arc a belongs to some shortest r–t path p', which we may suppose consists of π', a, and σ'. Since both p^* and p' are shortest r–t paths, $\pi(u^*)$ and π' must be shortest r-u^* paths, which means that the path consisting of $\pi(u^*)$, a, and σ' is also

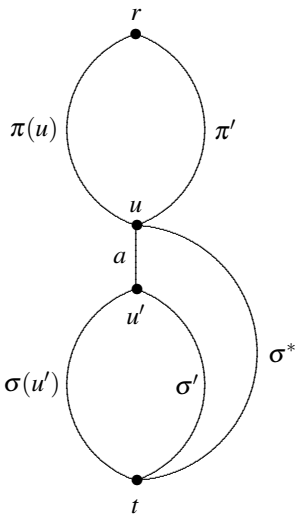


Figure 4.2: Illustration of the proof of Theorem 1.

a shortest r–t path. But this implies that $\pi(u') = \pi(u^*)\ a$, which contains one more arc than $\pi(u^*)$, is part of a shortest r–t path, contrary to the definition of u^*. □

Corollary 2. *Given any sound diagram D for $P(0)$, some diagram at least as small as D exactly represents $P(0)$.*

Proof. If D is improper, it already represents $\mathrm{P}(0)$ exactly. Otherwise, D is not minimal, and at least one arc or node can be removed to yield a sound diagram for $\mathrm{P}(0)$. The process can be repeated until the resulting sound diagram is improper and therefore exactly represents $\mathrm{P}(0)$. □

Theorem 1 is also a key to understand when soundness makes diagrams smaller. The contradiction in the proof would fail if $\Delta > 0$ and the weight of r–u path $\pi(u)$ exceeded that of π'. In that case, there would be suffixes of u that work with π' but are too expensive when combined with $\pi(u)$. Hence, we denote as *sound reduction* a generalization of the reduction operation by which we remove a node and redirect its incoming arcs to some other node while preserving soundness. Due to the following results from [HS17], we know that it suffices to find sound relaxations of minimum size. These results use the concept of *sufficient suffix set*, which for a node u in a diagram D consists of the subset of suffixes $\overline{\mathrm{Suf}}_D(u) \subseteq \mathrm{Suf}_D(u)$ which result in a near-optimal solution when combined with some prefix in $\mathrm{Pre}_D(u)$.

Theorem 2. *In a minimal sound diagram D for $\mathrm{P}(\Delta)$, a node $u \in U_i$ can be sound reduced into another node $v \in U_i$ if and only if*

(i) $\overline{\mathrm{Suf}}_D(u) \subseteq \mathrm{Suf}_D(v)$

(ii) $\sum_{j=1}^{i} f_j(p_j) + \sum_{j=i+1}^{n} f_j(s_{j-i}) > z^* + \Delta \quad \forall p \in \mathrm{Pre}_D(u),\ s \in \mathrm{Suf}_D(v) \setminus \mathrm{Suf}_D(u)$

Theorem 3. *In a minimal sound diagram D for $\mathrm{P}(\Delta)$ with a node $u \in U_i$ that cannot be sound-reduced into the other nodes of U_i, any sound diagram D' for $\mathrm{P}(\Delta)$ has a node $u' \in U'_i$ such that $\overline{\mathrm{Suf}}_D(u) = \overline{\mathrm{Suf}}_{D'}(q)$.*

Corollary 3. *A sound diagram D for $\mathrm{P}(\Delta)$ has minimum number of nodes and arcs if and only if D is minimal and sound reduction cannot be further applied.*

4.5 Representing Continuous Variables

We now consider the question of how to construct sound decision diagrams for MILP problems. With little loss of generality, we consider MILP problems in which the integer variables are bounded:

$$\min \left\{ cx + dy \;\middle|\; Ax + By \geq b,\ x \in S,\ y \geq 0 \right\} \tag{M}$$

Here $S \subseteq S_1 \times \cdots \times S_n$ and each S_j has the form $\{x_j \in \mathbb{Z} \mid \alpha_j \leq x_j \leq \beta_j\}$. We assume that (M) has an optimal solution x^* with a finite optimal value z^*; this is reasonable because post-optimality analysis, by definition, is not applicable unless this is the case.

A major challenge of our problem is deciding how to modify the representation scheme of decision diagrams to accommodate continuous variables. Broadly speaking, we would like the representation to accomplish three things:

1. The set of possible assignments to the continuous variables must be "discretized" in some way, since a decision diagram cannot natively handle variables with infinite domains.

2. The representation should try to maximize the number of nodes that have equivalent suffixes, since such nodes can be *merged* when constructing the reduced diagram; otherwise the diagram will provide little benefit over a straightforward branching tree representation.

3. The weight of a path p in the decision diagram that corresponds to a solution x should ideally be equal to the cost of x. If not, the weight should at least provide a lower bound on the cost if x is feasible, and an upper bound if x is infeasible. Otherwise, there is no way to define a notion of reduction that preserves soundness, since the diagram cannot reliably distinguish Δ-optimal solutions from Δ-costly solutions based on the cost of their corresponding paths.

Additionally, it is worth noting that a MILP problem may have infinitely many optimal solutions, and almost always has infinitely many near-optimal solutions. In particular, if there are two optimal solutions which only differ in the continuous variables, then any convex combination of the two is also optimal. Thus, the set of

near-optimal solutions of a MILP is the union of finitely many polyhedra. This is in contrast to ILP problems whose set of near-optimal solutions is always a finite set. Thus, for a MILP problem it may be impossible to exhaustively enumerate the set of optimal solutions even in principle.

Bearing these considerations in mind, we consider two major approaches for the representation scheme: *explicit representation*, where arcs corresponding to continuous variables appear in the diagram, and *implicit representation*, where the diagram only contains arcs corresponding to discrete variables. We will primarily focus on the implicit representation approach, due to several advantages over the explicit representation approach in our setting.

4.6 Explicit Representation

The explicit representation approach incorporates some discretized representation of the set of possible assignments to the continuous variables into the decision diagram. Since the set of near-optimal solutions of a MILP is a union of finitely many polyhedra, it cannot be exhaustively enumerated in general. However, standard LP theory shows that when optimizing linear functions over (a union of) polyhedra, it suffices to only consider the extreme points and extreme rays. Thus, for the remainder of this section we restrict our attention to the *extreme points* of the set of near-optimal sotutions as our representation target.

One approach to evaluating decision diagram representations is to consider how the labeling of the arcs is determined. Recall that for discrete variables, each variable x_j is associated with the set of outgoing arcs from a layer of nodes U_j, and the label $\ell_j(a)$ of an arc a leaving a node $u \in U_j$ represents assignment of value $\ell_j(a)$ to variable x_j. To extend this to continuous variables, we could try something as follows:

1. One option is to use the same labeling system as for discrete variables. That is, each continuous variable y_j is associated with the set of outgoing arcs from a layer of nodes U_j, and the label $\ell_j(a)$ of an arc a leaving a node $u \in U_j$ represents assignment of value $\ell_j(a)$ to variable y_j.

   The advantage of this method is that the arcs corresponding to continuous variables have exactly the same interpretation as they do for discrete variables, making it easy to extend results from the ILP case. However, it suffers from two major issues. First, it is difficult to determine apriori the set of possible labels (i.e., the domain of the variable); too few values and the solution lacks precision, too many value and the diagram suffers from the curse of dimensionality. Further, even if we are able to find a suitable set of labels, the diagram is very unlikely to allow for node merging, since most solutions will take different values in the continuous variables, corresponding to arcs with different labels (and therefore inequivalent suffixes).

2. Another option is to use an interval instead of a single number as the label. That is, each continuous variable y_j is associated with the set of ougoing arcs of a layer of nodes U_j, and the label $\ell_j(a) = [\alpha, \beta]$ of an arc leaving a node $u \in U_j$ represents assignment of some value in the interval $[\alpha, \beta]$ to variable y_j.

   This method allows a single path in the diagram to encompass multiple solutions, potentially enabling nontrivial node merging. Further, while an r–t path in the diagram does not immediately yield an exact value of the continuous variable, the exact values may be obtained by solving a linear program. On the other hand, we still have the problem of determining appropriate endpoints for each interval label. In addition, the interval representation naturally imposes a "rectangular" structure on the set of represented solutions, which may not provide a good approximation of the set of near-optimal solutions which may be the finite union of arbitrary polyhedra.

The problem with both of these methods is that each continuous variable is treated *separately*, whereas the variables defining the set of near-optimal solutions typically interact linearly (i.e., they lie in some affine subspace). Thus, we propose a method that discretizes the continuous part of the problem *collectively* by enumerating the extreme points according to their corresponding "bases". In particular, recall that for LP problems extreme points of the feasible region correspond to *basic feasible solutions*, which satisfy sufficiently many linearly independent constraints with equality. This concept can be extended to MILP problems to yield a discretization of the feasible region, where the values of all continuous variables in the model are simultaneously determined by the basis matrix of the continuous part of the problem.

Formally, recall our MILP problem of the form

$$\min \left\{ cx + dy : Ax + By \geq b, x_j \in S_j \ \forall j, y \geq 0 \right\} \tag{M}$$

where $x \in \mathbb{Z}^{n_1}$, $y \in \mathbb{R}^{n_2}$, and each S_j has the form $\{x_j \in \mathbb{Z} \mid \alpha_j \leq x_j \leq \beta_j\}$. Let

$$F(x) := \{y : By \geq b - Ax, y \geq 0\} = \{y : B'y \geq b' - A'x\} \tag{4.2}$$

be the set of feasible $y \in \mathbb{R}^{n_2}$ corresponding to a feasible $x \in \mathbb{Z}^{n_1}$, where

$$B' = \begin{pmatrix} B \\ I_{n_2 \times n_2} \end{pmatrix}, \quad b' = \begin{pmatrix} b \\ 0_{n_2 \times 1} \end{pmatrix}, \quad A' = \begin{pmatrix} A \\ 0_{n_2 \times n_1} \end{pmatrix}$$

(note B' is a $(m + n_2) \times n_2$ matrix, b' is a $(m + n_2)$-dimensional vector, and A' is a $(m + n_2) \times n_1$ matrix). We call $(\bar{x}, \bar{y})$ a *basic feasible solution (BFS)* of (M) if

1. $\bar{x}_j \in S_j$ for all j
2. $\bar{y} \in F(\bar{x})$, and
3. $\bar{y}$ satisfies n_2 linearly independent constraints in $F(\bar{x})$ with equality

Analogously to a BFS in linear programming, a BFS of (M) is associated with a *basis matrix* B'_{bas}, which is the non-singular $n_2 \times n_2$ submatrix of B' formed by selecting the n_2 linearly independent rows specified by the above definition. $\bar{y}$ then satisfies $B'_{\text{bas}}\bar{y} = b'_{\text{bas}} - A'_{\text{bas}}\bar{x}$ (where b'_{bas} and A'_{bas} are the corresponding subvector/submatrix of b' and A'), and therefore $\bar{y} = (B'_{\text{bas}})^{-1}(b'_{\text{bas}} - A'_{\text{bas}}\bar{x})$. Hence, we can write the cost of the BFS $(\bar{x}, \bar{y})$ as

$$c\bar{x} + d\bar{y} = c\bar{x} + d(B'_{\text{bas}})^{-1}(b'_{\text{bas}} - A'_{\text{bas}}\bar{x}) = [c - d(B'_{\text{bas}})^{-1}A'_{\text{bas}}]\bar{x} + d(B'_{\text{bas}})^{-1}b'_{\text{bas}} \quad (4.3)$$

where the last expression is affine in $\bar{x}$.

To represent a BFS as a path in a decision diagram, we use n_1 arc layers to represent what value is assigned to each discrete variable and n_2 arc layers to represent which rows of B' are chosen to form the basis matrix B'_{bas}. In particular, we define n_2 new variables $z_1, \ldots, z_{n_2}$, each with domain $\{1, \ldots, m+n_2\}$, where the assignment $z_k = v$ corresponds to setting row k of B'_{bas} to be equal to row v of B'. As before, each variable z_k corresponds to the outgoing arcs from a layer of nodes U_k, and the label $\ell_k(a)$ of an arc leaving a node $u \in U_k$ represents assignment of value k to variable z_k.

Alternatively, we could define $m+n_2$ new variables $\hat{z}_1, \ldots, \hat{z}_{m+n_2}$, each with domain $\{0, 1\}$, where $z_v = 1$ if row v of B' is selected to be part of the basis and 0 otherwise. However, this formulation creates an arc layer in the diagram for every row of B', which may be inefficient if B' has a large number of redundant rows.

In principle, the *discrete layers* (i.e., layers of the diagram corresponding to an assignment of the discrete variables) could either come before or after the *continuous layers* (i.e., layers of the diagram corresponding to a choice of basis matrix). To compare the two options, consider a branching tree representation of Feas(M) using our scheme, where the n_1 discrete layers appear *before* the n_2 continuous layers, and consider a node $u \in U_{n_1+1}$ (i.e., after we have traversed all of the discrete layers). Then the subtree rooted at u represents the *polyhedron* $F(x_u) = \{y : By \geq b - Ax_u, y \geq 0\}$ where x_u is the unique prefix of u in the tree, so merging nodes in the subsequent continuous layers requires reasoning about polyhedra and their corresponding basis matrices. On the other hand, if the n_1 discrete layers appear *after* the n_2 continuous layers, the the subtree rooted at a node $u \in U_{n_2+1}$ represents the *discrete set* $G(y_u) := \{x : Ax \geq b - By_u, x \in S\}$, where y_u is the unique prefix of u in the tree. Thus, merging nodes in the subsequent layers can be done according to standard MDD reduction techniques, and it is for this simplicity that we choose to put the continuous layers before the discrete layers.

We are now ready to describe how to construct an exact MDD representation of the feasible region of (M). Recall that in the continuous layers, each arc represents a selection of one of the $m + n_2$ rows of B' to be part of the basis matrix B'_{bas}. To avoid symmetry, we require the index of the selected row to increase as we traverse the layers; this can be enforced by maintaining the index of the last chosen row at every node, and only allowing outgoing arcs to have labels larger than this index. In addition, we exclude any choices that result in linear dependence among the selected

constraints; this can be enforced by forming the partial basis matrix and computing its rank at every node. Arcs in the continuous layer all have 0 cost except for the last layer, where the arc has cost $d(B'_{\text{bas}})^{-1}b'_{\text{bas}}$ where B'_{bas} is the basis resulting from the selection specified by the arcs.In the subsequent discrete layers, each arc still represents setting its corresponding discrete variable to a certain value, but now the cost of setting the variable to value v is modified from $c_j v$ to $(c - d(B'_{\text{bas}})^{-1}A'_{[bas]})_j v$. By (4.3), this ensures that the weight of a root-terminal path in the MDD is equal to the cost of the corresponding BFS.

This approach has the advantage of allowing the weight of a path in the MDD to be exactly equal to the cost of the corresponding solution. Furthermore, the discretization of the continuous variables only depends on the size of the constraint matrices rather than their entries. However, there are still a few drawbacks:

1. Node merging is still difficult because arcs with the same label will typically have different weights, preventing them from being merged.

2. The diagram can quickly become very large for practical instances, since the diagram has layers for both continuous and discrete variables.

Finally, for many problems, analyzing the exact values of the continuous variables may not be particularly relevant. For example, for parallel machine scheduling, we may care about what machine each job is assigned to, but not about when each individual job starts running, since this is fixed once the assignmentsare known. In fact, most MILP solvers (CPLEX, SCIP, Gurobi) only generate multiple solutions with respect to the discrete variables, i.e., two solutions which have the same value in the discrete variables but are different in the continuous variables are considered to be equivalent. Therefore, for the remainder of this work we focus our attention on the other approach: implicit representation.

4.7 Implicit Representation

The implicit representation approach is to interpret the diagram as a projection on the integer variables, in resemblance to what happens at a branching tree. Hence, we will conduct search and post-optimality analysis only with respect to the integer variables $x = (x_1, \ldots, x_n)$. We therefore suppose that for any given value of x, the continuous variables y take optimal values. It is convenient to let $z^*(\bar{x})$ be the optimal value of the linear programming problem that results when x is fixed to $\bar{x}$ in (M). Thus

$$z^*(\bar{x}) = \min \left\{ c\bar{x} + dy \;\middle|\; By \geq b - A\bar{x},\ y \geq 0 \right\} \tag{4.4}$$

A solution $x = \bar{x}$ belongs to the projection when 4.4 is a feasible problem.

We say that an r–t path in decision diagram D *represents* a point in the x-projection of (M) when $x \in \text{Sol}(D)$. Diagram D *represents* the x-projection of (M)

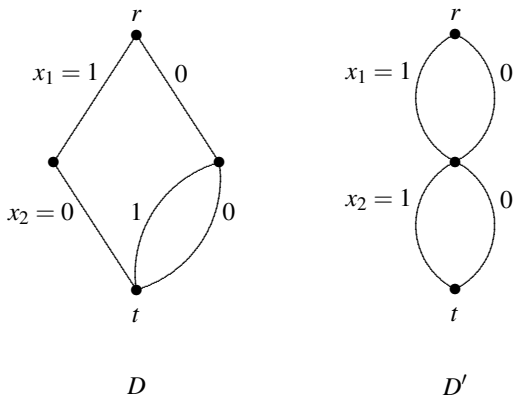


Figure 4.3: Counterexample to the extension of Theorem 1 to MILP problems. Arc labels are shown.

when (a) Sol(D) contains all and only feasible points in the x-projection of (M), and (b) each arc a leaving a node in U_j has weight $c_j x_j(a)$. While the weight of an r–t path p is $cx(p)$, we define the *cost* of p to be the optimal value $z^*(x(p))$ of (M) when x is fixed to $x(p)$.

When diagram D represents (M), the optimal value z^* of (M) is the minimum of $z^*(x(p))$ over all r–t paths p. A minimum cost solution may not correspond to a shortest path since arc weights reflect only the cost of integer variables.

We similarly define Δ-optimal solutions by supposing that y is set to an optimal value. Thus we say that $x = \bar{x}$ is the projection of a Δ-optimal solution of (M) when $z^*(\bar{x}) \leq z^* + \Delta$. We also define M($\Delta$) to be the x-projection of Δ-optimal solutions of (M). An r–t path p of a decision diagram D represents the x-projection of a Δ-optimal solution when $x(p)$ is Δ-optimal. Diagram D exactly represents M(Δ) if its r–t paths represent all and only points in M(Δ).

4.7.1 Sound Decision Diagrams for MILP

A diagram D is *sound* for M(Δ) when every Δ-optimal solution of (M) is represented by an r–t path of D, and every r–t path either represents a Δ-optimal solution or a Δ-costly solution. (Recall that a solution is Δ-*costly* if its cost is greater than $z^*+\Delta$, and Δ-*cheap* otherwise, and among Δ-cheap solutions the feasible and infeasible solutions are called Δ-*optimal* and Δ-*fake* respectively.)

Lemmas 3 and 4 clearly carry over to the MILP case. However, the proof of Theorem 1 breaks down. In fact, a proper sound diagram for M(0) can be minimal, and it can be more compact than a reduced diagram that represents M(0) exactly. This means that sound diagrams can be beneficial for representing optimal solutions of MILPs, as well as for representing suboptimal solutions.

This can be seen in a small counterexample to Theorem 1. Consider the MILP

$$\min \left\{ x_1 + x_2 + y_1 \;\middle|\; y_1 \geq 1 - x_1 - x_2, \;\; x_1, x_2 \in \{0,1\}, \; y_1 \geq 0 \right\}$$

The reduced diagram D in Fig. 4.3 exactly represents M(0). Each of the three feasible integer solutions has cost 1, and each is therefore optimal. The diagram D' in the figure is sound, proper, and minimal for $\Delta = 0$. It is sound because it contains paths that represent the three optimal solutions, and the remaining path represents a solution with cost greater than 1. It is proper because this fourth path is suboptimal. It is minimal because every node and every arc is part of a minimum-cost path. Diagram D' therefore refutes Theorem 1. Furthermore, it is more compact than diagram D.

4.7.2 Building Sound Diagrams for MILP

We now lay the groundwork for construction of sound decision diagrams using top-down compilation. The diagrams are built in much the same way as a branching tree, except that nodes are superimposed along the way, based on state information, resulting in a directed acyclic graph rather than a tree. It is assumed that the original optimization problem (M) has been solved by some method and the optimal value z^* is known. No other information from the solution is used.

The state of a node u in layer U_j is a pair $(\bar{b}, v)$, where $\bar{b}$ is interpreted as a right-hand side (RHS) of the MILP constraint set that can result from fixing variables $x_1, \ldots, x_{j-1}$, and v is the length of a shortest r-u path. Thus $\bar{b}$ is the RHS of a constraint set of the form

$$A_{[j]} x_{[j]} + By \geq \bar{b} \tag{4.5}$$

where $x_{[j]} = (x_j, \ldots, x_n)$ and matrix $A_{[j]}$ consists of columns $j, \ldots, n$ of A. We will refer to $\bar{b}$ as the *RHS state* and to v as the *length state*. At the terminal node, $x_{[n+1]}$ is the tuple of length 0, and similarly for $A_{[n+1]}$, so that $\bar{b}$ is interpreted as the RHS of the linear system $By \geq \bar{b}$.

The RHS state $\bar{b}$ defines a set of *feasible suffixes* in $\mathrm{Suf}_D(u)$; that is, a set of suffixes $x_{[j]}$ for which

$$A_{[j]} x_{[j]} + By \geq \bar{b}, \;\; x_{[j]} \in S_{[j]}, \; y \geq 0 \tag{4.6}$$

is satisfied by some y, where $S_{[j]} = S_1 \times \cdots \times S_n$. At the terminal node, the set of feasible suffixes is a singleton containing the 0-length tuple $x_{[n+1]}$ if $By \geq \bar{b}$, $y \geq 0$ is feasible, and is the empty set otherwise. The RHS state also defines a lower bound $L_j(\bar{b})$ on the cost of any feasible suffix, namely the optimal value of the LP relaxation of (M) with variables $x_1, \ldots, x_{j-1}$ removed and the RHS set to $\bar{b}$. Thus

$$L_j(\bar{b}) := \min \left\{ c_{[j]} x_{[j]} + dy \;\middle|\; A_{[j]} x_{[j]} + By \geq \bar{b}, \; x_{[j]} \in \overline{S}_{[j]}, \; y \geq 0 \right\} \tag{4.7}$$

where $\overline{S}_{[j]}$ is the convex hull of S_j, which is a closed interval $[\alpha_j, \beta_j]$ for some $\alpha_j \leq \beta_j$. Setting $x_j = \delta_j$ at a node with state $(\bar{b}, v)$ creates a transition to a new state $(\bar{b} - A_j\delta_j, v + c_j\delta_j)$.

We can build a sound decision diagram D by branching on the possible values δ_j of x_j at each node in each layer U_j. Each branch creates a transition to a new state $(\bar{b} - A_j\delta_j, v + c_j\delta_j)$. If the bound $L_{j+1}(\bar{b} - A_j\delta_j)$ at the new state satisfies the *bounding test*

$$v + c_j\delta_j + L_{j+1}(\bar{b} - A_j\delta_j) > z^* + \Delta \tag{4.8}$$

there is no need to create an arc that leads to this state. In this case, the left-hand side of (4.8) is, as shown below, a lower bound on the cost of any r–t path containing a. If the bounding test is failed, we create an arc a with label δ_j that runs from u to any node on the next layer that has RHS state $\bar{b} - A_j\delta_j$.

When two arcs transition to the same RHS state, they can lead to the same node or to distinct nodes with the same RHS state but different length states. Distinguishing nodes with different length states has the advantage that the resulting length states are generally larger, and the bounding test is passed more often, leading to the deletion of more arcs that will not be part of a Δ-optimal solution. On the other hand, identifying nodes with the same RHS state can help prevent an explosion in the size of the decision diagram. Even this may be inadequate to prevent an explosion, because most of the RHS states are likely to be different.

To deal with this problem, we take advantage of the fact that an entire range of RHS states $\bar{b}$ can result in the same set of feasible suffixes and can therefore be regarded as *equivalent* states. For example, suppose that (M) is the small problem instance

$$\min\left\{4x_1 + 5x_2 - y_1 \;\middle|\; x_1 + 3x_2 - y_1 \geq 3,\ x_1, x_2 \in \{0, 1, 2, 3\},\ y_1 \geq 0\right\} \tag{4.9}$$

The optimal solution is $(x_1, x_2, y_1) = (0, 1, 0)$, with optimal value $z^* = 5$. In layer 2 of a sound diagram for this problem, inequality (4.5) is $3x_2 - u_1 \geq \bar{b}$, and all RHS states in the interval $[\epsilon, 3]$ are equivalent (where $\epsilon > 0$ is an arbitrarily small number). The set of feasible suffixes for any $\bar{b} \in [\epsilon, 3]$ is $\{1, 2, 3, \ldots\}$ because, for any such $\bar{b}$, $y_1 \geq 0$ implies that $3x_2 - y_1 \geq \bar{b}$ is satisfiable precisely when $x_2 \in \{1, 2, 3, \ldots\}$.

We therefore allow an arc with label δ_j from RHS state $\bar{b}$ to lead to any RHS state that is equivalent to $\bar{b} - A_j\delta_j$. This can result in fewer nodes in the next layer of the diagram. For instance, suppose that while creating a sound diagram for example (4.9) we branch on x_1 at the root node r, as shown in Fig. 4.4(a). The four branches transition to RHS states 3, 2, 1, and 0, respectively. However, the first three RHS states are equivalent, and the corresponding arcs can lead to the same node. This results in two nodes rather than four in layer U_2.

When multiple arcs lead to the same node, the RHS state of the node is the smallest of the RHS states that result from the corresponding transitions. Simiarly, the length state of the node is the smallest of the length states that result from the

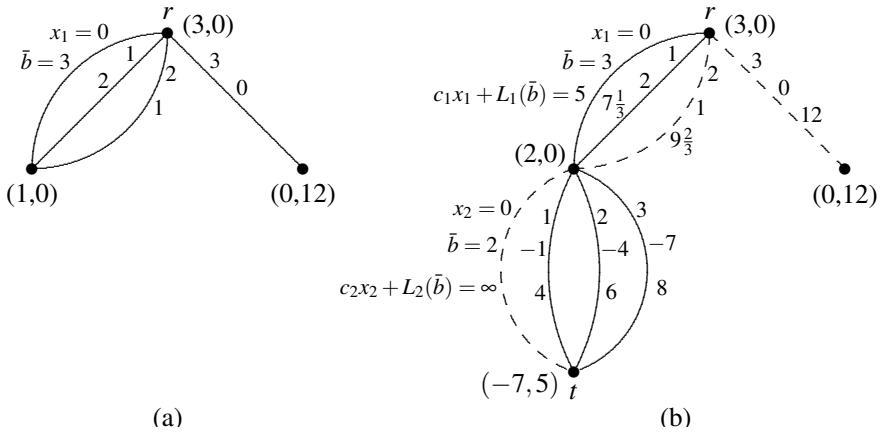


Figure 4.4: (a) Branches at the root node for a small MILP instance. Three branches lead to equivalent RHS states. Each arc shows its label and the corresponding RHS state. (b) Sound decision diagram for the same instance with $\Delta = 4$ (solid arcs). Dashed arcs are are not generated, due to failure of the bounding test. Each arc shows its label, RHS state, and cost bound.

transitions. This ensures that the bounding mechanism is valid. The leftmost three arcs in Fig. 4.4(a) therefore lead to a node with state $(1, 0)$, where the RHS state 1 is the minimum of the RHS states 3, 2, and 1, and the length state 0 is the minimum of the three arc lengths 0, 4, and 8.

Figure 4.4(b) shows a sound diagram for problem (4.9) when $\Delta = 4$. The three leftmost branches at the the root create transitions to states $(3, 0)$, $(2, 4)$, and $(1, 8)$, respectively. The corresponding bounds are $L_2(3) = 5$, $L_2(2) = 3\frac{1}{3}$, and $L_2(1) = 1\frac{2}{3}$. The first two branches fail the bounding test (4.8), but the third passes because $v + 2c_1 + L_2(1) = 0 + 8 + 1\frac{2}{3} > 5 + 4 = z^* + \Delta$. The corresponding arc is therefore omitted (and shown as a dashed line). The branches corresonding to $x_1 = 0, 1$ generate arcs that lead to the same node, because the corresponding RHS states 3 and 2 are equivalent. This node has state $(2, 0)$, because its RHS state is the minimum of 3 and 2, and its length state is the minimum of the length states 0 and 4. The branch corresponding to $x_1 = 3$ passes the bounding test, and no arc is created. Finally, one of the four branches at state $(2, 0)$ pass the bounding test, leaving three arcs to the terminus.

Four of the r–t paths in the resulting diagram represent solutions $(x_1, x_2, y_1) = (0, 1, 0)$, $(0, 2, 3)$, $(1, 1, 1)$, $(0, 3, 6)$ which have values 5, 7, 8, and 9 respectively. These are precisely the 4-optimal solutions of (4.9). The other two solutions, namely $(x_1, x_2, y_1) = (1, 2, 4)$ and $(1, 3, 7)$ which have value 10 and 12 respectively, represent solutions that are 4-costly. These solutions must be discarded as paths in the diagram are enumerated.

The basic theorem that justifies top-down compilation is stated below. In the-

ory, a sound diagram can admit paths that represent infeasible solutions, but the equivalency requirement for RHS states results in feasible solutions only.

Theorem 4. *Consider a decision diagram D with layers $U_1, \dots, U_{n+1}$ and problem (M) with optimal value z^*. Suppose that each node $u \in U_j$ of D $(j = 1, \dots, n+1)$ is associated with a state $(\bar{b}, v)$, where the root node r is associated with state $(b, 0)$. Suppose further than for each $\delta_j \in S_j$ for which $v + c_j\delta_j + L_j(\bar{b}) \leq z^* + \Delta$, there is exactly one arc from u to a node in U_{i+1} with state (b', v'), where b' is equivalent to $\bar{b} - A_j\delta_j$, $b' \leq \bar{b} - A_j\delta_j$, and $v' \leq v + c_j\delta_j$. Then D is sound for $\mathrm{M}(\Delta)$. Moreover, every r–t path of D represents a feasible solution of (M).*

Proof. We begin by showing that for any node $u \in U_j$ in D with state $(\bar{b}, v)$, $v + L_j(\bar{b})$ is a lower bound on the cost $z^*(x(p))$ of any r–t path p that contains u. Let $\delta = x(p)$. We know that

$$\begin{aligned}\min\left\{ \sum_{j'=1}^{j-1} c_{j'}\delta_{j'} + c_{[j]}x_{[j]} + dy \;\middle|\; A_{[j]}x_{[j]} + By \geq b - \sum_{j'=1}^{j-1} A_{j'}\delta_{j'},\ x_{[j]} \in S_{[j]},\ y \geq 0 \right\} \\ \leq \min\left\{ c\delta + dy \;\middle|\; By \geq b - A\delta,\ y \geq 0 \right\} = z^*(x(p))\end{aligned} \tag{4.10}$$

because the minimization problem on the right of the inequality is a restriction of the one on the left. By construction of the states, we have that

$$v \leq \sum_{j'=1}^{j-1} c_{j'}\delta_{j'} \quad \text{and} \quad \bar{b} \leq b - \sum_{j'=1}^{j-1} A_{j'}\delta_{j'}$$

These and (4.10) imply

$$\min\left\{ v + c_{[j]}x_{[j]} + dy \;\middle|\; A_{[j]}x_{[j]} + By \geq \bar{b},\ x_{[j]} \in S_{[j]},\ y \geq 0 \right\} \leq z^*(x(p))$$

By definition of $L_j(\bar{b})$, this implies $v + L_j(\bar{b}) \leq z^*(x(p))$, as desired.

To show that D is sound, we first show that any Δ-optimal solution $x = \delta$ of (M) is represented by an r–t path p of D. It suffices to show, by induction on layers, that for each layer U_j, there is an r-u path π for which $u \in U_i$ and $x(\pi) = (\delta_1, \dots, \delta_{j-1})$. The claim is trivially true for layer U_1. We therefore assume that the claim is true for layer U_j and show that there is an r-u' path π' for some $u' \in U_{j+1}$ for which $x(\pi') = (\delta_1, \dots, \delta_j)$. It suffices to show that there is an arc a leaving u with label δ_j, because we can let u' be the node at the other end of a. If $(\bar{b}, v)$ is the state at node u, we know that setting $x_j = \delta_j$ transitions to state $(\bar{b} - A_j\delta_j, v + c_j\delta_j)$. As shown above, the associated bound $L_{j+1}(\bar{b} - A_j\delta_j)$ satisfies $v + c_j\delta_j + L_{j+1}(\bar{b} - A_j\delta_j) \leq z^*(x(p))$. But since it is given that $z^*(x(p)) \leq z^* + \Delta$, we have $v + c_j\delta_j + L_{j+1}(\bar{b} - A_j\delta_j) \leq z^* + \Delta$, and arc a with label δ_j therefore occurs in D, by construction.

We must now show that every r–t path p in D either represents a Δ-optimal solution or a Δ-costly solution of (M). We will show that p in fact represents a

feasible solution of (M). Thus if its cost is at most $z^* + \Delta$, p represents a Δ-optimal solution, and otherwise it is Δ-costly. This proves the theorem.

Let $\delta = x(p)$. We first show that the RHS state $\bar{b}$ of any node $u \in U_j$ on path p is equivalent to state $b - \sum_{j'=1}^{j-1} A_{j'}\delta_{j'}$. This is trivially true for $j = 1$. We therefore suppose it is true for layer U_j and show that it is true for the node u' of path p in layer U_{j+1}. That is, we wish to show that the RHS state b' of u' is equivalent to state $b - \sum_{j'=1}^{j} A_{j'}\delta_{j'}$. Because $\bar{b}$ is equivalent to $b - \sum_{j'=1}^{j-1} A_{j'}\delta_{j'}$ by the induction hypothesis, we have

$$\begin{aligned}\left\{x_{[j]} \in S_{[j]} \;\middle|\; A_{[j]}x_{[j]} + By \geq \bar{b} \text{ for some } y \geq 0\right\} = \\ \left\{x_{[j]} \in S_{[j]} \;\middle|\; A_{[j]}x_{[j]} + By \geq b - \sum_{j'=1}^{j-1} A_{j'}\delta_{j'} \text{ for some } y \geq 0\right\}\end{aligned} \tag{4.11}$$

Also state b' is equivalent to $\bar{b} - A_j\delta_j$ by construction of D, which means

$$\begin{aligned}\left\{x_{[j+1]} \in S_{[j+1]} \;\middle|\; A_{[j+1]}x_{[j+1]} + By \geq b' \text{ for some } y \geq 0\right\} = \\ \left\{x_{[j+1]} \in S_{[j+1]} \;\middle|\; A_{[j+1]}x_{[j+1]} + By \geq \bar{b} - A_j\delta_j \text{ for some } y \geq 0\right\}\end{aligned} \tag{4.12}$$

To show that state b' is equivalent to $b - \sum_{j'=1}^{j} A_{j'}\delta_{j'}$, we show that $X_1 = X_2$, where

$$\begin{aligned}X_1 &= \left\{x_{[j+1]} \in S_{[j+1]} \;\middle|\; A_{[j+1]}x_{[j+1]} + By \geq b' \text{ for some } y \geq 0\right\} \\ X_2 &= \left\{x_{[j+1]} \in S_{[j+1]} \;\middle|\; A_{[j+1]}x_{[j+1]} + By \geq b - \sum_{j'=1}^{j} A_j\delta_j \text{ for some } y \geq 0\right\}\end{aligned}$$

Suppose first that $\bar{x}_{[j+1]} \in X_1$. Then $A_{[j+1]}\bar{x}_{[j+1]} + By' \geq b'$ for some $y' \geq 0$. Due to (4.12), this implies

$$A_{[j+1]}\bar{x}_{[j+1]} + B\bar{y} \geq \bar{b} - A_j\delta_j$$

for some $\bar{y} \geq 0$. This implies $A_j\delta_j + A_{[j+1]}\bar{x}_{[j+1]} + B\bar{y} \geq \bar{b}$, which implies by (4.11) that

$$A_j\delta_j + A_{[j+1]}\bar{x}_{[j+1]} + B\hat{y} \geq b - \sum_{j'=1}^{j-1} A_j\delta_j$$

for some $\hat{y} \geq 0$. This implies $A_{[j+1]}\bar{x}_{[j+1]} + B\hat{y} \geq b - \sum_{j'=1}^{j} A_j\delta_j$, which means that $\bar{x}_{[j+1]} \in X_2$. A similar argument shows that if $\bar{x}_{[j+1]} \in X_2$, then $\bar{x}_{[j+1]} \in X_1$, and we conclude that $X_1 = X_2$.

We have shown that the RHS state of any node in U_j on path p is equivalent to $b - \sum_{j'=1}^{j-1} A_j\delta_j$. Thus, in particular, the terminal node t has an RHS state $\hat{b}$ that is equivalent to $b - A\delta$. Now consider the node u of p in layer U_n, which we may assume has state $(\bar{b}, v)$. Setting $x_n = \delta_n$ at u transitions to the RHS state $\bar{b} - A_n\delta_n$. Since the

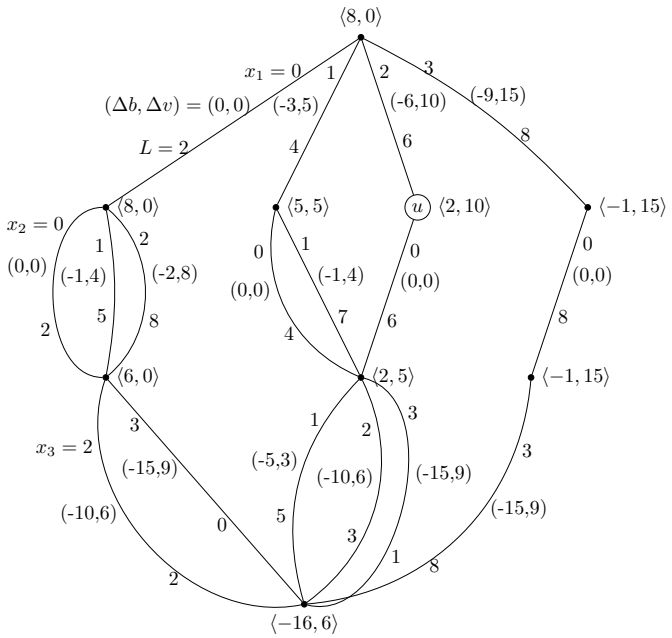


Figure 4.5: A MILP-SDD with $z^* = 2$ and $\Delta = 6$. Although every path through node u has cost $> z^* + 3$, it cannot be eliminated by bounding tests

corresponding arc occurs in D, $\bar{b} - A_n\delta_n$ is equivalent to $\hat{b}$, which in turn is equivalent to $b - A\delta$. We also have from (4.8) that

$$v + c_n\delta_n + L_n(\bar{b} - A_n\delta_n) \leq z^* + \Delta \tag{4.13}$$

Now if δ is infeasible, RHS state $b - A\delta$ has no feasible suffixes, which means that the equivalent state $\bar{b} - A_n\delta_n$ has no feasible suffxes. This implies that $L_n(\bar{b} - A_n\delta_n)$ is infinite, which is inconsistent with (4.13). We conclude that δ is feasible, and p represents a feasible solution. □

While this construction yields a valid sound decision diagram for M(Δ), it does not yield a particularly strong bounding test because the RHS state and length state are updated independently of each other. As an example, consider $\min\{5x_1 + 4x_2 + 3x_3 - y_1 : 3x_1 + x_2 + 5x_3 - y_1 \geq 8,\ x_1, x_2, x_3 \in \{0, 1, 2, 3\},\ y_1 \geq 0\}$. The optimal solution is $(x_1, x_2, x_3, y_1) = (0, 0, 3, 7)$ with value 2. Figure 4.5 shows a reduced sound decision diagram with $\Delta = 6$. Here each arc is labeled with the value of x_j, the change in righthand side and objective value (Δb and Δv respectively), and the lower bound given by the LP relaxation $L = v + c_j x_j + L_{j+1}(b - \Delta b)$. Now suppose $\delta = 3$; then every solution through node u has cost at least $6 > z^* + \delta$, so we would like to eliminate it. But the tightest lower bound we can obtain is

$$w(r, u) + w(u, t) + \mathrm{LP}(u) = 10 + 3 + \min\{-y_1 : -y_1 \geq 8 - 6 - 15\} = 0$$

where $w(r,u)$ is the length of the minimum weight r–u path, and similarly for $w(u,t)$; since $0 \leq 5 = z^* + 3$, we cannot eliminate u using the bounding test. Intuitively, this is because the diagram does not exactly represent the cost of a given path, and therefore may strictly under-approximate the cost of a given solution. In fact, the lower bound on z^* we obtain from the terminal node t (based on its node state) is -10, which is quite far from the actual optimal value $z^* = 2$.

To improve the lower bounds, we change how we define the node state when multiple arcs lead to the same node. In particular, the RHS state and length state of a node $u \in U_j$ are set to the those resulting from the incoming prefix with the *lowest LP-based bound*, i.e., the one that minimizes the lower bound $v + c_j\delta_j + L_{j+1}(\bar{b} - A_j\delta_j)$ on the cost of any path that goes through u. Intuitively, this corresponds to keeping track of the "most promising" prefix at each node, in the sense that it has the weakest corresponding LP-based lower bound among all other prefixes of that node (and therefore is more likely to extend to a near-optimal solution). Perhaps surprisingly, this also yields a valid sound decision diagram for $\mathrm{M}(\Delta)$:

Theorem 5. *Consider a decision diagram D with layers $U_1, \ldots, U_{n+1}$ and problem (M) with optimal value z^*. Suppose that each node $u \in U_j$ of D ($j = 1, \ldots, n+1$) is associated with a state $(\bar{b}, v)$, where the root node r is associated with the state $(b, 0)$. Suppose further that for each $\delta_j \in S_j$ for which $v + c_j\delta_j + L_{j+1}(\bar{b} - A_j\delta_j) \leq z^* + \Delta$, there is exactly one arc from u to a node $u' \in U_{j+1}$ with state $(b - A_{(j+1)}\delta^*_{(j+1)}, c_{(j+1)}\delta^*_{(j+1)})$, where $b - A_{(j+1)}\delta^*_{(j+1)}$ is equivalent to $\bar{b} - A_j\delta_j$ and $\delta^*_{(j+1)}$ is the prefix of u' that minimizes $c_{(j+1)}\delta_{(j+1)} + L_{j+1}(b - A_{(j+1)}\delta_{(j+1)})$. Then D is sound for $\mathrm{M}(\Delta)$. Moreover, every r-t path of D can be extended to a feasible solution to (M).*

Proof. We begin by showing that for any node $u \in U_j$ in D with state $(\bar{b}, v)$, $v + L_j(\bar{b})$ is a lower bound on the cost $z^*(x(p))$ of any r-t path p that contains u. Let $\delta = x(p)$. Then by definition of L_j

$$
\begin{aligned}
&c_{(j)}\delta_{(j)} + L_j(b - A_{(j)}\delta_{(j)}) \\
&= \min\left\{c_{(j)}\delta_{(j)} + c_{[j]}x_{[j]} + dy \mid A_{[j]}x_{[j]} + By \geq b - A_{(j)}\delta_{(j)}, x_{[j]} \in \overline{S}_{[j]}, y \geq 0\right\} \\
&\leq \min\left\{c\delta + dy \mid By \geq b - A\delta, y \geq 0\right\} = z^*(x(p))
\end{aligned}
$$

because the minimization problem on the right of the inequality is a restriction of the one on the left. On the other hand, $v + L_j(\bar{b}) = c_{(j)}\delta^*_{(j)} + L_j(b - A_{(j)}\delta^*_{(j)}) \leq c_{(j)}\delta_{(j)} + L_j(b - A_{(j)}\delta_{(j)})$ where $\delta^*_{(j)}$ by definition minimizes the rightmost expression among all prefixes $\widehat{\delta}_{(j)}$ of u. Hence $v + L_j(\bar{b}) \leq z^*(x(p))$.

To show that D is sound, we first show that any Δ-optimal solution $x = \delta$ of (M) is represented by an r-t path of D. It suffices to show, by induction on layers, that for each layer U_j, there is an r-u path π for which $u \in U_i$ and $x(\pi) = (\delta_1, \ldots, \delta_{j-1})$. The claim is trivially true for U_1. We therefore assume that the claim is true for layer U_j and show that there is an r-u' path π' for some $u' \in U_{j+1}$ for which $x(\pi') = (\delta_1, \ldots, \delta_j)$.

It suffices to show that there is an arc a leaving u with label δ_j, because we can let u' be the node at the other end of a.

If $(\bar{b}, v)$ is the state at node u, we know that setting $x_j = \delta_j$ transitions to a state (b', v') equivalent to $(\bar{b} - A_j\delta_j, v + c_j\delta_j)$. As shown above, the associated bound $v + L_{j+1}(b') \leq z^*(x(p))$. But since it is given that $z^*(x(p)) \leq z^* + \Delta$, we have $v + c_j\delta_j + L_{j+1}(\bar{b} - A_j\delta_j) \leq z^* + \Delta$, and therefore arc a with label δ_j occurs in D, by construction.

We must now show that every r-t path p in D either represents a Δ-optimal solution or a Δ-costly solution of (M). We will show that p in fact represents a feasible solution of (M). Thus if its cost is at most $z^* + \Delta$, p represents a Δ-optimal solution, and otherwise it is Δ-costly. This proves the theorem.

Let $\delta = x(p)$. We first show that the RHS state $\bar{b}$ of any node $u \in U_j$ on path p is equivalent to $b - A_{(j)}\delta_{(j)}$. This is trivially true for $j = 1$. We therefore suppose it is true for layer U_j and show that it is true for the node u' of path p in layer U_{j+1}. That is, we wish to show that the RHS state b' of u' is equivalent to state $b - A_{(j+1)}\delta_{(j+1)}$. Notationally, we want to show $X_{j+1}(b') = X_{j+1}(b - A_{(j+1)}\delta_{(j+1)})$, where

$$X_j(\bar{b}) = \left\{ x_{[j]} \in S_{[j]} \mid A_{[j]}x_{[j]} + By \geq \bar{b} \text{ for some } y \geq 0 \right\}$$

is the set of "feasible suffixes" of a node in layer U_j with RHS state $\bar{b}$.

Because $\bar{b}$ is equivalent to $b - A_{(j)}\delta_{(j)}$ by the induction hypothesis, we have $X_j(\bar{b}) = X_j(b - A_{(j)}\delta_{(j)})$. This implies that $X_{j+1}(\bar{b} - A_j\xi_j) = X_{j+1}(b - A_{(j)}\delta_{(j)} - A_j\xi_j)$ for any ξ_j corresponding to an outgoing arc of u, because they are precisely the suffixes in each set whose first component is ξ_j. In particular, we have $X_{j+1}(\bar{b} - A_j\delta_j) = X_{j+1}(b - A_{(j)}\delta_{(j)} - A_j\delta_j) = X_{(j+1)}(b - A_{(j+1)}\delta_{(j+1)})$. Also state b' is equivalent to $\bar{b} - A_j\delta_j$ by construction of D, which means $X_{j+1}(b') = X_{(j+1)}(\bar{b} - A_j\delta_j)$. Together these imply $X_{(j+1)}(b') = X_{(j+1)}(b - A_{(j+1)}\delta_{(j+1)})$.

We have shown that the RHS state of any node in U_j on path p is equivalent to $b - A_{(j)}\delta_{(j)}$. Thus, in particular, the terminal node t has an RHS state $\hat{b}$ that is equivalent to $b - A\delta$. Now consider the node u of p in layer U_n, which we may assume has state $(\bar{b}), v)$. Setting $x_n = \delta_n$ at u transitions to the RHS state $\bar{b} - A_n\delta_n$. Since the corresponding arc occurs in D, $\bar{b} - A_n\delta_n$ is equivalent to $\hat{b}$, which in turn is equivalent to $b - A\delta$. We also have from the bounding test that

$$v + c_n\delta_n + L_n(\bar{b} - A_n\delta_n) \leq z^* + \Delta$$

Now if δ is infeasible, RHS state $b - A\delta$ has no feasible suffixes, which means that the equivalent state $\bar{b} - A_n\delta_n$ has no feasible suffixes. This implies that $L_n(\bar{b} - A_n\delta_n)$ is infinite, which is inconsistent with the bounding test. We conclude that δ is feasible, and p represents a feasible solution. □

4.7.3 Sound Reduction

Similar to the ILP case, we can define a notion of sound reduction for MILP. Specifically, given distinct nodes $u \neq v \in U_j$, we say that we *reduce* node u into node v when we redirect all incoming arcs of u to v, remove all outgoing arcs of u, and remove u. We say that we can *sound-reduce* u into v when we can reduce u into v while maintaining the soundness property of the diagram, i.e.,

1. (*Preservation*) The reduction does not eliminate any Δ-optimal solutions (i.e., feasible solutions with cost at most $z^* + \Delta$).

2. (*Validity*) The reduction does not introduce any Δ-fake solutions (i.e., infeasible solutions with cost less than or equal to $z^* + \Delta$).

Note that the validity condition is equivalent to ensuring that every newly introduced solution is either Δ-optimal or Δ-costly (i.e., has cost greater than $z^* + \Delta$).

The following theorem, which generalizes Theorem 2 from [HS17], gives necessary and sufficient conditions for sound-reducing one node into another node in fairly general terms:

Theorem 6. *Suppose we have a sound decision diagram containing nodes $u \neq v \in U_j$ for some j. Then u can be sound-reduced into v if and only if*

$$\min_{\substack{\pi \in \mathrm{Pre}(u) \\ \sigma \in \mathrm{Suf}(u) \ominus \mathrm{Suf}(v)}} \left\{ c_{(j)} x(\pi) + c_{[j]} x(\sigma) + \underline{\mathrm{LP}}_y(x(\pi, \sigma)) \right\} > z^* + \Delta$$

where $\mathrm{Pre}(u)$ *is the set of prefixes of* u, $\mathrm{Suf}(u)$ *is the set of suffixes of* u, $\ominus$ *denotes symmetric difference (i.e.,* $A \ominus B := (A \setminus B) \cup (B \setminus A)$*), and*

$$\underline{\mathrm{LP}}_y(x) = \min_{y \geq 0} \{dy : By \geq b - Ax\}$$

Proof. We will show that the stated condition is equivalent to the combination of the preservation and validity conditions.

(*Preservation*): The solutions which are eliminated as a result of reduction are precisely those consisting of a prefix of u together with a suffix of u that is not a suffix of v, i.e., $x(\pi, \sigma)$ such that $\pi \in \mathrm{Pre}(u)$ and $\sigma \in \mathrm{Suf}(u) \setminus \mathrm{Suf}(v)$. To say that no such solutions are Δ-optimal is equivalent to saying that all such solutions have cost greater than $z^* + \Delta$ (since infeasible solutions have cost $+\infty$ due to the $\underline{\mathrm{LP}}_y$ term and therefore always have cost greater than $z^* + \Delta$), i.e.,

$$\min_{\substack{\pi \in \mathrm{Pre}(u) \\ \sigma \in \mathrm{Suf}(u) \setminus \mathrm{Suf}(v)}} \left\{ c_{(j)} x(\pi) + c_{[j]} x(\sigma) + \underline{\mathrm{LP}}_y(x(\pi, \sigma)) \right\} > z^* + \Delta$$

(*Validity*): The solutions which are created as a result of reduction are precisely those consisting of a prefix of u that is not a prefix of v, together with a suffix of v that

is not a suffix of u, i.e., $x(\pi, \sigma)$ such that $\pi \in \text{Pre}(u) \setminus \text{Pre}(v)$ and $\sigma \in \text{Suf}(v) \setminus \text{Suf}(u)$. By construction of the decision diagram, $\text{Pre}(u) \cap \text{Pre}(v) = \emptyset$, so it is equivalent to consider $\pi \in \text{Pre}(u)$. Furthermore such an $x(\pi, \sigma)$ cannot be Δ-optimal, since this would imply that the diagram prior to sound-reduction was not sound (since it was missing this newly created Δ-optimal solution), contrary to assumption. Hence it is equivalent to ensure that all such $x(\pi, \sigma)$ has cost greater than $z^* + \Delta$, i.e.,

$$\min_{\substack{\pi \in \text{Pre}(u) \\ \sigma \in \text{Suf}(v) \setminus \text{Suf}(u)}} \left\{ c_{(j)} x(\pi) + c_{[j]} x(\sigma) + \underline{\text{LP}}_y(x(\pi, \sigma)) \right\} > z^* + \Delta$$

Combining the two conditions gives us the stated condition. □

The relationship to Theorem 2 becomes clearer when we note that the preservation condition can also be stated as

$$\text{Suf}_\Delta(u) \subseteq \text{Suf}(v)$$

where $\text{Suf}_\Delta(u)$ denotes the set of suffixes of u that are part of some Δ-optimal solution. To see why, note that the solutions which are eliminated as a result of reduction are precisely those involving suffixes of u that are not suffixes of v, and the set of such suffixes with cost at most $z^* + \Delta$ is precisely $\text{Suf}_\Delta(u) \setminus \text{Suf}(v)$. To say that this set is empty is equivalent to saying that $\text{Suf}_\Delta(u) \subseteq \text{Suf}(v)$.

The following corollaries of Theorem 6 provide more concrete sufficient conditions for sound reduction:

Corollary 4. *Suppose we have a sound decision diagram containing nodes $u \neq v \in U_j$ for some j. Suppose we have a function $f : \mathbb{R}^n \to \mathbb{R}$ such that*

$$f(x) \leq cx + \underline{\text{LP}}_y(x) \qquad \text{for all } x_j \in S_j,\ j = 1, \ldots, n$$

Then the following is a sufficient condition for sound-reducing u into v:

$$\min_{\substack{\pi \in \text{Pre}(u) \\ \sigma \in \text{Suf}(u) \ominus \text{Suf}(v)}} f(x) > z^* + \Delta$$

In particular, if $f(x) = \sum_{i=1}^n f_i(x_i)$ for $f_i : \mathbb{R} \to \mathbb{R}$ (i.e., $f(x)$ is separable), then the sufficient condition can be written as

$$\min_{\pi \in \text{Pre}(u)} \sum_{i \in (j)} f_i(x(\pi)) + \min_{\sigma \in \text{Suf}(u) \ominus \text{Suf}(v)} \sum_{i \in [j]} f_i(x(\sigma)) > z^* + \Delta$$

Proof. Since $f(x)$ is a lower bound on $cx + \underline{\text{LP}}_y(x)$, the stated condition is even stronger than the condition of theorem 6. □

Corollary 5. *Let $F := \{\lambda : B^T \lambda \leq d, \lambda \geq 0\}$. Then for any $\lambda \in F$, the following is a sufficient condition for sound-reducing u into v:*

$$(c - \lambda^T A) x(\pi, \sigma) + \lambda^T b > z^* + \Delta \qquad \forall \pi \in \text{Pre}(u), \sigma \in \text{Suf}(u) \Delta \, \text{Suf}(v)$$

Proof. We first show that F is non-empty, and therefore such λ's do exist. As such, suppose for contradiction that F is empty, i.e., the system of inequalities $B^T\lambda \leq d, \lambda \geq 0$ is infeasible. By general theorems of alternative (e.g., [Gal89]), this implies the existence of $r \geq 0$ such that $Br \geq 0$ and $d^T r < 0$. By assumption, (M) has an optimal solution (x^*, y^*) with some finite optimal value. Then for any $t > 0$,

$$Ax^* + B(y^* + tr) = Ax^* + By^* + tBr \geq b \qquad y^* + tr \geq 0$$
$$cx^* + d(y^* + tr) = cx^* + dy^* + t \cdot dr < cx^* + dy^*$$

and therefore $y^* + tr$ is a feasible solution of (M) with lower objective value than (x^*, y^*), a contradiction.

We now proceed with the proof. By LP duality and the non-emptiness of F,

$$\begin{aligned} cx + \underline{\text{LP}}_y(x) &= cx + \min_y\{dy : By \geq b - Ax, y \geq 0\} \\ &= cx + \max_\lambda\{\lambda^T(b - Ax) : B^T\lambda \leq d, \lambda \geq 0\} \\ &= cx + \max_\lambda\{\lambda^T(b - Ax) : \lambda \in F\} \\ &\geq cx + \lambda^T(b - Ax) = (c - \lambda^T A)x + \lambda^T b \qquad \forall \lambda \in F \end{aligned}$$

Hence, $(c - \lambda^T A)x + \lambda^T b \leq cx + \underline{\text{LP}}_y(x)$ for all feasible x, and the result follows from corollary 4. □

Suppose the suffixes of u and v all exist as paths in the diagram with arc weights given by $f(x) = (c - \lambda^T A)x + \lambda^T b$, and for every node u the minimum cost path from u to the terminal node t is known and given by $w(u, t)$. Then this test can be implemented reasonably efficiently by recursively computing the *least cost-differing suffix* $\text{LCDS}_j(u, v)$, defined as the minimum cost suffix of u that is not a suffix of v, using the following recursion:

$$\text{LCDS}_{n+1}(u, v) = +\infty$$
$$\text{LCDS}_j(u, v) = (c - \lambda^T A)_j x_j + \min_{a=(u,u_+)} \begin{cases} \text{LCDS}_{j+1}(u_+, v_+) & \exists a' = (v, v_+) \text{ s.t. } \ell(a) = \ell(a') \\ \lambda^T b + w(u_+, t) & \text{otherwise} \end{cases}$$

where $a = (u, u_+)$ is an outgoing arc of u with head node u_+, $a' = (v, v_+)$ is an outgoing arc of v with head node v_+, and $\ell(a)$ and $\ell(a')$ are the labels (i.e., the value assigned to x_j when traversing the arc) of arcs a and a' respectively.

Lemma 5. *Suppose we have a sound decision diagram containing nodes $u \neq v \in U_j$ for some j. Let $\alpha(u)$ be the minimum cost among all prefixes of u and let $\beta_i(u)$ be the weakest RHS value of constraint i among all prefixes of u, i.e.,*

$$\alpha(u) = \min_{\pi \in \text{Pre}(u)} c_{(j)} x(\pi), \qquad \beta_i(u) = \max_{\pi \in \text{Pre}(u)} A_{(j),i} x(\pi) \ \forall i$$

and let $\beta(u) = (\beta_1(u), \ldots, \beta_m(u))$ *be the vector of* $\beta_i(u)$*'s. Then the following is a sufficient condition for sound-reduction* u *into* v*:*

$$\alpha(u) + \min_{\substack{\sigma \in \mathrm{Suf}(u) \ominus \mathrm{Suf}(v) \\ y \geq 0}} \left\{ c_{[j]}x(\sigma) + dy : A_{[j]}x(\sigma) + By \geq b - \beta(u) \right\} > z^* + \Delta$$

Proof. Observe that

$$\begin{aligned} & \min_{\substack{\pi \in \mathrm{Pre}(u) \\ \sigma \in \mathrm{Suf}(u) \setminus \mathrm{Suf}(v)}} \left\{ c_{(j)}x(\pi) + c_{[j]}x(\sigma) + \underline{\mathrm{LP}}_y(x(\pi, \sigma)) \right\} \\ \geq & \underbrace{\min_{\pi \in \mathrm{Pre}(u)} \left\{ c_{(j)}x(\pi) \right\}}_{=\alpha(u)} + \min_{\substack{\pi \in \mathrm{Pre}(u) \\ \sigma \in \mathrm{Suf}(u) \setminus \mathrm{Suf}(v)}} \left\{ c_{[j]}x(\sigma) + \underline{\mathrm{LP}}_y(x(\pi, \sigma)) \right\}, \end{aligned}$$

where

$$\begin{aligned} & \min_{\substack{\pi \in \mathrm{Pre}(u) \\ \sigma \in \mathrm{Suf}(u) \setminus \mathrm{Suf}(v)}} \left\{ c_{[j]}x(\sigma) + \underline{\mathrm{LP}}_y(x(\pi, \sigma)) \right\} \\ = & \min_{\substack{\pi \in \mathrm{Pre}(u) \\ \sigma \in \mathrm{Suf}(u) \setminus \mathrm{Suf}(v) \\ y \geq 0}} \left\{ c_{[j]}x(\sigma) + dy : By \geq b - A_{(j)}x(\pi) - A_{[j]}x(\sigma) \right\} \\ \geq & \min_{\substack{\sigma \in \mathrm{Suf}(u) \setminus \mathrm{Suf}(v) \\ y \geq 0}} \left\{ c_{[j]}x(\sigma) + dy : A_{[j]}x(\sigma) + By \geq b - \beta(u) \right\}. \end{aligned}$$

Thus the lefthand side of the stated condition is a lower bound on $cx + \underline{\mathrm{LP}}_y(x)$ and the result follows from corollary 4. □

4.7.4 Identifying Equivalent States

In general, it is an NP-hard problem to check whether two RHS states are equivalent. We therefore develop some easily checked sufficient conditions for equivalence that can be used in practice. For this purpose, it is convenient to associate with each RHS state $\bar{b}$ an *equivalency range* $[\phi(\bar{b}), \psi(\bar{b})]$, where $\phi(\bar{b}) = (\phi_1(\bar{b}), \ldots, \phi_m(\bar{b}))$ and $\psi(\delta) = (\psi_1(\bar{b}), \ldots, \psi_m(\bar{b}))$. An equivalency range is actually a tuple of m ranges, and we say that $\hat{b} \in [\phi(\bar{b}), \psi(\bar{b})]$ when $\hat{b}_i \in [\phi_i(\bar{b}_i), \psi_i(\bar{b}_i)]$ for all i. The equivalency range has the property that every $\hat{b} \in [\phi(\bar{b}), \psi(\bar{b})]$ is an equivalent RHS state for (4.6). The range may not be maximal with respect to this property, and in fact it is generally hard to compute a maximal range. Nonetheless, we will see that nonmaximal ranges can be useful in practice.

As an example, we identified for problem instance (4.9) the equivalency range $[\epsilon, 3]$ in layer U_2. Since there is only one constraint, the range is a single interval. Thus $[\phi(\bar{b}), \psi(\bar{b})] = [\epsilon, 3]$ for any $\bar{b} \in [\epsilon, 3]$. This is a maximal equivalency range.

Whenever a RHS state $\bar{b}$ is created in layer j, we check whether $\bar{b}$ belongs to an equivalency range that has already been created. If not, we create an equivalency

range $[\phi(\bar{b}), \psi(\bar{b})]$. Because it is difficult to analyze equivalency ranges for an entire constraint set, we seek rules that allow us to derive them from equivalency ranges for individual constraints or small subsets of constraints. Let I index a subset of the constraints (4.6), and let $\bar{b}_I$ be the corresponding RHS. We will say that subset I is *separable* if the problem of finding equivalency ranges for the entire constraint set can be decomposed into finding ranges for I and its complement I' separately. That is, if R_I is an equivalency range for I, and $R_{I'}$ an equivalency range for the remainder of the constraint set, then $(R_I, R_{I'})$ is an equivalency range for the entire constraint set. The following will be a basic tool for this type of analysis

Lemma 6. *If subset I of the constraints (4.6) have no continuous variables in common with the set I' of the remaining constraints, then I is separable.*

Proof. Let $S(\bar{b}_I)$ be the set of feasible suffixes for subset I of constraints (4.6) and RHS state $\bar{b}_I$ in layer U_j. Let R_I be an equivalency range for I, and similarly for $R_{I'}$. We wish to show that $R = (R_I, R_{I'})$ is an equivalency range for $I \cup I'$. We therefore show that for any two states $\bar{b}, \hat{b} \in R$, $S(\bar{b}) = S(\hat{b})$. For this it suffices to show that if $x_{[j]} \in S(\bar{b})$ then $x_{[j]} \in S(\hat{b})$, because the converse follows by symmetry. Because $\bar{b}, \hat{b} \in R$, we have $\bar{b}_I, \hat{b}_I \in R_I$ and $\bar{b}_{I'}, \hat{b}_{I'} \in R_{I'}$, which implies $S(\bar{b}_I) = S(\hat{b}_I)$ and $S(\bar{b}_{I'}) = S(\hat{b}_{I'})$. So since $x_{[j]} \in S(\bar{b}) \subset S(\bar{b}_I)$, we have $x_{[j]} \in S(\hat{b}_I)$, which means that

$$A^I_{[j]} x_{[j]} + B^I y \geq \bar{b}_I \tag{4.14}$$

for some $y \geq 0$. Here A^I consists of the rows of A corresponding to constraint set I, and similarly for B^I. Since $x_{[j]} \in S(\bar{b}) \subset S(\bar{b}_{I'})$, we have $x_{[j]} \in S(\hat{b}_{I'})$, which means that

$$A^{I'}_{[j]} x_{[j]} + B^{I'} y \geq \bar{b}_{I'} \tag{4.15}$$

for some $y \geq 0$. Since (4.14) and (4.15) have no continuous variables in common, we can partition y into tuples y', y'' so that (4.14) contains only variables in y' and (4.15) contains only variables in y''. Thus (4.14) and (4.15) simultaneously hold for some $y = (y', y'')$. This implies that (4.6) holds for some $y \geq 0$, so that $x_{[j]} \in S(\hat{b})$, as desired. □

We immediately infer

Corollary 6. *A constraint that contains no continuous variables is separable.*

We can therefore analyze separately each constraint i that contains only integer variables. A maximal equivalency range for such a constraint can be computed within the decision diagram framework, but this incurs a substantial computational burden. We therefore use a simple rule. When all coefficients of the integer variables in constraint i are nonnegative and $\bar{b}_i \leq 0$, then the constraint is necessarily satisfied by all suffixes, and the same is true if $\bar{b}_i$ is replaced by any nonpositive RHS. We therefore have an equivalency range $[\phi_i(\bar{b}_i), \psi_i(\bar{b}_i)] = [-\infty, 0]$. Similarly, if all coefficients of integer variables are nonpositive and $\bar{b} \geq 0$, we have the equivalency range $[0, \infty]$.

We can also derive a rule for a subset I of constraints that have continuous variables, provided none are shared with other constraints. Suppose every component of $B^I y$ can go to infinity simultaneously, for suitably chosen $y \geq 0$. Then the constraints in I are satisfiable by every suffix, because the left-hand side can be made arbitrarily large without changing the values of the integer variables. To check whether $B^I y$ is unbounded in all components, we solve the LP

$$\max \left\{ \beta \;\middle|\; \beta e \leq B^I y,\ y \geq 0 \right\} \tag{4.16}$$

where e is a tuple of $|I|$ ones.

Corollary 7. *Suppose I indexes a subset of constraints (4.6) that share no continuous variables with other constraints. If (4.16) is unbounded, then $[-\infty, \infty]$ is a equivalency range for the constraints indexed by I, where ∞ denotes a tuple $(\infty, \ldots \infty)$ of length $|I|$. The constraints indexed by I can be ignored when computing equivalency ranges for the entire constraint set (4.6).*

The constraints can be ignored because they are separable (by Lemma 6) and have an equivalency range that places no restriction on the RHS.

We next state a simple rule for a constraint set that may share continuous variables with other constraints.

Lemma 7. *If a continuous variable $y_{j'}$ has a negative coefficient in no constraint of (4.6), then the subset I of constraints that contain $y_{j'}$ with a positive coefficient have the equivalency range $[-\infty, \infty]$. These constraints can be ignored when computing equivalency ranges for (4.6).*

This is because the left-hand sides of the constraints indexed by I can go to infinity simultaneously without affecting any constraints outside I.

If the decision diagram grows too large despite these rules for detecting equivalent RHS states, an additional strategy is available. If the equivalency ranges of two RHS states differ in one or more components, then we can add an artificial variable y_0 (with a coefficient of one) to the constraints corresponding to these components. We also introduce the term My_0 to the objective function of (M), where M is a large positive number. We refer to this procedure as *dualizing* the constraints. This may introduce infeasible solutions to the sound diagram, but they will be screened out when Δ-optimal solutions are extracted from the diagram.

Corollary 8. *Dualized constraints can be ignored when computing equivalency ranges for the entire constraint set. Moreover, for sufficiently large M, any path of the resulting decision diagram that represents a solution in violation of a dualized constraint will have cost greater than $z^* + \Delta$.*

Proof. Suppose I indexes the dualized constraints. Since these are precisely the constraints that contain y_0, Lemma 7 implies that they can be ignored when

computing equivalency ranges. Now let p be any path in the decision diagram for which $\bar{x} = x(p)$ violates one or more constraints indexed by I. Add the term My_0 to the objective function of the LP in (4.4), which defines the cost of p. For sufficiently large M, any basic solution with positive y_0 of this LP will have value greater than $z^* + \Delta$. The cost of p therefore exceeds $z^* + \Delta$. □

Naturally, no more constraints should be dualized than necessary to obtain a decision diagram of reasonable size, because dualization results in a larger number of spurious solutions that must be discarded. In addition, nodes with the same RHS state (due to dualization) should not be identified unless necessary, because identification of nodes results in smaller length states and therefore a stronger bounding test for discarding undesirable solutions.

4.7.5 Bottom-Up Processing

A sound diagram can generally be simplified in a bottom-up pass, because at this point an entire diagram is at hand. The first step is to obtain a reduced diagram, using well-known methods [Bry86; Ber+16]. This is followed by arc deletion and then by contraction.

Arc deletion is more effective than in the top-down pass, because the bounding test at a node is stronger when we know all paths to the terminus as well as all paths from the root. We associate a top-down state $(\Delta b^{\downarrow}, v^{\downarrow})$ and a bottom-up states $(\Delta b^{\uparrow}, v^{\uparrow})$ with each node u. The top-down RHS state $\Delta b^{\downarrow}$ reflects how much the RHS must be changed due to variables fixed along paths from r to u, while the bottom-up RHS state reflects how variables are fixed along paths from u to t. We compute $(\Delta b^{\downarrow}, v^{\downarrow})$ recursively in a top-down fashion as follows. At r, we have $(\Delta b^{\downarrow}, v^{\downarrow}) = (0, 0)$. Let δ_j^q for $q \in Q$ be the labels of arcs coming into u, and let $(\Delta b^{q\downarrow}, v_q^{\downarrow})$ be the corresponding states of nodes from which the arcs come. Then the state $(\Delta b^{\downarrow}, v^{\downarrow})$ at node u is given by

$$\Delta b_i^{\downarrow} = \min_{q \in Q} \left\{ \Delta b_i^{q\downarrow} - A_{ij}\delta_q \right\}, \text{ all } i; \quad v^{\downarrow} = \min_{q \in Q} \left\{ v_q^{\downarrow} + c_j \delta_q \right\}$$

To check whether an arc a from node $u \in U_j$ to node u' can be deleted, let $(\Delta b^{\downarrow}, v^{\downarrow})$ be the top-down state at node u, and $(\Delta b^{\uparrow}, v^{\uparrow})$ the bottom-up state at node u'. Let δ_j be the label of a. Solve an LP to compute the bound

$$\text{LP} = \min \left\{ dy \;\middle|\; By \geq b + \Delta b^{\downarrow} - A_j \delta_j + \Delta b^{\uparrow},\ y \geq 0 \right\} \tag{4.17}$$

Then arc a can be deleted if it passes the bounding test

$$\text{LP} + v^{\downarrow} + c_j \delta_j + v^{\uparrow} > z^* + \delta \tag{4.18}$$

because, in this case, arc a can be part of no Δ-optimal solution.

As an example, Fig. 4.6(a) shows the sound diagram generated as above for problem (4.9) with $\Delta = 5$. The top-down and bottom-up state at each node are indicated.

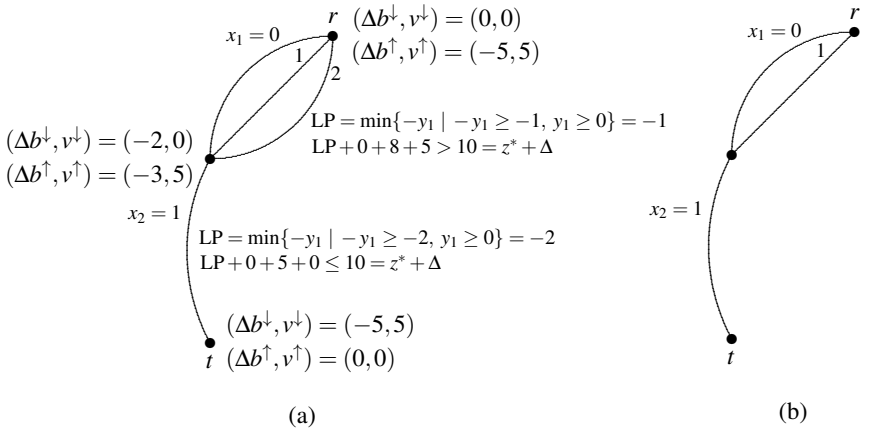


Figure 4.6: (a) Sound diagram for problem (4.9) with $\Delta = 5$, showing top-down and bottom up states at each node, and the bounding test for two of the arcs. (b) Sound diagram after arc deletion.

The arc with label $x_1 = 2$ passes the bounding test (4.18) and can be deleted. The remaining arcs fail the test. The resulting sound diagram appears in Fig. 4.6(b).

Theorem 7. *Soundness is preserved by deleting arcs that fail test (4.18).*

Proof. Let a be an arc that passes test (4.18), where a connects node $u \in U_j$ with node $u' \in U_{j+1}$. It suffices to show that any path p that contains a has cost greater than $z^* + \Delta$. Let $(\Delta b^{\downarrow}, v^{\downarrow})$ be the top-down state at u, and $(\Delta b^{\uparrow}, v^{\uparrow})$ the bottom-up state at u'. Let $\delta = x(p)$, so that $c\delta$ is the length of p. The cost of p is

$$z^*(x(p)) = c\delta + \min\left\{dy \;\middle|\; By \geq b - A\delta,\ y \geq 0\right\} \tag{4.19}$$

By definition of the states, $v^{\downarrow} + c_j\delta_j + v^{\uparrow} \leq c\delta$ and $b + \Delta b^{\downarrow} - A_j\delta_j + \Delta b^{\uparrow} \leq b - A\delta$. This and (4.19) imply that $z^*(x(p)) \geq v^{\downarrow} + c_j\delta_j + v^{\uparrow} + \mathrm{LP} > z^* + \Delta$, where the second inequality follows from the fact that a passes test (4.18). Path p therefore has cost greater than $z^* + \Delta$. □

Arc contraction is another device that may simplify a sound diagram, but it differs in that it introduces long arcs into the diagram. An arc a is a *long arc* when it skips layers, meaning that it runs from a node in U_j to node in U_k with $j + 1 < k$. A long arc represents multiple variable assignments, namely assignments of the form $(x_j, x_{j+1}, \ldots, x_{k-1}) = (x_j(a), \delta_{j+1}, \ldots, \delta_{k-1})$ for all $(\delta_{j+1}, \ldots, \delta_{k-1}) \in S_{j+1} \times \cdots \times S_{k-1}$. Other types of long arcs occur in *zero-suppressed* and *one-suppressed* diagrams, and the contraction procedure given here can be modified for these types of diagrams.

Because arc contraction can be applied repeatedly, it must be defined for a diagram that may already have long arcs. Suppose that all the arcs leaving node u lead to the same node u', and let $\ell(a)$ be the length of arc a. The arcs $\{a_q \mid q \in Q\}$ leaving u are *contracted* by shrinking them to a point, which means that any arc a from a node

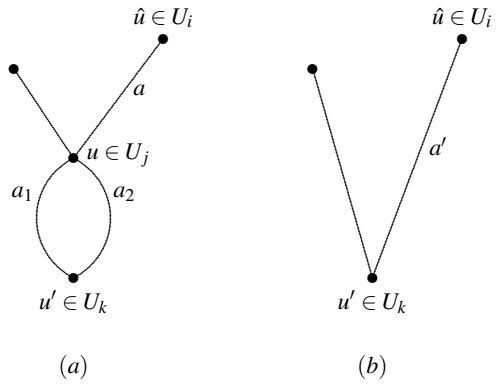


Figure 4.7: Contraction of arcs a_1 and a_2. (a) Before contraction. (b) After contraction.

$\hat{u} \in U_i$ to u becomes a longer arc a' from $\hat{u}$ to u'. The longer arc a' has the same label $x_i(a)$ but has length $\ell(a) + \min_{q \in Q}\{\ell(a_q)\}$. The node u and the arcs leaving it are deleted (Fig. 4.7).

Arc contraction can introduce additional paths to the decision diagram, but it preserves soundness when these paths have cost greater than $z^* + \Delta$. To assess their cost, we define top-down and bottom-up states essentially as before for nodes u and u' and define LP as in (4.17). The top-down states are computed recursively as follows. Let $\{a_q \mid q \in Q\}$ be the arcs coming into node u. Let $u_q \in U_{j(u_q)}$ be the node from which arc a_q originates, and let $(\Delta b^{\downarrow}(u_q), v^{\downarrow}(u_q))$ be the top-down state of u. Let $\delta_{qj} = x_j(a_q)$. Then the root node has state $(0, 0)$, and the state $(\Delta b^{\downarrow}, v^{\downarrow})$ of node u is given by

$$\Delta b_i^{\downarrow} = \min_{q \in Q} \left\{ \Delta b_i^{\downarrow}(u_q) - \alpha_{iq} \right\}, \text{ all } i; \quad v^{\downarrow} = \min_{q \in Q} \left\{ v^{\downarrow}(u_q) + \ell(a_q) \right\}$$

The reduction α_{iq} in the RHS is defined by

$$\alpha_{iq} = A_{ij(u_q)}\delta_{qj} + \max_{\delta' \in S'} \left\{ \sum_{j'=j(u_q)+1}^{j-1} A_{ij'}\delta'_{j'} \right\}$$

where $S' = S_{j(u_q)+1} \times \cdots \times S_{j-1}$. The bottom-up states are similarly defined.

Now assume that all the arcs leaving $u \in U_j$ arrive at $u' \in U_k$. To check whether these arcs can be contracted, we let $\bar{S}_j$ be the set of values in S_j that do not occur as labels on the arcs leaving u. Let $(b^{\downarrow}, v^{\downarrow})$ be the top-down state at u and $(b^{\uparrow}, v^{\uparrow})$ the bottom-up state at u'. For each $\delta_j \in \bar{S}_j$, we solve the LP problem

$$\mathrm{LP}(\delta_j) = \min \left\{ dy \;\middle|\; By \geq b + \Delta b^{\downarrow} - \alpha(\delta_j) + \Delta b^{\uparrow},\ y \geq 0 \right\}$$

The RHS reduction $\alpha(\delta_j)$ is given by

$$\alpha_i(\delta_j) = A_{ij}\delta_j + \max_{\delta' \in S'} \left\{ \sum_{j'=j+1}^{k-1} A_{ij'}\delta'_{j'} \right\}, \text{ all } i$$

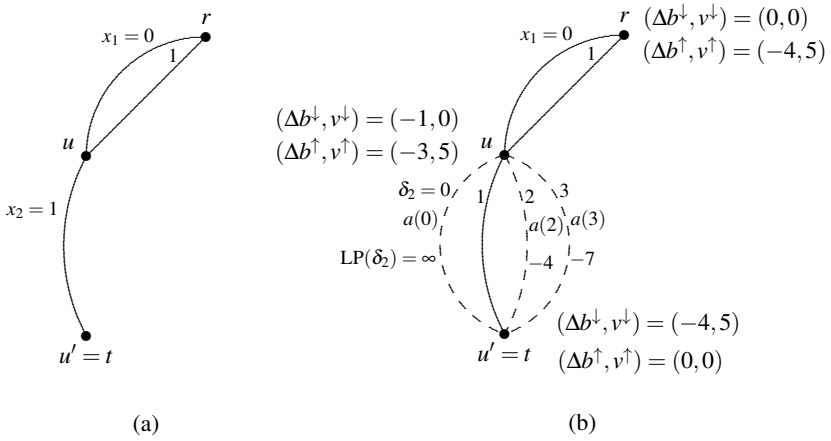


Figure 4.8: (a) Sound diagram for problem (4.9) with $\Delta = 4$. (b) Contraction test for 3 arcs (dashed lines). Only arc $a(0)$ passes the test, and there is no contraction.

where $S' = S_{j+1} \times \cdots \times S_{k-1}$. Let $a(\delta_j)$ be the missing arc from u to u' with label δ_j. The arcs from u to u' can be contracted if the following holds:

$$\mathrm{LP}(\delta_j) + v^{\downarrow} + \ell(a(\delta_j)) + v^{\uparrow} > z^* + \delta, \text{ all } \delta_j \in \bar{S}_j \tag{4.20}$$

The condition is vacuously satisfied if all possible arcs from u are present, because in this case $\bar{S}_j$ is empty.

The proof of the following is similar to the proof of Theorem 7.

Theorem 8. *Soundness is preserved by contracting the arcs that connect two nodes u and u', provided all the arcs leaving u arrive at u', and (4.20) is satisfied.*

As an example, consider the sound diagram in Fig. 4.6(b) for $\Delta = 5$, reproduced in Fig. 4.8(a). We wish to determine whether the arc from u to $u' = t$ can be contracted. Figure 4.8(b) shows the same diagram with the missing arcs from u to u' added as dashed lines. These are the arcs $a(0), a(2), a(3)$ with labels in $\bar{S}_2 = \{0, 2, 3\}$. The states are shown, as well as $\mathrm{LP}(\delta_2)$ for each $\delta_2 \in \bar{S}_2$. Arc $a(0)$ clearly satisfies the inequality in (4.20), because $\mathrm{LP}(0) = \infty$. However, arc $a(2)$ violates the inequality, because

$$\mathrm{LP}(2) + v^{\downarrow} + \ell(a(2)) + v^{\uparrow} = -4 + 0 + 10 + 0 = 6 \leq 10 = z^* + \Delta$$

Arc $a(3)$ likewise violates the inequality, and the existing arc from u to u' cannot be contracted.

4.7.6 Using the MIP Solver Solution Pool

So far, we have presented a method for performing post-optimality analysis for MILP problems that only requires the optimal objective value z^* to be given (in addition

to the problem data), which requires us to generate the set of near-optimal solutions. However, some modern MIP solvers like CPLEX and Gurobi can generate an exhaustive list of the set of near-optimal solutions [IBM19; Opt19] via the *solution pool* feature using algorithms like the one-tree method of [Dan+07]. In these cases, multiple solutions are often generated only with respect to the discrete variables, i.e., two solutions which have the same value for the discrete variables but different values for the continuous variables are considered equivalent. We can leverage the solution pool feature by using the *solver* to generate the set of near-optimal solutions. This allows us to focus only on the task of representing the set of near-optimal solutions as compactly as possible.

We associate each feasible assignment to the integer variables with its best attainable objective value. This can now be treated as a set of feasible solutions to an integer nonlinear programming problem (in particular, the objective is piecewise-linear and convex) together with its objective value, which is closer to the setting of [HS17]. The arc weights are set according to the canonical arc cost calculation of [Hoo13], which in general results in arc weights that are *non-separable*, i.e., two arcs with the same label in the same layer may have different associated weights.

More formally, recall that we have a MILP problem

$$\min\{cx + dy \mid Ax + By \geq b,\ x \in S,\ y \geq 0\} \tag{M}$$

with optimal value z^*, and we pick some tolerance $\Delta \geq 0$. Note that by first fixing the discrete variables, (M) can equivalently be written as

$$(\text{M}) = \min_x \left\{ cx + \min_y \{dy : By \geq b - Ax, y \geq 0\} \;\middle|\; x \in S \right\} = \min_x \{\Lambda(x) \mid x \in S\}$$

where

$$\begin{aligned}\Lambda(x) &= cx + \min_y \{dy : By \geq b - Ax, y \geq 0\} \\ &= cx + \max_\lambda \{\lambda^T (b - Ax) : B^T \lambda \leq d, \lambda \geq 0\}\end{aligned}$$

and therefore $\Lambda(x)$ is a convex piecewise-linear function (since it is the sum of a linear function and a maximum of affine functions). Thus (M) can equivalently be written as a *pure* integer nonlinear program with a convex piecewise-linear objective. Since this formulation only has integer variables, the theory from [HS17] extends almost directly with three major differences:

1. In [HS17] the objective function is linear and, more importantly, *separable*, which enables the compilation of *separable* (sound) decision diagrams (i.e., arcs with the same label in the same layer have the same cost). In our case, the objective function is no longer separable and in general we can only compile *non-separable* (sound) decision diagrams (i.e., arcs with the same label in the same label may have different costs).

2. In [HS17], the cost of an arc in layer j with label a directly corresponds to the coefficient of x_j in the objective, which allows them to do some limited sensitivity analysis. In our case, we determine the arc costs *canonically* based on the path weights according to [Hoo13], and therefore we lose the direct connection between arc costs and objective function coefficients.

3. In [HS17], the cost of any path added as a result of sound reduction is also equal to the cost of its corresponding solution. In our case, the cost of paths added as a result of sound reduction are based on canonical arc costs of the original set of paths, and therefore have no reltion with the cost of the corresponding solutions of those added paths. However, soundness is still maintained, since sound reduction only adds Δ-costly paths which can be filtered out based on cost. Nevertheless, we can no longer guarantee minimality of the sound diagram for $\mathrm{M}(\Delta)$.

we now describe our method in more detail. First, we ask the MIP solver, typically using the *Solution Pool* feature, to generate $\mathrm{M}(\Delta)$ (i.e., the set of all Δ-optimal solutions of (M)). The MIP solver then returns us these solutions in the form of tuples $(x, \Lambda(x))$, where x is feasible to (MIP) and $\Lambda(x)$ is defined as before and satisfies $\Lambda(x) \leq z^* + \Delta$.

Next we compile a decision diagram that exactly represents $\mathrm{M}(\Delta)$. This can be done using the framework of [Hoo13]. Namely, we first create a tree containing precisely the paths in $\mathrm{M}(\Delta)$ and we assign the cost $\Lambda(x)$ to the arc in the last layer of each corresponding path. Then we modify the arc costs layer by layer according to the following procedure:

Algorithm 3 Converting arc costs to canonical arc costs

for $i = n-1, n-2, ..., 2$ **do**
  for all node u on layer i **do**
    Let $c_{\min} = \min_{u' \in U_{\text{out}}}\{c_{uu'}\}$ ▷ U_{out} is the set of child nodes of u
    for all $u' \in U_{\text{out}}$ **do**
      Let $c_{uu'} \leftarrow c_{uu'} - c_{\min}$
    end for
    for all $u' \in U_{\text{in}}$ **do** ▷ U_{in} is the set of parent nodes of u
      Let $c_{u'u} \leftarrow c_{u'u} + c_{\min}$
    end for
  end for
end for

It is known that the decision diagram created in this way is the smallest possible decision diagram that exactly represents $\mathrm{M}(\Delta)$, though it obviously may not be the smallest possible sound decision diagram for $\mathrm{M}(\Delta)$.

Finally, we apply sound reduction of nodes as many times as possible. The LCDS scheme from [HS17] works with one major difference: since the arc costs are not separable, we must check that both the label *and* the weight are the same before recursing. The recursion is therefore stated as follows:

$$\mathrm{LCDS}_{n+1}(u,v) = +\infty$$

$$\mathrm{LCDS}_j(u,v) = \min_{a=(u,u_+)} w(a) + \begin{cases} \mathrm{LCDS}_{j+1}(u_+,v_+) & \exists a' = (v,v_+) \text{ s.t. } \begin{matrix} \ell(a) = \ell(a') \\ w(a) = w(a') \end{matrix} \\ w(u_+,t) & \text{otherwise} \end{cases}$$

where $\mathrm{LCDS}_j(u,v)$ is the minimum cost of a suffix of u that is not a suffix of v, $w(a)$ and $\ell(a)$ are the weight and label of arc a, (u,u_+) is an outgoing arc of u, and (v,v_+) is an outgoing arc of v.

4.8 Performing Post-Optimality Analysis

The most basic post-optimality task is to retrieve all feasible solutions whose cost is within a given distance of the optimal cost. That is, we wish to retrieve all δ-optimal solutions from a diagram D that is sound for $\mathrm{M}(\Delta)$ for a desired $\delta \in [0,\Delta]$. This is accomplished by enumerating the paths in a bottom-up pass while filtering out paths that have high cost. It is also easy to impose additional constraints on the solutions that we want to retrieve during enumeration since in bottom-up passes we have access to the whole path.

Another task the sound decision diagram supports is to determine the values that a given variable can take such that the resulting minimum cost is within δ of the optimum. We refer to this as the *δ-optimal domain* of the variable. To calculate the δ-optimal domain for a variable x_j, we need only scan the arcs leaving layer j and observe which ones are part of at least one path with cost at most $z^* + \Delta$; this can be done by precomputing the shortest r–u and u–t path for every node u.

4.9 Computational Experiments

Out of all the methods proposed, the only one that is scalable to larger instances is the method from Section 4.7.6 where we use the MIP solver to generate the Δ-optimal solutions; thus, we only present computational results for this method. We selected 30 MILP instances from the MIPLIB2017 benchmark which were classified as "easy" instances, had at least one continuous variable, and had the smallest number of discrete variables. Table 4.1 lists the MIPLIB instances used and some of their characteristics.

We used CPLEX 12.8.0 to generate the near-optimal solutions, where we used the Solution Pool feature [10] to exhaustively enumerate all solutions within $\widehat{\Delta}$ percent

InstanceName	BinVar	IntVar	CntVar	Constr	ObjVal
markshare_4_0	30	0	4	4	1.000
istanbul-no-cutoff	30	0	5252	20346	204.0817
fastxgemm-n2r6s0t2	48	0	736	5998	230.0000
neos5	53	0	10	63	15.0000
pk1	55	0	31	45	11.0000
neos-2978193-inde	64	0	20736	396	-2.3881
dano3_3	69	0	13804	3202	576.3446
pg5_34	100	0	2500	225	-14339.3535
pg	100	0	2600	125	-8674.3426
neos-1122047	100	0	5000	57791	161.0000
rmatr100-p10	100	0	7259	7260	423.0000
dano3_5	115	0	13758	3202	576.9249
assign1-5-8	130	0	26	161	212.0000
map10	146	0	164401	328818	-495.0000
map16715-04	146	0	164401	328818	-111.0000
mas74	150	0	1	13	11801.1857
mas76	150	0	1	12	40005.0540
n5-3	0	150	2400	1062	8105.0000
binkar10_1	170	0	2128	1026	6742.1999
timtab1	77	94	226	171	764772.0000
cost266-UUE	171	0	3990	1446	25148940.5600
mad	200	0	20	51	0.0268
graphdraw-domain	180	20	54	865	19686.0000
rmatr200-p5	200	0	37616	37617	4521.0000
supportcase12	0	200	799416	166781	-7559.5331
mik-250-20-75-4	75	175	20	195	-52301.0000
exp-1-500-5-5	250	0	740	550	65887.0000
ran14x18-disj-8	252	0	252	447	3712.0000
n9-3	0	252	7392	2364	14409.0000
neos-3046615-murg	240	16	18	498	1600.0000

Table 4.1: MIPLIB Instances Used

of the optimal value (i.e., $\Delta = \widehat{\Delta}|z^*|$), up to a maximum of 2,000,000,000 unique solutions. We imposed a time limit of 6 hours on the solution generation procedure and used a server with 16 cores and 125GB of memory. We conducted the experiment in three phases:

1. For the 30 instances, generate a sound decision diagram for M(0) (i.e., only the optimal solutions). Most problems either have a single optimal solution, in which case the sound reduction is trivial. But for some problems, CPLEX cannot generate all optimal solutions within the 6 hour time limit. We found 7 such models and we exclude them from further testing. The results are shown in Table 4.2, Figure 4.9, and Figure 4.10.

2. For the other 23 instances, generate a sound diagram for M($0.01|z^*|$). Divide the instances into "easy" if CPLEX finishes within the time limit, "hard" otherwise. We found 13 of the models were easy and the other 10 were hard. The results are shown in Table 4.3, Figure 4.11, and Figure 4.12.

3. For the easy instances, generate a sound diagram for M($0.1|z^*|$). For the hard instances, generate a sound diagram for M($0.001|z^*|$). The results for easy instances are shown in Table 4.4, Figure 4.13, and Figure 4.14. The results for hard instances are shown in Table 4.5, Figure 4.15, and Figure 4.16.

In order to clearly see the effect of sound reduction specifically, we do not incorporate any conditions for identifying equivalent states or arc-deletion/arc-contraction via bottom-up processing as part of our testing.

For each phase, there are two result graphs, where the instances are ordered according to the number of near-optimal solutions generated by CPLEX and the vertical axes are shown in log-scale. One graph shows the time taken (in seconds) to compile the sound decision diagram for each instance, which consists of four parts:

- **MIPSolveTime**: time to generate the list of Δ-optimal solution using CPLEX, which we denote $V_{\mathrm{M}}(\Delta)$
- **TreeCompTime**: time to generate a branching-tree representation of $V_{\mathrm{M}}(\Delta)$
- **ExactCompTime**: time to reduce the branching-tree representation to an exact reduced decision diagram representation of $V_{\mathrm{M}}(\Delta)$
- **SoundCompTime**: time to apply sound reduction as much as possible to the exact reduced decision diagram to obtain a sound decision diagram for $V_{\mathrm{M}}(\Delta)$

The other graph shows the size of the resulting representations of the set of near-optimal solutions in terms of the number of nodes:

- **TreeNumNodes**: size of branching-tree representation

- **ExactNumNodes**: size of exact reduced decision diagram representation
- **SoundNumNodes**: size of sound decision diagram representation

The computational results show that compiling the list of solutions into a sound decision diagram almost never takes more time than generating the solutions, and often takes much less time. On the other hand, the size of the sound decision diagram representation is often not much smaller than the branching-tree representation; however, we note that the number of paths encoded in the diagram is quite small for most of the instances.

4.10 Conclusion

We presented a novel decision-diagram based approach for post-optimality analysis of MILP problems by extending the framework of [HS17]. To incorporate continuous variables, we considered two broad approaches: explicit representation, in which we discretize the continuous part using basic feasible solutions, and implicit representation, where the continuous part is implicitly enforced by keeping track of the RHS and partial cost within the node states. We then presented an extension of sound reduction to our setting, sufficient conditions for identifying equivalent node states, and arc deletion/contraction strategies for further reducing the diagram. Finally, we performed computational tests to see how our framework works for realistic MILP problems, and showed that a sound decision diagram representation of the set of near-optimal solutions can be obtained without significantly more time compared to the time required to generate the solutions.

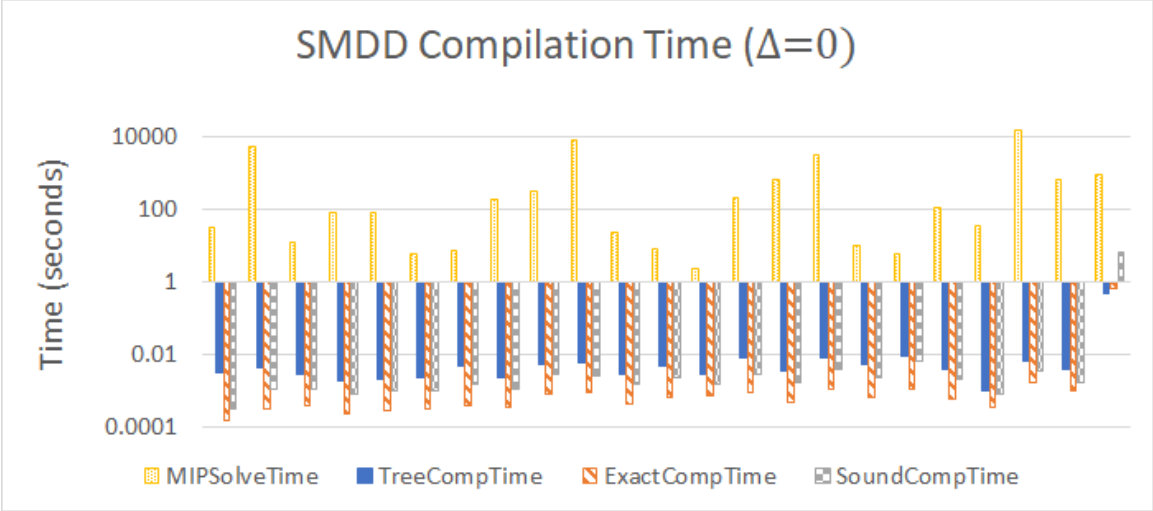


Figure 4.9: Time to Compile Sound Decision Diagram with $\widehat{\Delta} = 0$

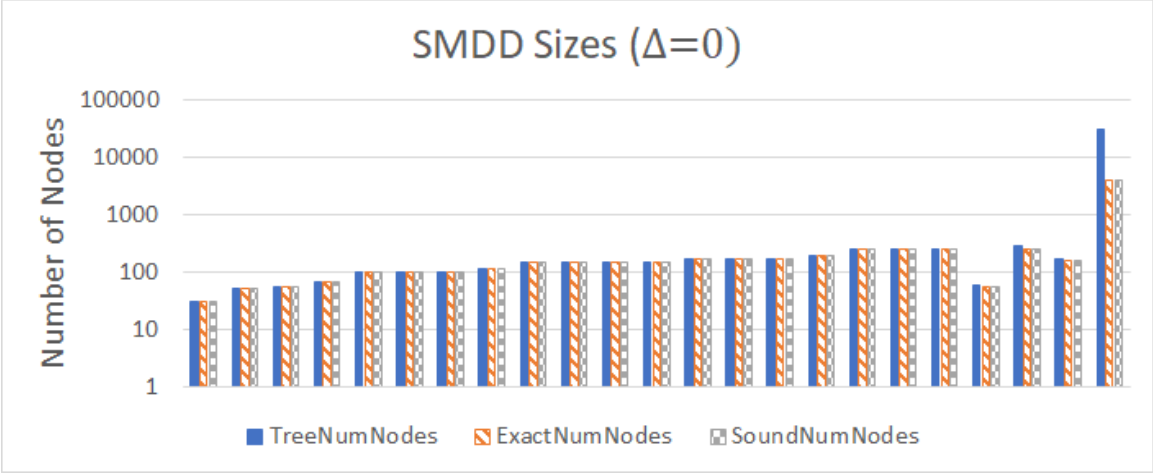


Figure 4.10: Size of Decision Diagram Representations with $\widehat{\Delta} = 0$

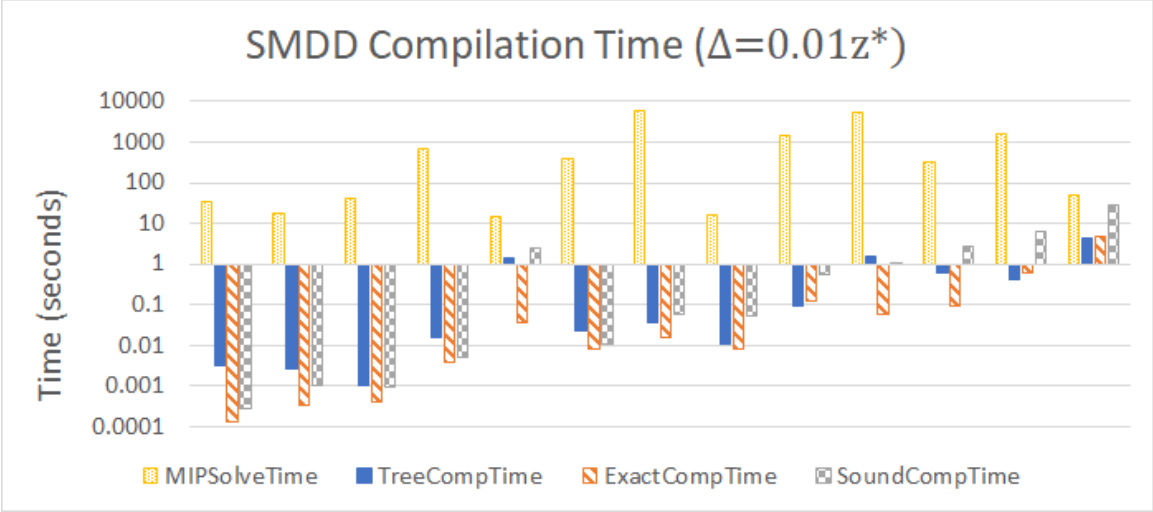


Figure 4.11: Time to Compile Sound Decision Diagram with $\widehat{\Delta} = 0.01$

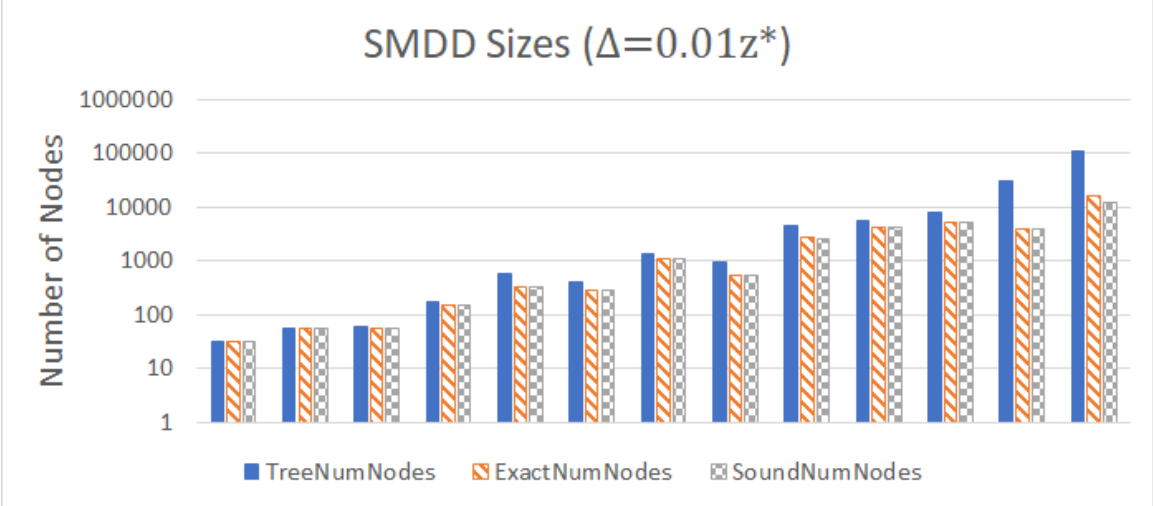


Figure 4.12: Size of Decision Diagram Representations with $\widehat{\Delta} = 0.01$

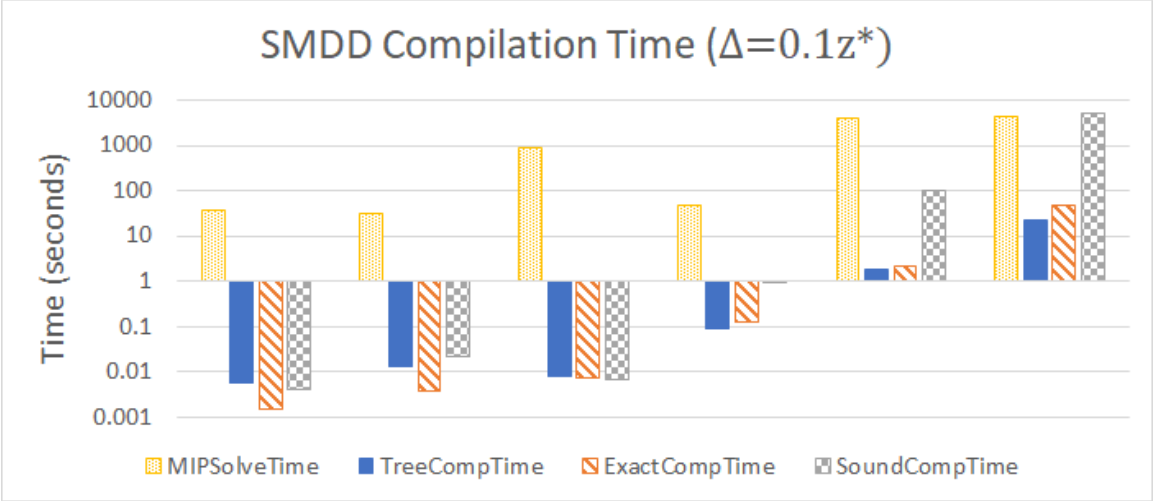


Figure 4.13: Time to Compile Sound Decision Diagram with $\widehat{\Delta} = 0.1$

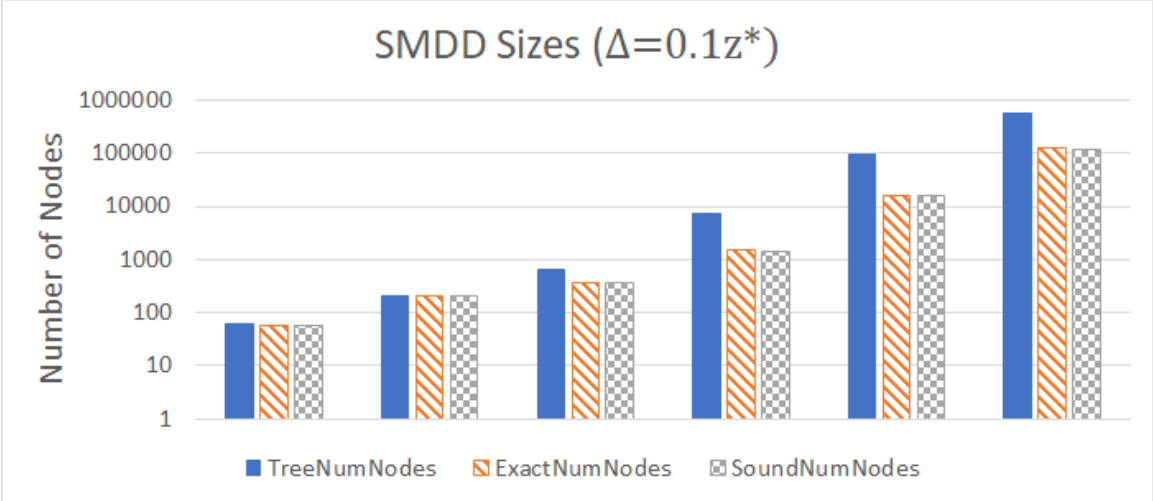


Figure 4.14: Size of Decision Diagram Representations with $\widehat{\Delta} = 0.1$

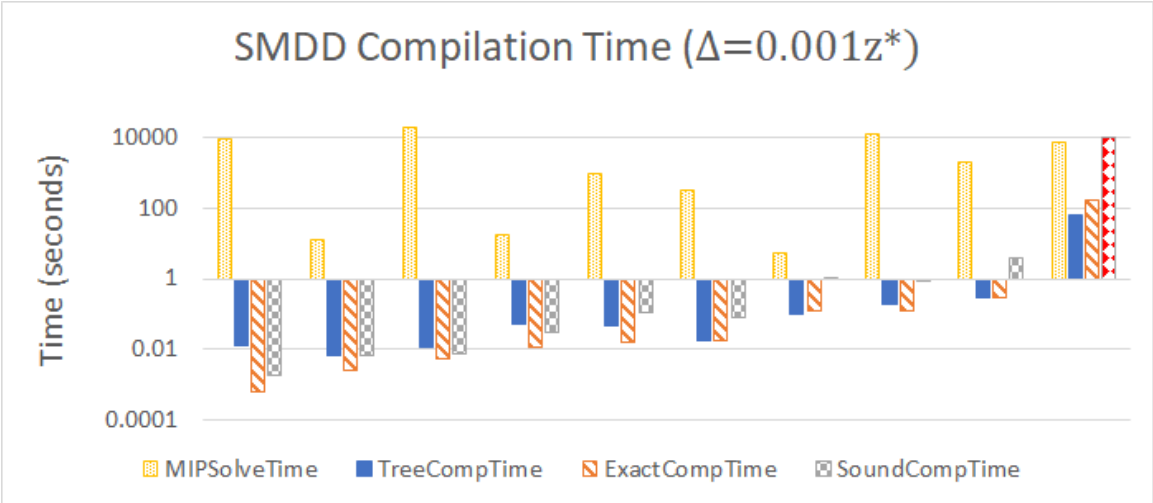


Figure 4.15: Time to Compile Sound Decision Diagram with $\widehat{\Delta} = 0.001$; Here the rightmost instance neos5 ran out of memory during the sound reduction phase

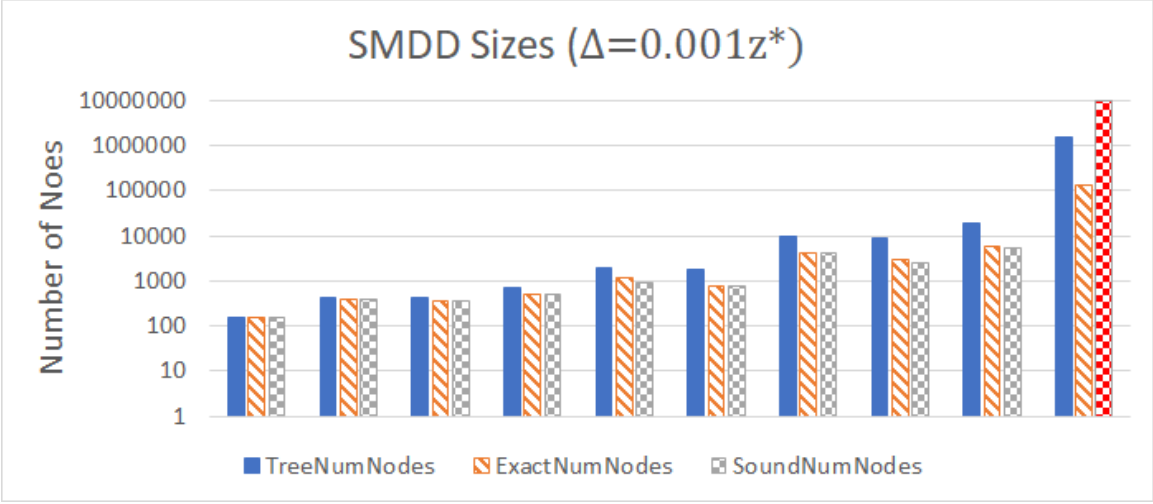


Figure 4.16: Size of Decision Diagram Representations with $\widehat{\Delta} = 0.001$; Here the rightmost instance neos5 ran out of memory during the sound reduction phase

instanceName	Paths	MIPSolveTime	TreeCompTime	ExactCompTime	SoundCompTime	TreeNodes	ExactNodes	SoundNodes
istanbul-no-cutoff	1	31.85802698	0.003074884	0.000142097	0.000311136	31	31	31
neos5	1	5378.256864	0.004070997	0.000309944	0.001055956	54	54	54
pk1	1	11.6641171	0.002839088	0.000386953	0.001108885	56	56	56
dano3_3	1	80.39648795	0.001842022	0.000232935	0.000762939	70	70	70
pg5_34	1	78.64459181	0.002030849	0.000285149	0.000983	101	101	101
pg	1	6.020258904	0.002194881	0.000298023	0.000962973	101	101	101
rmatr100-p10	1	7.007212162	0.004667044	0.000371218	0.001480103	101	101	101
dano3_5	1	187.3480401	0.002327919	0.000328064	0.001061916	116	116	116
map10	1	319.2726209	0.005108118	0.00081706	0.002771854	147	147	147
mas74	1	7774.87802	0.005494833	0.000897169	0.002518177	151	151	151
mas76	1	22.6823101	0.002924204	0.000413179	0.001446962	151	151	151
n5-3	1	8.250102997	0.004635096	0.000672817	0.00219512	151	151	151
binkar10_1	1	2.16109395	0.002701998	0.000696182	0.001489878	171	171	171
timtab1	1	195.8134489	0.007610083	0.000916958	0.002885103	172	172	172
cost266-UUE	1	685.737154	0.003551006	0.000454187	0.001650095	172	172	172
rmatr200-p5	1	3280.212475	0.008337021	0.001076937	0.003909111	201	201	201
mik-250-20-75-4	1	9.600122929	0.005438089	0.000653028	0.002276897	251	251	251
exp-1-500-5-5	1	5.81605792	0.008642912	0.001107931	0.006142855	251	251	251
ran14x18-disj-8	1	104.408813	0.003838062	0.000564098	0.002034903	253	253	253
markshare_4_0	2	34.39960122	0.001003027	0.000347853	0.000797033	60	56	56
n9-3	2	15270.23117	0.006546974	0.001577854	0.00335288	280	254	254
map16715-04	3	629.9044662	0.003711939	0.000957012	0.001643896	173	155	155
fastxgemm-n2r6s0t2	1150	891.238754	0.450829029	0.617268085	6.449658155	30570	3939	3939
neos-2978193-inde	0	21625.24041	NA	NA	NA	NA	NA	NA
neos-1122047	0	21619.81191	NA	NA	NA	NA	NA	NA
assign1-5-8	0	21616.65221	NA	NA	NA	NA	NA	NA
mad	0	22227.16135	NA	NA	NA	NA	NA	NA
graphdraw-domain	0	21601.17329	NA	NA	NA	NA	NA	NA
supportcase12	0	21677.52053	NA	NA	NA	NA	NA	NA
neos-3046615-murg	0	21621.34891	NA	NA	NA	NA	NA	NA

Table 4.2: Table of Results for $\widehat{\Delta} = 0$

instanceName	Paths	MIPSolveTime	TreeCompTime	ExactCompTime	SoundCompTime	TreeNodes	ExactNodes	SoundNodes
istanbul-no-cutoff	1	35.40369701	0.003133059	0.000123978	0.000274897	31	31	31
pk1	1	17.85193014	0.002689123	0.000334978	0.001025915	56	56	56
markshare_4_0	2	39.36630988	0.001055956	0.000391006	0.000887871	60	56	56
map16715-04	3	674.6803269	0.015972137	0.003809929	0.005210161	173	155	155
rmatr100-p10	7	14.70442414	1.430439949	0.037547112	2.379997015	565	336	336
map10	7	409.2945349	0.022800207	0.008388042	0.011033058	399	280	280
rmatr200-p5	9	6096.051809	0.035429001	0.015745163	0.058043003	1393	1077	1077
n5-3	16	15.91614079	0.010217905	0.008308887	0.052429199	924	552	538
ran14x18-disj-8	24	1445.110785	0.094877005	0.118139029	0.570495129	4711	2693	2663
mas76	50	5331.386539	1.507452965	0.05841589	1.053275108	5666	4155	4155
timtab1	329	331.3341088	0.609101772	0.089138031	2.679023027	8014	5337	5089
fastxgemm-n2r6s0t2	1152	1576.072292	0.431755066	0.607537031	6.393348932	30616	3932	3932
exp-1-500-5-5	1337	48.5034771	4.461433887	4.769818068	26.996418	112611	15822	12578
neos5	0	21629.97203	NA	NA	NA	NA	NA	NA
dano3_3	0	21668.24568	NA	NA	NA	NA	NA	NA
pg5_34	0	21638.73401	NA	NA	NA	NA	NA	NA
pg	0	21639.97958	NA	NA	NA	NA	NA	NA
dano3_5	0	21642.61049	NA	NA	NA	NA	NA	NA
mas74	0	21626.04941	NA	NA	NA	NA	NA	NA
binkar10_1	0	21676.41203	NA	NA	NA	NA	NA	NA
cost266-UUE	0	21644.49908	NA	NA	NA	NA	NA	NA
mik-250-20-75-4	0	21783.75266	NA	NA	NA	NA	NA	NA
n9-3	0	21619.53444	NA	NA	NA	NA	NA	NA

Table 4.3: Table of Results for $\widehat{\Delta} = 0.01$

instanceName	Paths	MIPSolveTime	TreeCompTime	ExactCompTime	SoundCompTime	TreeNodes	ExactNodes	SoundNodes
markshare_4_0	2	36.6132381	0.005554914	0.001462936	0.004102945	60	56	56
pk1	4	32.45357299	0.013266087	0.003823996	0.021386147	213	199	199
map16715-04	13	848.581403	0.008041143	0.007266998	0.006934881	652	352	352
istanbul-no-cutoff	776	49.07613301	0.088376999	0.122344017	0.962555885	7397	1567	1413
fastxgemm-n2r6s0t2	3456	4117.874002	1.775875092	2.112369061	103.6205142	92336	15449	15449
rmatr100-p10	14830	4152.188039	22.6640532	47.60716796	5187.308354	571303	130592	120574
map10	0	21651.28558	NA	NA	NA	NA	NA	NA
mas76	0	21674.54793	NA	NA	NA	NA	NA	NA
n5-3	0	21696.26481	NA	NA	NA	NA	NA	NA
timtab1	0	21652.62161	NA	NA	NA	NA	NA	NA
rmatr200-p5	0	21626.44729	NA	NA	NA	NA	NA	NA
exp-1-500-5-5	0	21744.88075	NA	NA	NA	NA	NA	NA
ran14x18-disj-8	0	21680.50462	NA	NA	NA	NA	NA	NA

Table 4.4: Table of Results for $\widehat{\Delta} = 0.1$

instanceName	Paths	MIPSolveTime	TreeCompTime	ExactCompTime	SoundCompTime	TreeNodes	ExactNodes	SoundNodes
mas74	1	9044.992402	0.012527943	0.000584126	0.001803875	151	151	151
mik-250-20-75-4	4	13.04642081	0.006618023	0.002384186	0.006876945	441	401	401
n9-3	4	20551.30065	0.01078701	0.005583048	0.007652998	441	361	361
pg	11	18.09140992	0.048659086	0.011086941	0.031275988	708	509	509
cost266-UUE	39	945.7149749	0.046290159	0.016383171	0.111884832	1897	1209	931
dano3_3	73	312.793139	0.018094063	0.018083096	0.076301098	1743	799	751
binkar10_1	90	5.13025403	0.095981836	0.127208948	1.003798962	10147	4206	4089
pg5_34	239	13136.73549	0.179249048	0.12211585	0.939352036	8740	2973	2625
dano3_5	631	2014.288355	0.280895948	0.291321993	4.04116106	19646	5741	5456
neos5	56466	7646.846363	61.62535405	170.1040699	NA	1559206	132936	NA

Table 4.5: Table of Results for $\widehat{\Delta} = 0.001$

Chapter 5

Conclusion

In this dissertation, we considered three projects aimed at developing techniques and methodologies to handle larger, more complex discrete optimization models. In the first project, we utilized logic-based Benders decomposition to effectively solve practically sized instances of the home healthcare problem, and also gained insights into the differences between different implementations of the decomposition scheme. In the second project, we proposed a simple generic framework for solving robust scheduling problems with combinatorial uncertainty sets, and tested the framework on robust job shop scheduling as a proof-of-concept. In the third project, we proposed decision diagrams as a method to enable postoptimality analysis of MILP problems, focusing on how to extend the sound reduction framework to models with continuous variables.

Each work has some interesting future research directions. For the home healthcare project, extending the model to handle visits requiring multiple aides is nontrivial and would bring the model closer to practice. Further, the home health care problem provides a challenging practical benchmark for other decomposition methods (e.g., branch-cut-and-price); investigating them may provide further insight into the advantages and disadvantages of using one decomposition technique over another. For the robust scheduling problem, a more systematic investigation of combinatorial uncertainty sets used in practice, as well as tests on additional scheduling models would aid us in further determining the key factors involved in robust scheduling models. For the work on postoptimality for MILPs, incorporating techniques from global optimization to potentially inspire a represenation scheme using interval labels would be promising. It is also worth investigating the complexity and fundamental limits of compactly representing sets of (near-)optimal solutions, perhaps using tools from the knowledge compilation literature.

Overall, incorporating the structure of the problem into the solution scheme is often the key step for efficiently solving large, complex discrete optimization models. That structure may be incorporated algorithmically, as with logic-based Benders decomposition and scenario generation, or graphically, as with decision diagrams. The

challenge with decomposition methods then is to strive for generic applicability while still capturing problem specific features. Investigating decomposition methods that strike the right balance will aid tremendously in pushing the boundaries of the scope and range of problems that can be tackled by discrete optimization, both in theory and in practice.

Bibliography

[08] *Release notes for SCIP 1.1*. https://scip.zib.de/doc-6.0.1/html/RN11.php. 2008.

[10] *IBM Support: Using CPLEX to examine alternate optimal solutions*. https://www-01.ibm.com/support/docview.wss?uid=swg21399929. 2010.

[11] *Release notes for SCIP 2.0*. https://scip.zib.de/doc-6.0.1/html/RN20.php. 2011.

[ABS97] David Avis, David Bremner, and Raimund Seidel. "How good are convex hull algorithms?" In: *Computational Geometry* 7.5-6 (1997), pp. 265–301.

[Ach09] T. Achterberg. "SCIP: Solving constraint integer programs". In: *Mathematical Programming Computation* 1 (2009), pp. 1–41.

[ADP03] Sammani D Abdullahi, Martin E Dyer, and Les G Proll. "Listing vertices of simple polyhedra associated with dual LI (2) systems". In: *International Conference on Discrete Mathematics and Theoretical Computer Science*. Springer. 2003, pp. 89–96.

[AE11] Nasr Al-Hinai and T.Y. ElMekkawy. "Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm". In: *International Journal of Production Economics* 132.2 (2011), pp. 279–291.

[AF92] David Avis and Komei Fukuda. "A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra". In: *Discrete & Computational Geometry* 8.3 (1992), pp. 295–313.

[AFM02] Jérôme Amilhastre, Hélene Fargier, and Pierre Marquis. "Consistency restoration and explanations in dynamic CSPs–application to configuration". In: *Artificial Intelligence* 135.1-2 (2002), pp. 199–234.

[AHK08] Tobias Achterberg, Stefan Heinz, and Thorsten Koch. "Counting solutions of integer programs using unrestricted subtree detection". In: *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer. 2008, pp. 278–282.

[AHP10] Henrik Reif Andersen, Tarik Hadzic, and David Pisinger. "Interactive cost configuration over decision diagrams". In: *Journal of Artificial Intelligence Research* 37 (2010), pp. 99–139.

[Ald+03] Michel Aldanondo et al. "Mass customization and configuration: Requirement analysis and constraint based modeling propositions". In: *Integrated Computer-Aided Engineering* 10.2 (2003), pp. 177–189.

[All+13] H. Allaoua et al. "A matheuristic approach for solving a home health care problem". In: *Electronic Notes in Discrete Mathematics* 41 (2013), pp. 471–478.

[Ang+05] Alfredo Anglani et al. "Robust scheduling of parallel machines with sequence-dependent set-up costs". In: *European Journal of Operational Research* 161.3 (2005), pp. 704–720.

[AS97] R. J. Abumaizar and J.A. Svestka. "Rescheduling job shops under random disruptions". In: *International Journal of Production Research* 35.7 (1997), pp. 2065–2082.

[Avi00] David Avis. "A revised implementation of the reverse search vertex enumeration algorithm". In: *Polytopes–combinatorics and computation*. Springer. 2000, pp. 177–198.

[Bah+97] R Iris Bahar et al. "Algebric decision diagrams and their applications". In: *Formal methods in system design* 10.2-3 (1997), pp. 171–206.

[Bal61] Michel L Balinski. "An algorithm for finding all vertices of convex polyhedral sets". In: *Journal of the Society for Industrial and Applied Mathematics* 9.1 (1961), pp. 72–88.

[Bar+96] C Bradford Barber et al. "The quickhull algorithm for convex hulls". In: *ACM Transactions on Mathematical Software (TOMS)* 22.4 (1996), pp. 469–483.

[BE07] Markus Behle and Friedrich Eisenbrand. "0/1 vertex and facet enumeration with BDDs". In: *2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM. 2007, pp. 158–165.

[Bec10] J. C. Beck. "Checking-up on branch and check". In: *Principles and Practice of Constraint Programming (CP 2010)*. Ed. by D. Cohen. Vol. 6308. Lecture Notes in Computer Science. New York: Springer, 2010, pp. 84–98.

[BEN09] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust Optimization*. Princeton University Press, 2009.

[Ben62] J. F. Benders. "Partitioning procedures for solving mixed-variables programming problems". In: *Numerische Mathematik* 4 (1962), pp. 238–252.

[Ber+16] David Bergman et al. *Decision diagrams for optimization*. Vol. 1. Springer, 2016.

[Ber13] David Bergman. "New techniques for discrete optimization". PhD thesis. PhD thesis, Tepper School of Business, Carnegie Mellon University, 2013.

[BFM98] David Bremner, Komei Fukuda, and Ambros Marzetta. "Primal–dual methods for vertex and facet enumeration". In: *Discrete & Computational Geometry* 20.3 (1998), pp. 333–357.

[BL98] Michael R Bussieck and Marco E Lübbecke. "The vertex set of a 01-polytope is strongly P-enumerable". In: *Computational Geometry* 11.2 (1998), pp. 103–109.

[BLE07] Cyril Briand, H. Trung La, and Jacques Erschler. "A robust approach for the single machine scheduling problem". English. In: *Journal of Scheduling* 10.3 (2007), pp. 209–221.

[BN02] A. Ben-Tal and A. Nemirovski. "Robust optimization – methodology and applications". In: *Mathematical Programming* 92.3 (2002), pp. 453–480.

[Bor+09] Endre Boros et al. "Generating Vertices of Polyhedra and Related Problems of Monotone Generation". In: *CRM Proceedings and Lecture Notes*. Vol. 48. 2009.

[Bor+11] Endre Boros et al. "The negative cycles polyhedron and hardness of checking some polyhedral properties". In: *Annals of Operations Research* 188.1 (2011), pp. 63–76.

[Bow72] V Joseph Bowman Jr. "Sensitivity analysis in linear integer programming". In: *AIIE Transactions* 4.4 (1972), pp. 284–289.

[Bry86] R.E. Bryant. "Graph-Based Algorithms for Boolean Function Manipulation". In: *IEEE Transactions on Computers* C-35.8 (1986), pp. 677–691.

[BS03] Dimitris Bertsimas and Melvyn Sim. "Robust discrete optimization and network flows". English. In: *Mathematical Programming* 98.1-3 (2003), pp. 49–71.

[BS04] Dimitris Bertsimas and Melvyn Sim. "The Price of Robustness". English. In: *Operations Research* 52.1 (2004), pp. 35–53.

[BS93] Giorgio C. Buttazzo and John A. Stankovic. "RED: Robust Earliest Deadline Scheduling". In: *Proc. of 3rd International Workshop On Responsive Computing Systems*. 1993, pp. 100–111.

[CÇH15] A. Ciré, E. Çoban, and J. N. Hooker. "Logic-based Benders decomposition for planning and scheduling: A computational analysis". In: *COPLAS Proceedings*. Ed. by R. Barták and M.A. Salido. 2015, pp. 21–29.

[CF06] G. Codato and M. Fischetti. "Combinatorial Benders' cuts for mixed-integer linear programming". In: *Operations Research* 54 (2006), pp. 756–766.

[CH12] A. Ciré and J. N. Hooker. "A heuristic logic-based Benders method for the home health care problem". Presented at Matheuristics 2012, Angra dos Reis, Brazil. 2012.

[ÇH13] E. Çoban and J. N. Hooker. "Single-facility scheduling by logic-based Benders decomposition". In: *Annals of Operations Research* 210 (2013), pp. 245–272.

[Cha+09] S. Chahed et al. "Exploring new operational research opportunities within the home care context: The chemotherapy at home". In: *Health Care Management Science* 12 (2009), pp. 179–191.

[Che65] NV Chernikova. "Algorithm for finding a general formula for the non-negative solutions of a system of linear inequalities". In: *USSR Computational Mathematics and Mathematical Physics* 5.2 (1965), pp. 228–233.

[CL06] J.-F. Cordeau and G. Laporte. "Modeling and optimization of vehicle routing and arc routing problems". In: *Handbook on Modelling for Discrete Optimization*. Ed. by G. Appa, L. Pitsoulis, and H. P. Williams. Springer, 2006, pp. 151–191.

[Cor+07] J.-F. Cordeau et al. "Vehicle routing". In: *Handbook in Operations Research and Management Science*. Ed. by C. Barnhart and G. Laporte. Vol. 14. Elsevier, 2007, pp. 367–428.

[Cre95] Alejandro Crema. "Average shadow price in a mixed integer linear programming problem". In: *European Journal of Operational Research* 85.3 (1995), pp. 625–635.

[CS15] P. Cappanera and M. G. Scutellà. "Joint assignment, scheduling and routing models to home care optimization: A pattern-based approach". In: *Transportation Science* 49 (2015), pp. 830–852.

[Dan+07] Emilie Danna et al. "Generating multiple solutions for mixed integer programming problems". In: *International Conference on Integer Programming and Combinatorial Optimization*. Springer. 2007, pp. 280–294.

[Dan55] G. B. Dantzig. "Linear programming under uncertainty". In: *Management Science* 1.3–4 (1955), pp. 197–206.

[Des+88] M. Desrochers et al. "Vehicle routing with time windows: Optimization and approximation". In: *Vehicle Routing: Methods and Studies*. Ed. by B. L. Golden and A. A. Assad. Amsterdam: North-Holland, 1988, pp. 65–84.

[DH00] MW Dawande and John N Hooker. "Inference-based sensitivity analysis for mixed integer/linear programming". In: *Operations Research* 48.4 (2000), pp. 623–634.

[DK95] Richard L. Daniels and Panagiotis Kouvelis. "Robust Scheduling to Hedge Against Processing Time Uncertainty in Single-Stage Production". In: *Management Science* 41.2 (1995), pp. 363–376.

[DL91] M. Desrochers and G. Laporte. "Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints". In: *Operations Research Letters* 10.1 (1991), pp. 27–36.

[Dye83] Martin E Dyer. "The complexity of vertex enumeration methods". In: *Mathematics of Operations Research* 8.3 (1983), pp. 381–402.

[FB09] M. M. Fazel-Zarandi and J. C. Beck. "Solving a location-allocation problem with logic-based Benders decomposition". In: *Principles and Practice of Constraint Programming (CP 2009)*. Ed. by I. P. Gent. Vol. 5732. Lecture Notes in Computer Science. New York: Springer, 2009, pp. 344–351.

[FLM97] Komei Fukuda, Thomas M Liebling, and François Margot. "Analysis of backtrack algorithms for listing all vertices and all faces of a convex polyhedron". In: *Computational Geometry* 8.1 (1997), pp. 1–12.

[FP95] Komei Fukuda and Alain Prodon. "Double description method revisited". In: *Franco-Japanese and Franco-Chinese Conference on Combinatorics and Computer Science*. Springer. 1995, pp. 91–111.

[FR94] Komei Fukuda and Vera Rosta. "Combinatorial face enumeration in convex polytopes". In: *Computational Geometry* 4.4 (1994), pp. 191–198.

[Gal89] David Gale. *The theory of linear economic models*. University of Chicago press, 1989.

[Gec16] Gecode Team. *Gecode: Generic Constraint Development Environment*. Available from http://www.gecode.org. 2016.

[Geo72] A. M. Geoffrion. "Generalized Benders decomposition". In: *Journal of Optimization Theory and Applications* 10 (1972), pp. 237–260.

[GF03] Esther Gelle and Boi Faltings. "Solving mixed and conditional constraint satisfaction problems". In: *Constraints* 8.2 (2003), pp. 107–141.

[Gle+18] Ambros Gleixner et al. *The SCIP Optimization Suite 6.0*. Technical Report. Optimization Online, July 2018.

[GLW00] Fred Glover, Arne Løkketangen, and David L Woodruff. "Scatter search to generate diverse MIP solutions". In: *Computing Tools for Modeling, Optimization and Simulation*. Springer, 2000, pp. 299–317.

[GN77] Arthur M Geoffrion and R Nauss. “Exceptional Paper–Parametric and Postoptimality Analysis in Integer Linear Programming”. In: *Management Science* 23.5 (1977), pp. 453–466.

[GR07] Menal Guzelsoy and Theodore K Ralphs. “Duality for mixed-integer linear programs”. In: *International Journal of Operations Research* 4.3 (2007), pp. 118–137.

[Gre+08] Peter Greistorfer et al. “Experiments concerning sequential versus simultaneous maximization of objective function and distance”. In: *Journal of Heuristics* 14.6 (2008), pp. 613–625.

[GSK12] Selcuk Goren, Ihsan Sabuncuoglu, and Utku Koc. “Optimization of schedule stability and efficiency under processing time variability and random machine breakdowns in a job shop environment”. In: *Naval Research Logistics (NRL)* 59.1 (2012), pp. 26–38.

[Had+04] Tarik Hadzic et al. “Fast backtrack-free product configuration using a precompiled solution space representation”. In: *Organisational aspects of Product Configuration Systems* 10.1 (2004), p. 133.

[HH06] Tarik Hadzic and John Hooker. “Postoptimality analysis for integer programming using binary decision diagrams”. In: *GICOLAG Workshop (Global Optimization: Integrating Convexity, Optimization, Logic Programming, and Computational Algebraic Geometry), Vienna. Technical report, Carnegie Mellon University*. 2006.

[HH07] T. Hadžić and J. N. Hooker. “Cost-Bounded Binary Decision Diagrams for 0–1 Programming”. In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*. Ed. by P. Van Hentenryck and L. Wolsey. Springer, 2007, pp. 84–98.

[HH16] A. Heching and J. N. Hooker. “Scheduling home hospice care by logic-based Benders decomposition”. In: *CPAIOR Proceedings*. Ed. by C.-G. Quimper. Vol. 9676. Lecture Notes in Computer Science. Springer, 2016, pp. 187–197.

[Hie+15] G. Hiermann et al. “Metaheuristics for solving a multimodal home-healthcare scheduling problem”. In: *Central European Journal of Operations Research* 23 (2015), pp. 89–113.

[HK84] S Holm and D Klein. “Three methods for postoptimal analysis in integer linear programming”. In: *Sensitivity, Stability and Parametric Analysis*. Springer, 1984, pp. 97–109.

[HL09] A. Hertz and N. Lahrichi. “A patient assignment algorithm for home care service”. In: *Journal of the Operational Research Society* 60 (2009), pp. 481–495.

[HO03a] J. N. Hooker and G. Ottosson. “Logic-based Benders decomposition”. In: *Mathematical Programming* 96 (2003), pp. 33–60.

[HO03b] J.N. Hooker and G. Ottosson. “Logic-based Benders decomposition”. English. In: *Mathematical Programming* 96.1 (2003), pp. 33–60.

[Hoe+99] Jesse Hoey et al. “SPUDD: Stochastic planning using decision diagrams”. In: *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc. 1999, pp. 279–288.

[Hoo00] J. N. Hooker. *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. New York: Wiley, 2000.

[Hoo05] J. N. Hooker. “A hybrid method for planning and scheduling”. In: *Constraints* 10 (2005), pp. 385–401.

[Hoo06] J. N. Hooker. “An integrated method for planning and scheduling to minimize tardiness”. In: *Constraints* 11 (2006), pp. 139–157.

[Hoo07] J. N. Hooker. “Planning and scheduling by logic-based Benders decomposition”. In: *Operations Research* 55 (2007), pp. 588–602.

[Hoo12] J. N. Hooker. *Integrated Methods for Optimization, 2nd ed.* Springer, 2012.

[Hoo13] John N Hooker. “Decision diagrams and dynamic programming”. In: *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. Springer. 2013, pp. 94–110.

[Hoo95] J. N. Hooker. “Logic-based Benders decomposition”. In: *INFORMS National Meeting (INFORMS 1995)*. 1995.

[HS13] Wei He and Di-hua Sun. “Scheduling flexible job shop problem subject to machine breakdown with route changing and right-shift strategies”. English. In: *The International Journal of Advanced Manufacturing Technology* 66.1-4 (2013), pp. 501–514.

[HS17] J.N. Hooker and T. Serra. “Concise Decision Diagram Representations of Solutions of Discrete Optimization Problems”. In: *Submitted* (2017).

[IBM19] IBM. *IBM ILOG CPLEX optimization studio CPLEX user’s manual, Version 12 Release 9*. 2019.

[Jan10] Mikoláš Janota. “SAT solving in interactive configuration”. PhD thesis. Citeseer, 2010.

[Jen03] Mikkel T. Jensen. “Generating robust and flexible job shop schedules using genetic algorithms”. In: *Evolutionary Computation, IEEE Transactions on* 7.3 (June 2003), pp. 275–288.

[JLF07] Stacy L. Janak, Xiaoxia Lin, and Christodoulos A. Floudas. "A new robust optimization approach for scheduling under uncertainty: II. Uncertainty with known probability distribution". In: *Computers & Chemical Engineering* 31.3 (2007), pp. 171–195.

[Jos03] Michael Joswig. "Beneath-and-beyond revisited". In: *Algebra, Geometry and Software Systems*. Springer, 2003, pp. 1–21.

[Kha+08] Leonid Khachiyan et al. "Generating All Vertices of a Polyhedron Is Hard". In: *Discrete & Computational Geometry* 39.1 (2008), pp. 174–190.

[Lab09] Philippe Laborie. "IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems". English. In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Ed. by Willem-Jan van Hoeve and John N. Hooker. Vol. 5547. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 148–162.

[Law84] S. R. Lawrence. *Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement)*. Report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, USA. 1984.

[Lee+00] Sangbum Lee et al. "Recursive MILP model for finding all the alternate optima in LP models for metabolic networks". In: *Computers & Chemical Engineering* 24.2-7 (2000), pp. 711–716.

[Lei11] Deming Lei. "Scheduling stochastic job shop subject to random breakdown to minimize makespan". English. In: *The International Journal of Advanced Manufacturing Technology* 55.9-12 (2011), pp. 1183–1192.

[Lin17] Koos van der Linden. "Decision diagrams for decomposed mixed integer linear programs". MA thesis. Delft University of Technology, Aug. 2017.

[LJF04] Xiaoxia Lin, Stacy L. Janak, and Christodoulos A. Floudas. "A new robust optimization approach for scheduling under uncertainty:: I. Bounded uncertainty". In: *Computers & Chemical Engineering* 28.6-7 (2004), pp. 1069–1085.

[LLY14] Chung-Cheng Lu, Shih-Wei Lin, and Kuo-Ching Ying. "Minimizing worst-case regret of makespan on a single machine with uncertain processing and setup times". In: *Applied Soft Computing* 23.0 (2014), pp. 144–151.

[LV16] E. Lam and P. Van Hentenryck. "A branch-and-price-and-check model for the vehicle routing problem with location congestion". In: *Constraints* 21 (2016), pp. 394–412.

[LWS94] V. Jorge Leon, S. David Wu, and Robert H. Storer. "Robustness Measures and Robust Scheduling for Job Shops". In: *IIE Transactions* 26.5 (1994), pp. 32–43.

[ML98] Helmut E. Mausser and Manuel Laguna. "A new mixed integer formulation for the maximum regret problem". In: *International Transactions in Operational Research* 5.5 (1998), pp. 389–403.

[MMB14] D. S. Mankowska, F. Meisel, and C. Bierwirth. "The home health care routing and scheduling problem with interdependent services". In: *Health Care Management Science* 17 (2014), pp. 15–30.

[MMD07] Robert Mateescu, Radu Marinescu, and Rina Dechter. "AND/OR multivalued decision diagrams for constraint optimization". In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2007, pp. 498–513.

[Mot+53] T.S. Motzkin et al. "The double description method". In: *Contibutions to the Theory of Games*. Ed. by Harold William Kuhn and Albert William Tucker. Vol. 2. Princeton, NJ: Princeton University Press, 1953, pp. 51–74.

[MU98] Sanjay V. Mehta and Reha M. Uzsoy. "Predictable scheduling of a job shop subject to breakdowns". In: *Robotics and Automation, IEEE Transactions on* 14.3 (June 1998), pp. 365–378.

[Mur09] Katta G Murty. "A problem in enumerating extreme points, and an efficient algorithm for one class of polytopes". In: *Optimization Letters* 3.2 (2009), pp. 211–237.

[NS11] Erna Budhiarti Nababan and Opim SalimSitompul. "Manipulating Tabu List to Handle Machine Breakdowns in Job Shop Scheduling Problems". In: *AIP Conference Proceedings* 1337.1 (2011), pp. 224–228.

[NSS12] S. Nickel, M. Schröder, and J. Steeg. "Mid-term and short-term planning support for home health care services". In: *European Journal of Operational Research* 219 (2012), pp. 574–587.

[Opt19] Gurobi Optimization. "Gurobi optimizer reference manual, 2019". In: *URL: http://www. gurobi. com* (2019).

[Pac04] Elena V Pachkova. "Duality in MIP". In: *Nordic MPS 2004. The Ninth Meeting of the Nordic Section of the Mathematical Programming Society*. 014. Linköping University Electronic Press. 2004.

[Pol+07] Nicola Policella et al. "From Precedence Constraint Posting to Partial Order Schedules: A CSP Approach to Robust Scheduling". In: *AI Commun.* 20.3 (Aug. 2007), pp. 163–180.

[Pro94] J Scott Provan. “Efficient enumeration of the vertices of polyhedra associated with network LP’s”. In: *Mathematical Programming* 63.1-3 (1994), pp. 47–64.

[PT09] B. Peterson and M. Trick. “A Benders’ approach to a transportation network design problem”. In: *CPAIOR Proceedings*. Ed. by W.-J. van Hoeve and J. N. Hooker. Vol. 5547. Lecture Notes in Computer Science. New York: Springer, 2009, pp. 326–327.

[PT14] Akram Pishevar and Reza Tavakkoi-Moghaddam. “β-Robust Parallel Machine Scheduling with Uncertain Durations”. In: *Universal Journal of Industrial and Business Management* 2.3 (2014), pp. 69 –74.

[Rah+17] R. Rahmaniani et al. “The Benders decomposition algorithm: A literature review”. In: *European Journal of Operational Research* 259 (2017), pp. 801–817.

[Ras+12] M. S. Rasmussen et al. “The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies”. In: *European Journal of Operational Research* 219 (2012), pp. 598–610.

[Red+12] R. Redjem et al. “Routing and scheduling of caregivers in home health care with synchronized visits”. In: *9th International Conference of Modeling, Optimization and Simulation*. Bordeaux, France, 2012, pp. 06–08.

[Ren+12] A. Rendl et al. “Hybrid heuristics for multimodal homecare scheduling”. In: *CPAIOR Proceedings*. Ed. by N. Beldiceanu, N. Jussien, and E. Pinson. Vol. 7298. Lecture Notes in Computer Science. Springer, 2012, pp. 339–355.

[RH16] K.-D. Rest and P. Hirsch. “Daily scheduling of home health care services using time-dependent public transport”. In: *Flexible Services and Manufacturing Journal* 28 (2016), pp. 495–525.

[Roo74] Gary M Roodman. “Postoptimality analysis in integer programming by implicit enumeration: The mixed integer case”. In: *Naval Research Logistics Quarterly* 21.4 (1974), pp. 595–607.

[Sad04] R. Sadykov. “A hybrid branch-and-cut algorithm for the one-machine scheduling problem”. In: *CPAIOR Proceedings*. Ed. by J. C. Régin and M. Rueher. Vol. 3011. Lecture Notes in Computer Science. Springer, 2004, pp. 409–415.

[Sad08] R. Sadykov. “A branch-and-check algorithm for minimizing the weighted number of late jobs on a single machine with release dates”. In: *European Journal of Operational Research* 189 (2008), pp. 1284–1304.

[SAJ11] M. A. Shafia, M. Pourseyed Aghaee, and A. Jamili. “A new mathematical model for the job shop scheduling problem with uncertain processing times”. In: *International Journal of Industrial Engineering Computations* 2.2 (2011), pp. 295–306.

[SDB11] Scott Sanner, Karina Valdivia Delgado, and Leliane Nunes de Barros. “Symbolic Dynamic Programming for Discrete and Continuous State MDPs”. In: *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*. UAI’11. Barcelona, Spain: AUAI Press, 2011, pp. 643–652.

[SKK03] Carsten Sinz, Andreas Kaiser, and Wolfgang Küchlin. “Formal methods for the validation of automotive product configuration data”. In: *Ai Edam* 17.1 (2003), pp. 75–97.

[SM13] Manas Ranjan Singh and Siba Sankar Mahapatra. “A Quantum Behaved Particle Swam Optimization for Flexible Job Shop Scheduling with Random Machine Breakdowns”. In: *In proceedings of the 2013 International Conference on Smart Technologies for Mechanical Engineering, 25 - 26 October 2013, New Delhi, India*. Oct. 2013, pp. 526–532.

[Spa16] P. Span. “Wages for home health care lag as demand grows”. In: *New York Times* (2016). September 23.

[Sub05] Sathiamoorthy Subbarayan. “Integrating CSP decomposition techniques and BDDs for compiling configuration problems”. In: *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer. 2005, pp. 351–365.

[SW85] Linus Schrage and Laurence Wolsey. “Sensitivity analysis for branch and bound integer programming”. In: *Operations Research* 33.5 (1985), pp. 1008–1023.

[Swa85] Garret Swart. “Finding the convex hull facet by facet”. In: *Journal of Algorithms* 6.1 (1985), pp. 17–48.

[TBB07] D. Terekhov, J. C. Beck, and K. N. Brown. “Solving a stochastic queueing design and control problem with constraint programming”. In: *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI 2007)*. Vol. 1. AAAI Press, 2007, pp. 261–266.

[TH11] A. Trautsamwieser and P. Hirsch. “Optimization of daily scheduling for home health care services”. In: *Journal of Applied Operational Research* 3 (2011), pp. 124–136.

[Tho01] E. Thorsteinsson. “Branch and check: A hybrid framework integrating mixed integer programming and constraint logic programming”. In: *Principles and Practice of Constraint Programming (CP 2001)*. Ed. by T. Walsh. Vol. 2239. Lecture Notes in Computer Science. Springer, 2001, pp. 16–30.

[Wil89] AC Williams. “Marginal values in mixed integer linear programming”. In: *Mathematical Programming* 44.1-3 (1989), pp. 67–75.

[Wol81] Laurence A Wolsey. “Integer programming duality: Price functions and sensitivity analysis”. In: *Mathematical Programming* 20.1 (1981), pp. 173–195.

[XHK05] H Xie*, P Henderson, and M Kernahan. “Modelling and solving engineering product configuration problems by constraint satisfaction”. In: *International Journal of Production Research* 43.20 (2005), pp. 4455–4469.

[XXC13] Jian Xiong, Li-ning Xing, and Ying-wu Chen. “Robust scheduling for multi-objective flexible job-shop problems with random machine breakdowns”. In: *International Journal of Production Economics* 141.1 (2013), pp. 112–126.

[Yal+14] S. Yalçındağ et al. “A two-stage approach for solving assignment and routing Problems in home health care services”. In: *Proceedings of the International Conference on Health Care Systems Engineering*. Ed. by A. Matta et al. Vol. 61. Proceedings in Mathematics and Statistics. New York: Springer, 2014, pp. 47–59.

[YN97] Takeshi Yamada and Ryohei Nakano. “Job-shop scheduling”. In: *Genetic algorithms in engineering systems*. Ed. by Peter J. Fleming and Ali M. S. Zalzala. Vol. 55. IEE Control Engineering Series. The Institution of Engineering and Technology, 1997, pp. 134–160.

[YY02] Jian Yang and Gang Yu. “On the Robust Single Machine Scheduling Problem”. English. In: *Journal of Combinatorial Optimization* 6.1 (2002), pp. 17–33.

[ZSF12] Zahra Zamani, Scott Sanner, and Cheng Fang. “Symbolic Dynamic Programming for Continuous State and Action MDPs”. In: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*. AAAI’12. Toronto, Ontario, Canada: AAAI Press, 2012, pp. 1839–1845.