Dissertation Proposal: Modern Methodologies for Practical Discrete Optimization Models

Ryo Kimura

March 29, 2018

1 Introduction

Modern discrete optimization models, especially those motivated by practical problems, continue to grow in complexity and scale. The development of modern methodologies to address such models arising from practice is of paramount importance, and even more so given that advanced analytical tools become increasingly widespread among businesses.

This proposal presents four projects, commonly aligned in the pursuit of solving larger, more complex models involving discrete variables. The first two sections present the use of decomposition as a method for handling more complex structures; the first a combination of assignment and scheduling; the second robustness with respect to combinatorial uncertainty sets. The latter two sections propose projects which aim to increase our understanding of recently developed tools for discrete optimization; multivalued decision diagrams and quantum annealing.

2 A Logic-based Benders Approach for Home Healthcare Scheduling

(To appear in Transportation Science)

Home healthcare scheduling is the problem of assigning home health aides to geographically dispersed patients such that as many patients are covered as possible and all aides have feasible visitation schedules. The combination of routing and scheduling constraints in a single problem makes practically sized instances difficult to solve optimally with a monolithic mixed integer programming (MIP) model. While several heuristic approaches have been suggested, we propose an exact method using logic-based Benders decomposition (LBBD), which exploits the natural decomposition of the problem into an assignment and routing component. This allows us to solve each component with solution methodologies especially suited to such problems (MIP for the former, constraint programming (CP) for the latter). We implement both standard LBBD and a variant called branch-and-check (B&Ch) and compare their performance

on instances derived from real-world data. We show that B&Ch better exploits the significant difference in difficulty between solving the master problem and solving the subproblems.

2.1 The Model

We define the home healthcare problem over $d = 5$ days. Each patient j must be visited on v_j days during this time period by an assigned aide posessing a set Q_j of qualifications. Each visit has a duration of p_j time units and must take place within a time window $[r_j, d_j]$. If a patient requires multiple visits during the scheduling horizon, these visits must be "spread out" if possible; in particular, if $v_j = 2$ then there must be at least two days between each visit, and if $v_j = 3$ then there must be at least one day between each visit.

On the other side, each aide i has a set Q'_i of qualifications. On any given day k, aide i begins at their starting location b_i, travels to the home of each assigned patient, and returns to their terminal location b'_i (in our case, $b_i = b'_i$). 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, where the aide "clocks in" upon arrival at the first patient of the day and "clocks out" upon departure from the last patient.

For variables, we let $y_{ijk} = 1$ if aide i is assigned to visit patient j on day k. We let $x_{ij} = 1$ if aide i is assigned to patient j and $\delta_j = 1$ if patient j is assigned to some aide. We also let the 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 (i.e., we assume here that visits to a patient j occurs at the same time; this reflects the real-world situation we modeled).

Finally, we add a generic constraint $(x, y) \in \mathcal{X}$ to represent aide-patient assignments that are in the current schedule and should stay fixed. We note that, while the constraint is straightforward to implement (since x and y are binary variables), it is extremely important in practice, since minimizing schedule volatility is of paramount importance in home healthcare, and thus it is far preferable to modify an existing schedule to accommodate new patients than to create an entirely new schedule from scratch.

Given these constraints, our goal is to maximize the number of patients that are assigned an aide. The problem can then be stated as follows:

$$\max \sum_j \delta_j \tag{1}$$

$$\text{s.t.} \sum_i x_{ij} = \delta_j,\ \sum_{i,k} y_{ijk} = v_j \delta_j,\ \forall j \tag{2}$$

$$y_{ijk} \leq x_{ij},\ \forall i, j, k \tag{3}$$

$$x_{ij} = 0,\ \forall i, j : Q_j \not\subseteq Q'_i \tag{4}$$

$$y_{ib_ik} = y_{ib'_ik} = 1,\ \forall i, k \tag{5}$$

$$\sum_{k=\ell}^{\ell+d-v_j-1} y_{ijk} \leq 1,\ \forall j : v_j = 2, 3,\ \ell = 1, \ldots, d - v_j + 1 \tag{6}$$

$$(x, y) \in \mathcal{X} \tag{7}$$

$$\delta_j, x_{ij}, y_{ijk} \in \{0, 1\},\ \forall i, j, k \tag{8}$$

$$n_{ik} = \sum_j y_{ijk},\ \text{all-different}\{\pi_{ik\nu} \mid \nu = 1, \ldots, n_{ik}\},\ \forall i, k \tag{9}$$

$$\pi_{ik\nu} \in \{j \mid y_{ijk} = 1\},\ \forall i, k, \nu = 1, \ldots, n_{ik} \tag{10}$$

$$\pi_{ik1} = b_i,\ \pi_{ikn_{ik}} = b'_i,\ \forall i, k \tag{11}$$

$$r_j \leq s_j \leq d_j - p_j,\ \forall i, j \tag{12}$$

$$s_{\pi_{ik\nu}} + p_{\pi_{ik\nu}} + t_{\pi_{ik\nu}\pi_{ik,\nu+1}} \leq s_{\pi_{ik,\nu+1}},\ \forall i, k, \nu = 1, \ldots, n_{ik} - 1 \tag{13}$$

$$\sum_k \left(s_{\pi_{ik,n_{ik}-1}} + p_{\pi_{ik,n_{ik}-1}} - s_{\pi_{ik2}} \right) \leq U_i,\ \forall i \tag{14}$$

$$s_j \in \mathbb{R},\ \forall j;\ \pi_{ikk'} \in \mathbb{Z},\ \forall i, k, k' \tag{15}$$

(1) states the objective of maximizing the number of assigned patients. (2) ensures patients who are assigned an aide get visited the approrpriate number of times by that aide. (3) ensures only aides who are assigned to a patient can visit that patient on any given day. (4) prevents aide-patient assignments with mismatched qualifications. (5) ensures every aide starts and ends every day at their starting/terminal locations. (6) ensures visits of patients requiring multiple visits are "spread out" over the scheduling period. (7) ensures the visit schedules of patients currently assigned to aides are not altered. (8) specifies the domains of the the variables which appear in the model. (9) enforces basic constraints of a sequence variable. (10) ensures the sequence variables are taken over those patients which are assigned by the master problem variables. (11) ensures the first and last visits in the sequence are the starting/terminal locations. (12) ensures the time windows are respected. (13) ensures the aide has enough time between scheduled visits to serve the patient and travel to the next assigned patient. (14) ensures the aide does not work more than the maximum time allowed. (15) specifies the domain of the variables which appear in the subproblems.

2.2 Logic-based Benders Decomposition

The home healthcare problem has a natural decomposition into an assignment component (assigning aides to patients) and a routing component (determining the weekly visitation schedule). To take advantage of this, we utilize logic-based Benders Decomposition (LBBD), which is an extension of classical 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 $C(x)$ is a constraint 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 (such as the present one), the subproblem decouples into smaller problems that can be solved independently of one another.

When none of the subproblem variables affect the objective, which is the case for the home healthcare problem, the subproblem is a feasibility problem, and simply checks whether there exists a feasible extension $(\bar{x},\bar{y})$ to the *master problem* solution $\bar{x}$. If the subproblem is feasible, we have found an optimal solution; otherwise, we generate a *Benders cut* $\alpha_{\bar{x}}(x) \geq \beta$ which cuts off $\bar{x}$ and potentially other similar infeasible solutions. The Benders cut is then added to the master problem, which is solved to obtain the next value $\bar{x}$ to which x is fixed. We repeat the process until all subproblems are feasible or we find that the original problem is infeasible.

2.3 Branch-and-Check

Branch-and-check (B&Ch) is a variant of LBBD that more tightly integrates the master problem and subproblem solving procedures. More specifically, in B&Ch we solve the subproblem and generate Benders cuts every time we find a *feasible* solution in the master problem's branch-and-bound tree. (By contrast, in standard LBBD we solve the subproblem and generate cuts every time we find an *optimal* solution to the master problem.) Thus, (in B&Ch) when the master problem's branch-and-bound procedure terminates, every feasible solution that remains in the tree is also feasible to the subproblem, since every such solution was checked for subproblem feasibility and was not cut off as a result. Consequently, the optimal solution in the master problem's branch-and-bound tree is also optimal to the original problem.

Thie means that, unlike standard LBBD, in B&Ch the master problem only needs to be solved once; however, the subproblem is solved many more times, since we must check *every* feasible solution we encounter in the master problem's branch-and-bound tree. Intuitively, this means B&Ch should yield a computational advantage compared to standard LBBD when the time required to solve the subproblem is several orders of magnitude smaller than the time required to solve the master problem. Since this is indeed the case for our problem, we expect B&Ch to achieve better performance than standard LBBD; we verify this hypothesis in the computational results.

2.4 Master Problem, Subproblem, and Benders Cuts

In our problem, the master problem variables are y_{ijk}, x_{ij}, and δ_j, corresponding to constraints (2)–(8). Once these variables are fixed, the sets $P_{ik} = \{j \mid y_{ijk} = 1\}$, which specify which patients aide i must visit on day k, are well-defined, and we obtain one subproblem for each aide-day pair (corresponding to constraints (9)–(15)), corresponding to the problem of determining whether there exists a feasible schedule where aide i visits every patient in P_{ik} on day k. However, in order to synchronize visits and ensure the total working hours limit is satisfied (constraints (14) and (15)), we define one subproblem for each aide, collecting together subproblems corresponding to the same aide but different days.

If the subproblem for aide i is infeasible, this indicates the assignments of patients P_{ik} to aide i on days k is infeasible; hence we add a Benders cut preventing this set of patients (or any superset thereof) from being assigned to aide i in the future. More specifically, we add the following:

$$\sum_k \sum_{j \in P_{ik}} (1 - y_{ijk}) \geq 1$$

While not presented here for space, in the paper we also describe a simple cut-strengthening scheme for the generated Benders cuts.

2.5 Relaxation

Computational experience from [8] and various other articles have shown that including a suitable relaxation of the subproblem in the master problem often significantly improves the performance of LBBD. For conciseness, we only present the most successful relaxation among the three we tested (which we call the *time window relaxation*), since it significantly outperformed the other two (details of the multi-commodity flow relaxation and assignment relaxation can be found in the paper).

The time window relaxation is based on the following simple observation: suppose aide i is required to visit a set $\overline{P}$ of patients on day k, and the time window of every patient in $\overline{P}$ falls before a certain time T. Then the total time aide i spends traveling to and serving patients in $\overline{P}$ must fall between the earliest time aid i can leave their starting location and the common deadline T. We can express the total amount of time aide i spends working on patients in $\overline{P}$ by a weighted sum of the y_{ijk} variables; the total travel time can be lower bounded by taking the minimum time of travel between any two patient locations or between the home location and a patient location.

More specifically, 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 assigned to aide i on day k to be the visit duration p_j plus the minimum travel time from the previous location, which may be the aide's starting location, i.e.,

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

where J_{ik} is the set of patients that are already assigned to aide i on day k, or 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.

We now observe that the sum of backward augmented durations of visits in $J[r_{b_i}, \alpha_{ik\ell}]$ must be at most the width of the backward interval $[r_{b_i}, \alpha_{ik\ell}]$, i.e.,

$$\sum_{j \in J[r_{b_i}, \alpha_{ik\ell}]} p'_{ijk} y_{ijk} \leq \alpha_{ik\ell} - r_{b_i}$$

A similar inequality can be defined to an aide's terminal location. These inequalities are tight when the total duration of the patients in $J[t, t']$ is large relative to the width $t' - t$ of the interval. In addition ,they are more effective when scheduling on a rolling basis, because the shortest travel time is computed only over patients who are already assigned to aide i on day k or are unassigned.

2.6 Computational Results

We tested both standard LBBD (S-LBBD) and branch-and-check (B&Ch) on real-world data provided by a major hospice care organization. We also implemented a monolithic MIP model based on multi-commodity flow to compare the performance of S-LBBD and B&Ch. To obtain an initial schedule, we ran a greedy heuristic followed by running S-LBBD while fixing most of the patients scheduled by the initial heuristic to obtain a 60-patient schedule. It is better than a heuristic schedule but worse than an optimal one, as one might expect when scheduling on a rolling basis.

Table 1 shows the solution time of MIP, S-LBBD, and B&Ch (using the time window relaxation) when we attempt to re-schedule $n = 8, \ldots, 24$ of the 60 patients in the initial schedule. We see that we can optimally solve instances where roughly a third of the patients are rescheduled, which is reasonable in practice. Moreover, the results indicate that B&Ch clearly outperforms S-LBBD; this confirms our intuition that B&Ch tends to fare better when there is a significant difference in the computational difficulty between the master problem and the subproblems.

2.7 Conclusion and Future Work

We presented both LBBD and B&Ch approaches to solving the home healthcare problem, and showed that they result in algorithms that can solve practically sized instances in a reasonable time. One limitation of our work is that because the subproblems of different aides are solved independently, we are unable to handle all but the simplest of inter-aide temporal constraints. For example, in our framework there is no way to impose synchronization of different aide visits in an efficient manner. A more sophisticated decomposition scheme may allow us to implement such constraints.

Table 1: Solution times for 60-patient hospice care instances

New patients	New visits	Patients covered	MIP Time (s)	S-LBBD Iters	S-LBBD Time (s)	B&Ch Time (s)
8	40	60	43.6	7	3.17	0.63
9	45	59	41	13	5.99	0.71
10	50	59	46.6	7	3.27	0.74
11	55	59	53.3	11	5.63	0.7
12	60	59	53.2	12	6.49	1.3
13	65	59	63	21	12.3	1.11
14	70	58	113	84	72.3	9.28
15	75	58	223	86	77	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
19	95	58	1189	119	388	90
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.

3 Robust Scheduling with Combinatorial Uncertainty Sets

Robust scheduling is a broad field concerned with scheduling problems that explicitly take the uncertainty of parameters into account. More specifically, robust scheduling aims to provide a solution which performs well against the worst-case realization of the parameter values. When the uncertainy set has a "nice" structure, the robust counterpart can often be reformulated into an efficiently solvable form by applying duality theory. However, when the uncertainty set has a combinatorial structure, such an explicit reformulation may not be available. Our goal is to develop a general framework for addressing such robust scheduling problems.

The main idea is to dynamically refine the combinatorial uncertainty set as we are solving the problem. More specifically, we can view the uncertainty set as representing some set of scenarios, and if we choose a small number of "representative worst-case" scenarios, guaranteeing robustness over the smaller set of scenarios may be equivalent to guaranteeing robustness over the entire uncertainty set. We first describe the general framework, then describe its application to the robust job-shop problem with processing time delays.

3.1 Robust Scheduling

Stated most generally, we wish to solve problems of the form

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

where f is an arbitrary objective function, S 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 [2]). 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 $\overline{x} \in S$, 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

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

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.

3.2 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

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

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.

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 $\text{Obj}(x^i, q^i) - v^i = 0$ **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

Note that while we must solve (SP) to optimality if we wish to *prove* that the solution x^* is robust to all of $\mathcal{Q}$, there is no requirement that Q needs to be constructed in this way. Indeed, we may improve the overall efficiency of SGA if we only search for optimally bad scenarios some of the time, and we are content with only finding scenarios that achieve a certain minimum level of objective degradation for other iterations. This idea is explored in the paper, but we do not discuss it here for space considerations.

3.3 Example: Robust Job-Shop

3.3.1 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
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* S of operations on each machine, which implicity defines a set of schedules $\{s(S, q) : q \in \mathcal{Q}\}$, where $\mathcal{Q}$ is the set of all feasible delay scenarios and $s(S, q)$ is the unique *semi-active schedule* that respects the specified sequence S (i.e., no operation can be started earlier while still satisfying all precedence relations implied by S), 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, [1]). Thus S is declared optimal if: for any other sequence of operations S', there is a delay scenario $q' \in \mathcal{Q}$ such that $\text{Obj}(s(S', q')) \geq \max_{q \in \mathcal{Q}} \text{Obj}(s(S, q))$ 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 [6], interpreting the job-shop problem as a special case of the resource-constrained project scheduling problem.

3.3.2 Model

We use a constraint programming model to represent our problem. In particular, we make use of the interval variables, sequence variables, and scheduling constraints provided IBM CP Optimizer ([4]). 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 $\textbf{startOf}(I)$ and $\textbf{endOf}(I)$ to refer to the first and second components of I respectively; in addition, we define $\textbf{lengthOf}(I) := \text{endOf}(I) - \text{startOf}(I)$.

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 $O_q(i') = \{o_{ijq} \mid i = i'\}$ 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:

$$\begin{aligned}
\min \quad & v && && (16)\\
\text{s.t.} \quad & v \geq \text{Obj}(C_{1q}, \ldots, C_{nq}) && \forall q \in \mathcal{Q} && (17)\\
& C_{jq} \geq \text{endOf}(o_{ijq}) && \forall i \in \mathcal{M}, \forall j \in \mathcal{J}, \forall q \in \mathcal{Q} && (18)\\
& \text{sameSequence}(S_{i1}, S_{iq}) && \forall i \in \mathcal{M}, q \in \mathcal{Q} && (19)\\
& \text{noOverlap}(S_{iq}) && \forall i \in \mathcal{M}, \forall q \in \mathcal{Q} && (20)\\
& \text{endBeforeStart}(o, o') && \forall (o, o') \in \mathcal{P}_q, \forall q \in \mathcal{Q} && (21)\\
& \text{lengthOf}(o_{ijq}) = d_{ij} && \forall i \in \mathcal{M}, \forall j \in \mathcal{J}, \forall q \in \mathcal{Q} && (22)\\
& C_{jq} \geq 0, C_{jq} \in \mathbb{Z} && \forall j \in \mathcal{J}, \forall q \in \mathcal{Q} && (23)\\
& o_{ijq} \in \text{CPOInterval} && \forall i \in \mathcal{M},\ \forall j \in \mathcal{J}, \forall q \in \mathcal{Q} && (24)\\
& S_{iq} \in \text{CPOSequence}(\mathcal{O}_q(i)) && \forall i \in \mathcal{M}, \forall q \in \mathcal{Q} && (25)
\end{aligned}$$

While we do not present it here, we note that generating the worst-case scenarios for a given sequence S (i.e., solving (SP)) can also be modeled as a constraint program.

3.3.3 Computational Results

Computational results for robust job-shop are underwhelming but unsurprising, and so are not shown here; for certain instances the robust solution provides significantly better worst case behavior than the solution obtained by solving the determinstic problem, while for other instances there is no significant difference. Further, the scenario generation algorithm (SGA) does not scale for larger values of k; this, however, is unsurprising, given that the deterministic job shop problem is already a notoriously difficult NP-hard problem.

3.4 Future Work

While the robust job-shop problem is well-motivated and its determinstic version has been extensively studied, it is perhaps unsuited for an evaluation of SGA, given the deterministic problem already requires specialized approaches to obtain a scalable algorithm. Scheduling problems where the deterministic problem is solvable in practice (though still NP-hard), such as the traveling repairman problem, or parallel machine scheduling problems, may given more promising/insightful results.

4 Post Optimality Analysis of Mixed Integer Programs using Decision Diagrams

Recent work by [7] has shown that multivalued decision diagrams (MDDs) provide a useful framework for representing and answering questions about near-optimal solutions, after the problem has been solved. Their work focused on pure integer programs, and showed that with an appropriate notion of "reduction" which preserves representability of near-optimal solutions (called *sound reduction*), such an MDD can be compiled efficiently and provide users the ability to ask questions about the set of near-optimal solutions.

This work proposes to extend their results to mixed integer programs (MIPs). The biggest challenge to this extension is determining how to incorporate continuous variables into the framework of decision diagrams. As they stand, decision diagrams are inherently a very discrete object; each path corresponds to a single feasible solution, and the set of solutions represented by an MDD is a finite, discrete set. However, the set of feasible solutions (and thus near-optimal solutions) for a MIP is neither finite nor discrete. Thus, we must either explicitly represent the continuous part in the MDD via some form of discretization, or implicitly represent it by careful construction of an MDD which does not represent the whole problem to full precision.

Thus the main questions for this project are as follows:

1. What is the "right way" to represent the set of near-optimal solutions of a MIP via an MDD?

2. What properties must such an MDD satisfy in order to allow for useful post-optimality analysis? Can such an MDD be constructed efficiently?

3. How does this framework perform for practical MIP instances?

4.1 Multivalued Decision Diagrams

For our purposes, a decision diagram D is a directed multigraph that represents the set of solutions to a problem of the form

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

where the domain S_j of each variable x_j is finite. 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 and U_{n+1} containing only the *terminal node* t. 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, and a *weight* $w_j(a)$ that represents the cost of assigning value $x_j(a)$ to variable x_j. The arcs leaving u must have distinct labels. If it possible for more than two arcs to leave each node, D is called a *multivalued decision diagram* (MDD).

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 (P) if its r-t paths represent precisely the values of x for which $x \in S$ and each arc a leaving a node in layer U_j is given weight $f_j(x_j(a))$. In this case, the weight of any r-t path p in D is the cost $\sum_j f_j(x_j(p))$ of the corresponding solution $x(p)$, and any shortest r-t path defines an optimal solution $x(p)$ of (P).

4.2 Sound Decision Diagrams for Pure IPs

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, i.e.,

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

We say a diagram D *exactly represents* $P(\Delta)$ if its r-t paths correspond exactly to the Δ-optimal solutions of (P). While exact representation is ideal, it may result in diagram with exponentially many nodes. Fortunately, a decision diagram is sometimes smaller when it contains more paths, hence implying that certain relaxations of the solution set can be encoded more concisely. As defined

by [3], sound decision diagrams take advantage of this by allowing solutions that are not Δ-optimal, but only when they are worse than Δ-optimal.

Formally, diagram D is *sound* for $P(\Delta)$ when every Δ-optimal solution of (P) is represented by an r-t path of D, and every r-t path of D either represents a Δ-optimal solution or has weight greater than $z^* + \Delta$. A path with weight greater than $z^* + \Delta$ can represent a feasible or infeasible solution of (P).

[7] defines the notion of *sound reduction* (which preserves soundness of a diagram), then prove that a sound diagram D for $P(\Delta)$ has a mininum number of nodes and arcs if and only if D is minimal (i.e., every node and arc is lies on some r-t path representing a Δ-optimal solution) and sound reduction cannot be further applied. This allows for the construction of minimal sound decision diagrams by repeated application of sound reduction. Further, given a diagram sound for $P(\Delta)$, they show how to efficiently calculate what is called the δ-*optimal domain* of a variable x_j (i.e., the set of values x_j can take among all δ-optimal solutions), for any given $\delta \leq \Delta$.

Our goal is to extend these results to the MIP case, and if not obtain something comparable or show that it is impossible.

4.3 Representation of Continuous Variables

We briefly present two potential approaches to incorporating continuous variables in a decision diagram. For terminology, let

$$(\text{MIP}) = \min\{c^\mathsf{T} x + d^\mathsf{T} y : Ax + By \geq b,\ x \in S,\ y \geq 0\}$$

Let z^* denote the optimal value of (MIP). Note that for any $\bar{x} \in S$, we obtain an upper bound on z^* via

$$z^*(\bar{x}) = c^\mathsf{T}\bar{x} + \min\{d^\mathsf{T} y : By \geq b - A\bar{x},\ y \geq 0\} = c^\mathsf{T}\bar{x} + L_{\bar{x}}(y)$$

4.3.1 Implicit Representation

One approach is to not explicitly represent continuous variables in the diagram, but instead only represent the discrete variables, then incorporate the continuous variables in filtering arcs. [5] shows how this can be done though the lens of Benders decomposition. Potential ideas include defining an appropriate notion of *equivalence* between two nodes, representation of the cost of a solution (since this is no longer explicitly represented in the diagram), and rules for determining when two nodes can be merged.

4.3.2 Explicit Representation

An alternative approach is to discretize the feasible region of the MIP by only considering basic feasible solutions. Specifically, we say $(\bar{x}, \bar{y})$ is a *basic feasible solution* of (MIP) if $\bar{x} \in S$ and $\bar{y}$ is a basic feasible solution of the LP defined by $L(\bar{x})$. An advantage of this approach is that the cost is exactly represented in the diagram; a drawback is that the objective function is no longer separable, and thus it is more difficult to merge nodes while maintaining soundness.

5 Quantum Annealing

Quantum computing is a method of computation unlike anything that comes before it; as a result, there is great potential for developing efficient algorithms for problems which were previously inaccessible by classical computers. However, very little of the nature of quantum computing is known, particularly in reference to its application to optimiziation.

There are many approaches to utilizing quantum computing for optimization. Among these, the most immediately testable is the framework of quantum annealing. Algorithmically speaking, quantum annealing is the quantum analogue of simulating annealing, and both are heuristic methods which are generally used to approximately solve hard combinatorial optimization problems. The main advantage of quantum annealing is its ability to escape local optima via quantum tunneling and that it can be (and in fact has been) implemented directly using a quantum chip, allowing the algorithm to be run extremely quickly. However, there are many caveats to quantum annealing as well; most notably, quantum annealing can only natively solve quadratic unconstrained binary optimization problems, a narrow but still NP-hard class of problems. Also, physical limitations on the quantum annealing chip require us to solve a minor-embedding problem on a subgraph of the Chimera graph every time we want to solve a problem, which is NP-hard in general. Our overaching goal is to better understand the properties of quantum annealing as an algorithm with respect to combinatorial optimization. More specifically,

1. Are there any common structural properties among optimization problems that are known to "work well" with quantum annealing?

2. Which classes of graphs are easiest to minor-embed in a Chimera graph? Are there any classes of optimization problems which correspond to such graphs?

5.1 Quantum Annealing

Quantum annealing is an algorithm that heuristially solves the Ising problem

$$(P_I) = \min_{s\in\{-1,+1\}^n} \left\{ s^\mathsf{T} J s + h^\mathsf{T} s \right\}$$

parameterized by $J \in \mathbb{S}^n$ and vector $h \in \mathbb{R}^n$, where $\mathbb{S}^n$ is the set of $n \times n$ symmetric matrices. By a simple transformation, this is equivalent to an unconstrained binary optimization problem (QUBO), i.e.,

$$(P_Q) = \min_{x\in\{0,1\}^n} \left\{ x^\mathsf{T} Q x \right\}$$

parameterized by $Q \in \mathbb{S}^n$.

Roughly speaking, quantum annealing works by exploiting the *quantum adiabatic principle*, which states that an Ising system (defined by J and h) in a state of lowest energy stays (with high probability) in a state of lowest energy

when J and h are changed gradually over time. In particular, a quantum annealer is initially configured to an Ising system whose state of lowest energy is trivially known (e.g., $J = I$ is the identity matrix and $h = 0$ is the zero vector). Then I and 0 are gradually changed to the J and h of interest, then we take a measurement to observe the state of the system. When this is repeated many times, with high probability the state of the system will be in a low energy state of the Ising system defined by J and h.

This effectively means that quantum annealing provides us with a way to very quickly produce an ensemble of "good" solutions to a QUBO. Investigating how best to exploit this particular property of quantum annealing is an overarching goal of this project.

5.2 Minor Embedding Problem

A major practical roadblock for utilizing quantum annealing is the necessity of embedding the QUBO we wish to solve into the quantum annealer. Due to physical limitations, the quantum annealer is configured with a fixed connectively structure (in particular, a Chimera graph). However, we can force different nodes to be treated as the same by adjusting the coefficient of J appropriately; this effectively amounts to *contracting* the corresponding arc in the underlying graph. Thus, whenever we wish to solve a particular QUBO via quantum annealing, we must generate a *minor embedding* of the graph corresponding to the constraint matrix into the underlying graph of the quantum annealer.

While certain classes of problems have been solved via quantum annealing, there is result connecting the structure of of the constraint matrix of a QUBO and the ease of finding a minor embedding. Clarifying this connection is a major focus of this project.

References

[1] 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.

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

[3] Tarik 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*. Ed. by Pascal Van Hentenryck and Laurence Wolsey. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 84–98.

[4] 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.

[5] Koos van der Linden. “Decision diagrams for decomposed mixed integer linear programs”. MA thesis. TU Delft, 2017.

[6] 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.

[7] Thiago Serra and JN Hooker. “Compact Representation of Near-Optimal Integer Programming Solutions”. In: (2017).

[8] Erlendur S. Thorsteinsson. “Branch-and-Check: A Hybrid Framework Integrating Mixed Integer Programming and Constraint Logic Programming”. In: *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*. CP ’01. London, UK, UK: Springer-Verlag, 2001, pp. 16–30.