Planning and Scheduling by Logic-Based Benders Decomposition

J. N. Hooker
Carnegie Mellon University
john@hooker.tepper.cmu.edu

July 2004, Revised October 2005

Abstract

We combine mixed integer linear programming (MILP) and constraint programming (CP) to solve an important class of planning and scheduling problems. Tasks are allocated to facilities using MILP and scheduled using CP, and the two are linked via logic-based Benders decomposition. Tasks assigned to a facility may run in parallel subject to resource constraints (cumulative scheduling). We solve problems in which the objective is to minimize cost, makespan, or total tardiness. We obtain significant computational speedups, of several orders of magnitude for the first two objectives, relative to the state of the art in both MILP and CP. We also obtain better solutions and bounds for problems than cannot be solved to optimality.

We address a fundamental planning and scheduling problem that occurs frequently in manufacturing and supply chain contexts. Tasks must be assigned to facilities and scheduled on each facility subject to release dates and deadlines. Some of the tasks assigned to a given facility may run in parallel if desired, provided the total resource consumption at any time remains within limits ("cumulative scheduling"). Precedence and other side constraints may be imposed. The objective is to minimize cost, makespan, or total tardiness.

Although problems of this sort arise often, they have proved to be quite difficult to solve. In the absence of an effective formal algorithm, a common-sense approach is often used in practice. The problem is decomposed into an assignment portion and a scheduling portion. Central managers assign tasks to facilities, whereupon operations managers develop detailed schedules for the tasks assigned to each facility. Since the schedule for one or more facilities may prove infeasible or otherwise unsuitable, the schedulers may telephone headquarters and ask for a different allocation of tasks. The give-and-take continues until a mutually satisfactory solution is reached. However, an informal process of this sort is time consuming, and even if carried to completion it offers no guarantee of obtaining a good solution.

One possibility that suggests itself is to apply Benders decomposition, which mimics the informal feedback loop in a rigorous fashion. The Benders master problem allocates tasks

to facilities, and the subproblem separates into several independent scheduling problems. Benders cuts added to the master problem are a mathematical equivalent of telephone calls from the operations managers. Under typical conditions the Benders algorithm terminates with an optimal solution of the overall allocation and scheduling problem.

The classical Benders approach (Benders 1962, Geoffrion 1972) is inappropriate for this problem, however, because it requires that the subproblem be a continuous linear or nonlinear programming problem. Scheduling is a highly combinatorial problem that has no practical linear or nonlinear programming model. Fortunately, the idea of Benders decomposition can be extended to a *logic-based* form that accommodates an arbitrary subproblem, such as a discrete scheduling problem. This paper explores how a logic-based Benders method can be applied to planning and scheduling problems.

An additional advantage of logic-based Benders is that it permits one to exploit the relative strengths of mixed integer linear programming (MILP) and constraint programming (CP). MILP is a natural approach to the allocation task of the master problem, and CP provides an effective scheduling technology that can be applied to the subproblem. Logic-based Benders therefore unites MILP and CP in integrated approach that obtains provably optimal solutions.

The planning and scheduling problem can be formulated entirely as an MILP problem, or entirely as a CP problem. However, these models are hard to solve. By linking MILP and CP, we obtained substantial speedups relative to the existing state of the art in both MILP (as represented by CPLEX) and CP (as represented by the ILOG Scheduler).

In particular, we obtained speedups of several orders of magnitude for minimum cost and minimum makespan problems. This allowed us to solve larger instances to optimality than could be solved previously. For minimum tardiness problems the computational advantage of a Benders method is less but nonetheless substantial. Although we were not able to solve significantly larger instances to optimality, we obtained much better solutions and better bounds on the optimal value, within a given amount of computation time, than provided by MILP and CP. One advantage of the Benders method is that it can be terminated at almost any point with a feasible solution and a lower bound on the optimal value (except in the case of the minimum cost problems). The bound improves steadily as the algorithm runs.

Unlike classical Benders, logic-based Benders provides no standard scheme for generating Benders cuts. Cuts must be devised for each problem class, and a major goal of this paper is to show that this can be done for a variety of objective functions. Specifically we develop Benders cuts for (a) minimum cost problems in which cost is a function of the assignment alone; (b) minimum makespan problems in which all tasks have the same release date; (c) minimum total tardiness problems in which all tasks have the same release date. We also develop relaxations of the subproblem for inclusion in the master problem, since this often proves important for good performance.

1 Previous Work

Classical Benders decomposition solves a problem by enumerating values of certain *primary* variables. For each set of values enumerated, it solves the subproblem that results from fixing the primary variables to these values. (In our case, the primary variables define the allocation of tasks to facilities.) Solution of the subproblem generates a Benders cut that the primary variables must satisfy in all subsequent solutions enumerated. The Benders cut is a linear inequality based on Lagrange multipliers obtained from a solution of the subproblem dual. The next set of values for the primary variables is obtained by solving the master problem, which contains all the Benders cuts so far generated. The process continues until the master problem and subproblem converge in value.

Logic-based Benders decomposition was introduced by Hooker and Yan (1995) in the context of logic circuit verification. The idea was formally developed by Hooker (2000) and applied to 0-1 programming by Hooker and Ottosson (2003). In logic-based Benders, the Benders cuts are obtained by solving the *inference dual* of the subproblem, of which the linear programming dual is a special case. The solution of the inference dual is a proof of optimality when the primary variables are fixed to their current values. The Benders cut is constructed by using this same proof to derive a valid bound on the optimal value when the primary variables take other values. Although logic-based Benders cuts can assume any form, in the present context they must be formulable as linear inequalities, since the master problem is an MILP.

Jain and Grossmann (2001) successfully applied logic-based Benders to minimum-cost planning and scheduling problems in which the subproblems are disjunctive scheduling problems in which tasks must be run one at a time, rather than cumulative scheduling problems. Harjunkoski and Grossmann (2002) extended this work to multistage problems. The Benders cuts are particularly simple in these cases because the subproblem is a feasibility problem rather than an optimization problem. A goal of the present paper is to show that a Benders approach can succeed for a variety of planning and scheduling problems in which the Benders cuts are less obvious. In particular, we (a) accommodate cumulative scheduling, and (b) develop Benders cuts for two additional objective functions for which the subproblem is an optimization problem. Preliminary results are presented in conference papers (Hooker 2004, 2005).

In related work, Hooker (2005) applied a Benders method to minimizing the number of late jobs. Chu and Xia (2005) solved the minimum makespan problem by generating Benders cuts with the help of an integer programming model of the subproblem, but the speedups are substantially less than reported here.

Hooker (2000) observed that the master problem need only be solved once by a branching algorithm that accumulates Benders cuts as they are generated. Thorsteinsson (2001) showed that this approach, which he called *branch-and-check*, can result in substantially better performance on the Jain and Grossmann problems than standard logic-based Ben-

ders. We did not implement branch and check for this study because it would require hand coding of a branch-and-cut algorithm for the master problem. But we obtained substantial speedups without it.

Logic-based Benders methods have also been adapted to solving integer programming problems (Hooker and Ottosson 2003, Chu and Xia 2004) and the propositional satisfiability problem (Hooker 2000, Hooker and Ottosson 2003). Similar ideas have been applied to minimal dispatching of automated guided vehicles (Corré et al. 2004), steel production scheduling (Harjunkowski and Grossmann 2001), real-time scheduling of computer processors (Cambazard et al. 2004), traffic diversion (Xia et al. 2004), batch scheduling in a chemical plant (Maravelias and Grossmann 2004), and polypropylene batch scheduling in particular (Timpe 2002). In all these applications (except integer programming), the subproblem is a feasibility problem.

Classical Benders decomposition can also be useful in a CP context, as shown by Eremin and Wallace (2001).

2 The Problem

The planning and scheduling problem may be defined as follows. Each task $j \in \{1, \dots, n\}$ is to be assigned to a facility $i \in \{1, \dots m\}$, where it consumes processing time p_{ij} and resources at the rate c_{ij}. Each task j has release time r_j and deadline d_j (viewed as a due date in tardiness problems). The tasks assigned to facility i must be given start times s_j in such a way that the total rate of resource consumption on facility i is never more than C_i at any given time. If x_j is the facility assigned to task j, the problem may be written

$$\begin{array}{lll} \text{minimize} & g(x,s) & \\ \text{subject to} & r_j \le s_j \le d_j - p_{x_j j}, \ \text{ all } j & (a) \\ & \displaystyle\sum_{j \in J_{it}} c_{ij} \le C_i, \ \text{ all } i,t & (b) \end{array} \tag{1}$$

where $J_{it} = \{j \mid s_j \le t < s_j + p_{ij}\}$ is the set of tasks underway at time t in facility i. It is assumed that for each j, $c_{ij} \le C_i$ for at least one i.

An important special case is that in which the scheduling problem on facility i is an m-machine scheduling problem, for which $C = m$ and each $c_{ij} = 1$. For a one-machine ("disjunctive") scheduling problem we have $C_i = 1$.

Precedence and other side constraints may be added if desired. For instance, one may require that tasks j and k be scheduled on the same facility, and that task j precede k, by writing the constraints $x_j = x_k$ and $s_j + p_{x_j j} < s_k$.

We investigate three objective functions:

- *cost*, given by $g(x,s) = \sum_j F_{x_j j}$, where F_{ij} is the fixed cost of processing task j on facility i;

- *makespan*, given by $g(x, s) = \max_j\{s_j + p_{x_j j}\}$;
- *total tardiness*, given by $g(x, s) = \sum_j (s_j + p_{x_j j} - d_j)^+$, where α^+ is α if $\alpha > 0$ and 0 otherwise.

When the tardiness objective is used, the deadlines become due dates and constraint (a) becomes $r_j \leq s_j$ for all j.

3 Constraint Programming Formulation

The problem can be written in a constraint programming fashion using the "global" constraints *element* and *cumulative*. Given a vector v of values, the constraint

$$\text{element}(y, v, u) \tag{2}$$

sets u equal to the yth component of v. Thus one can implement an expression of the form v_y by replacing v_y with u and writing constraint (2).

If $s = (s_1, \ldots s_n)$ and similarly for p and c, the constraint

$$\text{cumulative}(s, p, c, C) \tag{3}$$

requires that tasks be scheduled at times $s_1, \ldots, s_n$ so that the total rate of resource consumption at any given time never exceeds C. Thus $\sum_{j \in J_t} c_j \leq C$ for all t, where $J_t = \{j \mid s_j \leq t < s_j + p_{ij}\}$ is the set of tasks underway at time t. The cumulative constraint is widely used for resource-constrained scheduling problems in which tasks can run simultaneously.

Constraint programming solvers rely heavily on efficient "filtering" or domain reduction algorithms for these and other constraints. The filtering algorithms remove from the domain of each s_j some or all values that cannot be part of a solution satisfying the constraint.

The planning and scheduling problem now becomes

$$\begin{array}{ll} \text{minimize} & g(x, s) \\ \text{subject to} & r_j \leq s_j \leq d_j - u_j, \text{ all } j \\ & \text{element}(x_j, (p_{1j}, \ldots, p_{mj}), u_j), \text{ all } j \\ & \text{cumulative}((s_j | x_j = i), (p_{ij} | x_j = i), (c_{ij} | x_j = i), C_i), \text{ all } i \end{array} \tag{4}$$

where $g(x, s)$ is the desired objective function and $(s_j | x_j = i)$ denotes the tuple of start times for tasks assigned to facility i. The first and second constraints enforce the time windows and the third enforces the resource constraint.

For purposes of computational testing we formulated (4) using the high-level modeling language of OPL Studio. The formulation for the minimum cost problem appears in Fig. 1. Formulations for the other objectives are similar. The assignAlternatives and setTimes search options specify a branching method that results in substantially better performance than the default method.

```
[declarations]

scheduleHorizon = max(j in J) task[j].d;
DiscreteResource facility[i in I](C[i]);
AlternativeResources facilitySet(facility);
Activity scheduleTask[j in J];

minimize
    sum(j in J) cost[j]
subject to {
    forall(j in J)
        scheduleTask[j] requires(taskf[i,j].r) facilitySet;
    forall(j in J)
        forall(i in I)
            activityHasSelectedResource(scheduleTask[j],facilitySet,facility[i]) <=>
                                    scheduleTask[j].duration = taskf[i,j].p & cost[j] = taskf[i,j].F;
    forall(j in J) {
        scheduleTask[j].start >= task[j].r;
        scheduleTask[j].end <= task[j].d;
    };
};

search {
    assignAlternatives;
    setTimes;
};
```

Figure 1: OPL Studio code for the CP version of a minimum cost problem. Parameters r_j, d_j, p_{ij}, and F_{ij} are represented by task[j].r, task[j].d, taskf[i,j].p, and taskf[i,j].F, respectively. The resource limit C_i is C[i]. Index set J is $\{1, \ldots n\}$, and I is $\{1, \ldots, m\}$.

4 Mixed Integer Programming Formulation

The most straightforward MILP formulation discretizes time and enforces the resource capacity constraint at each discrete time. Let the 0-1 variable $x_{ijt} = 1$ if task j starts at

discrete time t on facility i. The minimum cost formulation is

$$\begin{aligned}
\min \quad & \sum_{ijt} F_{ij} x_{ijt} \\
\text{subject to} \quad & \sum_{it} x_{ijt} = 1, \ \text{ all } j & (a) \\
& \sum_{j} \sum_{t' \in T_{ijt}} c_{ij} x_{ijt'} \leq C_i, \ \text{ all } i, t & (b) \\
& x_{ijt} = 0, \ \text{ all } j, t \text{ with } t \leq r_j \text{ or } t > d_j - p_{ij} & (c)
\end{aligned} \tag{5}$$

where each x_{ijt} is a 0-1 variable. $T_{ijt} = \{t' \mid t - p_{ij} < t' \leq t\}$ is the set of discrete times at which a task j in progress on facility i at time t might start processing. Constraint (a) ensures that each task starts once on one facility, (b) enforces the resource limit, and (c) the time windows. The minimum makespan problem is

$$\begin{aligned}
\min \quad & M \\
\text{subject to} \quad & M \geq \sum_{it} (t + p_{ij}) x_{ijt}, \ \text{ all } j \\
& \sum_{it} x_{ijt} = 1, \ \text{ all } j \\
& \sum_{j} \sum_{t' \in T_{ijt}} c_{ij} x_{ijt'} \leq C_i, \ \text{ all } i, t \\
& x_{ijt} = 0, \ \text{all } j, t \text{ with } t \leq r_j \text{ or } t > d_j - p_{ij}
\end{aligned} \tag{6}$$

The minimum total tardiness problem is

$$\begin{aligned}
\min \quad & \sum_{j} T_j \\
\text{subject to} \quad & T_j \geq \sum_{it} (t + p_{ij}) x_{ijt} - d_j, \ \text{ all } j \\
& \sum_{it} x_{ijt} = 1, \ \text{ all } j \\
& \sum_{j} \sum_{t' \in T_{ijt}} c_{ij} x_{ijt'} \leq C_i, \ \text{ all } i, t \\
& x_{ijt} = 0, \ \text{all } j, t \text{ with } t \leq r_j \\
& T_j \geq 0, \ \text{ all } j
\end{aligned} \tag{7}$$

where T_j is the tardiness of task j.

Models (5)-(7) can grow large due to time-indexed variables. We therefore investigated a smaller discrete event model suggested by Türkay and Grossmann (1996), which uses continuous time. Given n tasks, there are $2n$ events. The model decides, for each event, which task it is associated with, whether it is the start or end of the task, and when the task starts. Let binary variable x_{ijk} be 1 when event k is the start of task j on facility i, and let y_{ijk} be 1 when event k is the end of task j on facility i. Continuous variable s_{ik} is the start time of event k on facility i when the event is assigned to facility i. Continuous variable f_{ij} is the finish time of task j on facility i when task j is assigned to facility i. The disaggregation of the start and finish time variables allows the model to impose resource constraints.

The minimum cost model is

$$
\begin{aligned}
\min \quad & \sum_{ijk} F_{ij} x_{ijk} \\
\text{subject to} \quad & \sum_{ik} x_{ijk} = 1, \ \ \sum_{ik} y_{ijk} = 1, \ \ \text{all } j && (a) \\
& \sum_{ij} x_{ijk} + y_{ijk} = 1, \ \ \text{all } k && (b) \\
& \sum_{k} x_{ijk} = \sum_{k} y_{ijk}, \ \ \text{all } i, j && (c) \\
& s_{i,k-1} \le s_{ik}, \ \ \text{all } i, k && (d) \\
& r_j x_{ijk} \le s_{ik}, \ \ f_{ij} \le d_j, \ \ \text{all } i, j, k && (e) \\
& s_{ik} + p_{ij} x_{ik} - M(1 - x_{ijk}) \le f_{ij} \\
& \qquad \le s_{ik} + p_{ik} x_{ijk} + M(1 - x_{ijk}), \ \ \text{all } i, j, k && (f) \\
& s_{ik} - M(1 - y_{ijk}) \le f_{ij} \le s_{ik} + M(1 - y_{ijk}), \ \ \text{all } i, j, k && (g) \\
& R_{ik} \le C_i, \ \ \text{all } i, k && (h) \\
& R_{i1} = R^s_{i1}, \ \ R^s_{ik} = \sum_j c_{ij} x_{ijk}, \ \ R^f_{ik} = \sum_j c_{ij} y_{ijk}, \ \ \text{all } i, k && (i) \\
& R^s_{ik} + R_{i,k-1} - R^f_{ik} = R_{ik}, \ \ \text{all } i, k && (j) \\
& x_{ijk}, y_{ijk} \in \{0, 1\}, \ \ s_{ik}, f_{ij}, R_{ik}, R^s_{ik}, R^f_{ik} \ge 0, \ \ \text{all } i, j, k
\end{aligned}
$$

Constraints (a), (b) and (c) ensure that each task is assigned to one facility, starts and ends once, and starts and ends on the same facility. Constraint (d) requires events $1, \ldots, 2n$ to occur in chronological order. Constraint (e) enforces the time windows. Constraints (f) and (g) define the finish time variables in terms of the start time variables.

The rest of the model encodes the resource constraints. Variable R_{ik} is the total resource consumption of tasks running on facility i at the time that event k occurs. Constraint (h)

imposes the upper bound on resource consumption at the time of each event, which is sufficient to impose the bound at all times. Constraint (i) computes the resource consumption R^s_{ik} of any task starting at the time of event k and the consumption R^f_{ik} of any task finishing at the time of event k. Constraint (j) maintains R_{ik} at the time of each event in the fashion of an inventory balance equation.

Although it is smaller than (5), the discrete event model proved to be much harder to solve. We therefore omitted it from the computational results given below.

5 Logic-based Benders Decomposition

Logic-based Benders decomposition applies to problems of the form

$$\begin{array}{ll} \text{minimize} & f(x,y) \\ \text{subject to} & C(x,y) \\ & x \in D_x,\ y \in D_y \end{array} \tag{8}$$

where $C(x,y)$ is a set of constraints containing variables x, y. D_x and D_y denote the domains of x and y, respectively. When x is fixed to a given value $\bar{x} \in D_x$, the following *subproblem* results:

$$\begin{array}{ll} \text{minimize} & f(\bar{x},y) \\ \text{subject to} & C(\bar{x},y) \\ & y \in D_y \end{array} \tag{9}$$

Here $C(\bar{x},y)$ is the constraint that results from fixing $x = \bar{x}$ in $C(x,y)$.

The *inference dual* of (9) is the problem of inferring the tightest possible lower bound on $f(\bar{x},y)$ from $C(\bar{x},y)$. It can be written

$$\begin{array}{ll} \text{maximize} & v \\ \text{subject to} & C(\bar{x},y) \overset{P}{\Longrightarrow} f(\bar{x},y) \geq v \\ & v \in R,\ p \in \mathcal{P} \end{array} \tag{10}$$

where $A \overset{P}{\Longrightarrow} B$ means that proof P deduces B from A, and $\mathcal{P}$ is a family of proofs. The inference dual becomes the linear programming dual when (9) is a linear programming problem and the proofs in $\mathcal{P}$ correspond to nonnegative linear combinations of the constraints; see Hooker (2000) for details.

The solution of the dual can be viewed as a proof of the tightest possible bound $\hat{v}$ on $f(x,y)$ when $x = \bar{x}$. The basic idea of Benders decomposition is to use the same proof schema to derive a bound $B_{\bar{x}}(x)$ for other values of x. In classical Benders, the same linear combination is used. In general the bounding function $B_{\bar{x}}(x)$ should have two properties:

B1. $B_{\bar{x}}(x)$ provides a valid lower bound on $f(x,y)$ for any given $x \in D_x$. That is, $f(x,y) \geq B_{\bar{x}}(x)$ for any feasible (x,y) in (8).

B2. In particular, $B_{\bar{x}}(\bar{x}) = \hat{v}$.

If z is the objective function value of (8), the valid inequality $z \geq B_{\bar{x}}(x)$ is a *Benders cut*.

In iteration H of the Benders algorithm, we solve a *master problem* whose constraints are the Benders cuts so far generated:

$$\begin{array}{ll} \min & z \\ \text{subject to} & z \geq B_{x^h}(x),\ h = 1, \ldots, H-1 \\ & z \in R,\ x \in D_x \end{array} \tag{11}$$

Here $x^1, \ldots, x^{H-1}$ are the solutions of the previous $H-1$ master problems. Then the solution $\bar{x}$ of (11) defines the next subproblem (9).

If we let $v_1^*, \ldots, v_{H-1}^*$ denote the optimal values of the previous $H-1$ subproblems, the algorithm continues until the optimal value z_H^* of the master problem equals $v^* = \min\{v_1^*, \ldots, v_{H-1}^*\}$. At any point in the algorithm, z_H^* and v^* provide lower and upper bounds on the optimal value of the problem.

The following is shown by Hooker (2000). It is convenient to regard (8) or (9) as having an infinite optimal value if it is infeasible.

Theorem 1 *If the bounding function $B_{\bar{x}}(x)$ satisfies (B1) and (B2) in each iteration of the Benders algorithm, and D_y is finite, the Benders algorithm converges to the optimal value of (8) after finitely many steps.*

In the planning and scheduling problem (1), variables x, s correspond respectively to x, y in (8). Any assignment $\bar{x}$ of tasks to facilities creates the subproblem:

$$\begin{array}{ll} \min & g(\bar{x}, s) \\ \text{subject to} & \text{cumulative}((s_j|\bar{x}_j = i), (p_{ij}|\bar{x}_j = i), (c_{ij}|\bar{x}_j = i), C_i), \text{ all } i \\ & r_j \leq s_j \leq d_j - p_{\bar{x}_j j},\ \text{all } j \end{array} \tag{12}$$

which decomposes into a separate scheduling problem for each facility i. After solving the subproblem with a finite-domain CP solver, we generate a Benders cut that becomes part of the master problem (11), which is formulated as an MILP. The domain D_s can be regarded as finite, as required in Theorem 1, since the CP solver assumes a finite set of start times (such as integers). Release times and deadlines are typically given as integers.

The bounding function $B_{\bar{x}}(x)$ is generally obtained by examining the type of reasoning that led to a bound for $x = \bar{x}$ and extending this reasoning to obtain a bound for general x. In the present context, however, only the primal solution (the schedule itself) is available from the commercial CP solver. We therefore restrict ourselves in this study to Benders cuts that can be formulated without internal information from the CP solver.

6 Minimizing Cost

The cost objective presents the simplest case, since cost can be computed solely in terms of master problem variables. The subproblem (12) takes the value $\sum_j F_{x_j j}$ when it is feasible and ∞ when infeasible. Let $J_{hi} = \{j \mid x_j^h = i\}$ be the set of tasks assigned to facility i in iteration h. If there is no feasible schedule for facility i, the most obvious Benders cut simply rules out assigning this same set J_{hi} of tasks to facility i. In this case the bounding function takes the form

$$B_{x^h}(x) = \begin{cases} \infty & \text{if the scheduling subproblem on some facility } i \text{ is infeasible, and } x_j^h = x_j \text{ for all } j \in J_{hi} \\ \displaystyle\sum_j F_{x_j j} & \text{otherwise} \end{cases} \tag{13}$$

The function B_{x^h} satisfies condition (B1) because the cost $\sum_j F_{x_j j}$ is equal to $B_{x^h}(x)$ for any feasible (x, s) and is infinite for any infeasible (x, s). It satisfies (B2) because $B_{x^h}(x^h)$ is the optimal value of the subproblem.

This bound is unnecessarily weak, however, since in many cases a proper subset of the tasks in J_{hi} are responsible for the infeasibility. Ideally the scheduler would keep track of which tasks actually play a role in the proof of infeasibility. Since this information is not available, a possibly smaller set $\bar{J}_{hi}$ of tasks that create infeasibility can be found by a simple greedy algorithm. Initially let $\bar{J}_{hi} = J_{hi} = \{j_1, \ldots, j_k\}$. For $\ell = 1, \ldots, k$ do the following: try to schedule the tasks in $\bar{J}_{hi} \setminus \{j_\ell\}$ on facility i, and if there is no feasible schedule, remove j_ℓ from $\bar{J}_{hi}$. $\bar{J}_{hi}$ can now replace J_{hi} in (13), and (B1) and (B2) remain satisfied. Computational testing suggests that, in most cases, re-solving the scheduling problem k times is well worth the time investment to obtain a stronger bound.

To write the master problem, let I_h be the set of facilities for which the scheduling problem is infeasible in iteration h. The master problem (10), written as a 0-1 programming problem, becomes

$$\begin{aligned} \text{minimize} \quad & \sum_{ij} F_{ij} x_{ij} \\ \text{subject to} \quad & \sum_i x_{ij} = 1, \text{ all } j & (a) \\ & \sum_{j \in \bar{J}_{hi}} (1 - x_{ij}) \geq 1, \text{ all } i \in I_h, \ h = 1, \ldots, H-1 & (b) \\ & \text{relaxation of subproblem} & (c) \end{aligned} \tag{14}$$

where $x_{ij} \in \{0, 1\}$, and where constraints (b) are the Benders cuts.

Experience shows that it is important to include a relaxation of the subproblem within the master problem. A straightforward relaxation can be obtained as follows. For any two times t_1, t_2, let $J(t_1, t_2)$ be the set of tasks j whose time windows lie between t_1 and t_2; that is, $[r_j, d_j] \subset [t_1, t_2]$. If the tasks $j \in J \subset J(t_1, t_2)$ are assigned to the same facility i, then clearly the "area" $\sum_{j \in J} p_{ij} c_{ij}$ of these tasks can be at most $C_i(t_2 - t_1)$ if they are to be scheduled in the time interval $[t_1, t_2]$. This yields the valid inequality

$$\frac{1}{C_i} \sum_{j \in J(t_1, t_2)} p_{ij} c_{ij} x_{ij} \leq t_2 - t_1, \tag{15}$$

which we refer to as inequality $R^i(t_1, t_2)$. If we let $\bar{r}_1, \ldots, \bar{r}_{n_r}$ be the distinct elements of $\{r_1, \ldots, r_n\}$ in increasing order, and similarly for $\bar{d}_1, \ldots, \bar{d}_{n_d}$, we have a relaxation consisting of the inequalities

$$R^i(\bar{r}_j, \bar{d}_k), \;\; j = 1, \ldots, n_r, \; k = 1, \ldots, n_d \tag{16}$$

for each facility i. These inequalities serve as the relaxation (c) in (14). There are doubtless stronger relaxations, but we require a linear relaxation that is expressed in terms of the variables x_{ij}.

Many of these inequalities may be redundant of the others, and if desired they can be omitted from the relaxation. Let

$$T^i(t_1, t_2) = \frac{1}{C_i} \sum_{j \in J(t_1, t_2)} p_{ij} c_{ij} - t_2 + t_1$$

measure the "tightness" of $R^i(t_1, t_2)$. It is easily verified that $R^i(t_1, t_2)$ dominates $R^i(u_1, u_2)$ whenever $[t_1, t_2] \subset [u_1, u_2]$ and $T^i(t_1, t_2) \geq T^i(u_1, u_2)$. Dominated inequalities can now be removed.

A set of undominated inequalities can be generated for each facility using the algorithm of Fig. 2. It has $O(n^3)$ complexity in the worst case, since it is possible that none of the inequalities are eliminated. This occurs, for instance, when each $r_j = j - 1$, $d_j = j$, and $p_{ij} = 2$. However, the algorithm need only be run once as a preprocessing routine.

In practice the relaxation can be simplified by supposing that the release times are all $r_0 = \min_j\{r_j\}$. Then the relaxation (c) in (14) consists of

$$R^i(r_0, \bar{d}_k), \;\; k = 1, \ldots, n_d$$

for each facility i. The redundant inequalities can be eliminated running the simple $O(n)$ algorithm of Fig. 3 for each facility. Similarly, one can suppose that the deadlines are all $d_0 = \max_j\{d_j\}$ and use the inequalities $R^i(\bar{r}_j, d_0)$.

Precedence constraints may be accommodated if they apply only to tasks assigned to the same facility. Thus if tasks j and k must be assigned to the same facility, and j must precede k, we add the constraint $x_j = x_k$ to the master problem (14) and $t_j + p_{ij} \leq t_j$ for all i to the subproblem (12).

Let $\mathcal{R}_i = \emptyset$.
For $j = 1, \ldots, p$:
  Set $k' = 0$.
  For $k = 1, \ldots, q$:
    If $r_k \geq r_j$ and $T^i(\bar{r}_j, \bar{d}_k) < T^i(\bar{r}_j, \bar{d}_{k'})$ then
      Remove from $\mathcal{R}_i$ all $R^i(\bar{r}_{j'}, \bar{d}_k)$ for which $T^i(\bar{r}_j, \bar{d}_k) \geq T^i(\bar{r}_{j'}, \bar{d}_k)$.
      Add $R^i(\bar{r}_j, \bar{d}_k)$ to $\mathcal{R}_i$ and set $k' = k$.

Figure 2: *$O(n^3)$ algorithm for generating an inequality set $\mathcal{R}_i$ that relaxes the time window constraints for facility i. By convention $\bar{d}_0 = -\infty$.*

Let $\mathcal{R}_i = \emptyset$, and set $j = 0$.
For $k = 1, \ldots, p_d$:
  If $T^i(r_0, \bar{d}_k) > T^i(r_0, \bar{d}_j)$ then add $R^i(r_0, \bar{d}_k)$ to $\mathcal{R}_i$ and set $j = k$.

Figure 3: *$O(n)$ algorithm for generating an inequality set $\mathcal{R}_i$ that relaxes the time window constraints for facility i, where $r_0 = \bar{d}_0 = \min_j\{r_j\}$ and $T^i(r_0, r_0) = 0$.*

7 Minimizing Makespan

This case is less straightforward because the subproblem is an optimization problem. However, there are relatively simple linear Benders cuts when all tasks have the same release date, and they simplify further when all deadlines are the same. We also use a linear subproblem relaxation that is valid for any set of time windows.

The Benders cuts are based on the following fact:

Lemma 2 *Consider a minimum makespan problem P in which tasks $1, \ldots, n$ with release time 0 and deadlines $d_1, \ldots, d_n$ are to be scheduled on a single facility i. Let M^* be the minimum makespan for P, and $\hat{M}$ the minimum makespan for the problem $\hat{P}$ that is identical to P except that tasks $1, \ldots, s$ are removed. Then*

$$M^* - \hat{M} \leq \Delta + \max_j\{d_j\} - \min_j\{d_j\} \tag{17}$$

where $\Delta = \sum_{j=1}^{s} p_{ij}$. In particular, when all the deadlines are the same, $M^* - \hat{M} \leq \Delta$.

Proof. Consider any optimal solution of $\hat{P}$ and extend it to a solution S of P by scheduling tasks $1, \ldots, s$ sequentially after $\hat{M}$. That is, for $k = 1, \ldots, s$ let task k start at time $\hat{M} + \sum_{j=1}^{k-1} p_{ij}$. The makespan of S is $\hat{M} + \Delta$. If $\hat{M} + \Delta \leq \min_j\{d_j\}$, then S is clearly feasible for P, so that $M^* \leq \hat{M} + \Delta$ and the lemma follows. Now suppose $\hat{M} + \Delta > \min_j\{d_j\}$. This implies

$$\hat{M} + \Delta + \max_j\{d_j\} - \min_j\{d_j\} > \max_j\{d_j\} \tag{18}$$

Since $M^* \leq \max_j\{d_j\}$, (18) implies (17), and again the lemma follows.

The bound $M^* - \hat{M} \leq \Delta$ need not hold when the deadlines differ. Consider for example an instance with three tasks where $(r_1, r_2, r_3) = (0, 0, 0)$, $(d_1, d_2, d_3) = (2, 1, \infty)$, $(p_{i1}, p_{i2}, p_{i3}) = (1, 1, 2)$, $(c_{i1}, c_{i2}, c_{i3}) = (2, 1, 1)$, and $C = 2$. Then if $s = 1$, we have $M^* - \hat{M} = 4 - 2 > \Delta = p_{i1} = 1$.

Now consider any given iteration of the Benders algorithm, and suppose for the moment that all the time windows are identical. Let J_{hi} be the set of tasks assigned to facility i in a previous iteration h, and M^*_{hi} the corresponding minimum makespan incurred by facility i. The solution of the current master problem removes task $j \in J_{hi}$ from facility i when $x_{ij} = 0$. Thus by Lemma 2, the resulting minimum makespan for facility i is reduced by at most

$$\sum_{j \in J_{hi}} p_{ij}(1 - x_{ij}) \tag{19}$$

and we therefore have a lower bound on the makespan of facility i:

$$M^*_{hi} - \sum_{j \in J_{hi}} p_{ij}(1 - x_{ij})$$

This yields a bounding function

$$B_{x^h}(x) = \max_i \left\{ M^*_{hi} - \sum_{j \in J_{hi}} p_{ij}(1 - x_{ij}) \right\} \tag{20}$$

that bounds the overall makespan and therefore satisfies (B1). The function satisfies (B2) as well because $B_{x^h}(x^h) = M^*_{hi}$, which is the optimal subproblem value.

As in the minimum cost problem, the bound (20) can be strengthened by finding a smaller set $\bar{J}_{hi}$ that results in the same solution of the scheduling problem. This is again done by a greedy algorithm. Initially let $\bar{J}_{hi} = J_{hi} = \{j_1, \ldots, j_k\}$. For $\ell = 1, \ldots, k$ do the following: find a minimum makespan schedule for the tasks in $\bar{J}_{hi} \setminus \{j_\ell\}$ on facility i, and if the resulting minimum makespan is M^*_{hi}, remove j_ℓ from $\bar{J}_{hi}$. Now $\bar{J}_{hi}$ can replace J_{hi} in (20).

We can write Benders cuts (b) in the master problem as follows:

$$\begin{aligned}
&\text{minimize} && M \\
&\text{subject to} && \sum_i x_{ij} = 1, \text{ all } j && (a) \\
& && M \geq M^*_{hi} - \sum_{j \in \bar{J}_{hi}} (1 - x_{ij}) p_{ij}, \text{ all } i,\ h = 1, \ldots, H-1 && (b) \\
& && M \geq \frac{1}{C_i} \sum_j c_{ij} p_{ij} x_{ij}, \text{ all } i && (c)
\end{aligned} \tag{21}$$

where $x_{ij} \in \{0, 1\}$. The relaxation (c) is similar to that for the minimum cost problem.

If the deadlines differ (and the release times still the same), the minimum makespan for facility i is at least

$$M_{hi}^* - \left(\sum_{j \in \bar{J}_{hi}} p_{ij}(1 - x_{ij}) + \max_{j \in \bar{J}_{hi}} \{d_j\} - \min_{j \in \bar{J}_{hi}} \{d_j\} \right) \tag{22}$$

if one or more tasks are removed, and is M_{hi}^* otherwise. The bounding function that results can be linearized to obtain the Benders cuts

$$\left.\begin{aligned} & M \geq M_{hi}^* - \sum_{j \in \bar{J}_{hi}} (1 - x_{ij}) p_{ij} - w_{hi} \\ & w_{hi} \leq \left(\max_{j \in \bar{J}_{hi}} \{d_j\} - \min_{j \in \bar{J}_{hi}} \{d_j\} \right) \sum_{j \in \bar{J}_{hi}} (1 - x_{ij}) \\ & w_{hi} \leq \max_{j \in \bar{J}_{hi}} \{d_j\} - \min_{j \in \bar{J}_{hi}} \{d_j\} \end{aligned}\right\} \quad \text{all } i,\ h = 1, \ldots, H-1 \tag{23}$$

When no tasks in $\bar{J}_{hi}$ are removed from facility i, the second constraint of (23) forces z_{hi} to zero, and the bound becomes M_{hi}^* as required by (B2). If one or more tasks are removed from facility i, the bound becomes (22) as desired. The Benders cuts (23) replace (21b) when the deadlines differ and all release times are equal.

8 Minimizing Total Tardiness

For this problem we again assume that all release times are zero. We consider two approaches to formulating Benders cuts, the first of which constructs cuts in a manner similar to those obtained for the minimum makespan problem. The following lemma provides the basis for these cuts:

Lemma 3 *Consider a minimum total tardiness problem P in which tasks $1, \ldots, n$ have release time 0, due dates $d_1, \ldots, d_n$, and are to be scheduled on a single facility i. Let T^* be the minimum tardiness for this problem, and $\hat{T}$ the minimum tardiness for the problem $\hat{P}$ that is identical to P except that tasks $1, \ldots, s$ are removed. Then*

$$T^* - \hat{T} \leq \sum_{k=1}^{s} \left(\sum_{j=1}^{n} p_{ij} - d_k \right)^+ \tag{24}$$

Proof. Consider any optimal solution $\hat{S}$ of $\hat{P}$. We may assume that the makespan of $\hat{S}$ is at most $M = \sum_{j=s+1}^{n} p_{ij}$, since if it is greater than M, we can move at least one task to

an earlier time without increasing the total tardiness of $\hat{S}$. To obtain a feasible solution S for P, schedule tasks $1, \ldots, s$ sequentially after M. That is, for $k = 1, \ldots, s$ let task k start at time $M + \sum_{j=1}^{k-1} p_{ij}$. The tardiness of task k in S is at most

$$\left(M + \sum_{j=1}^{s} p_{ij} - d_k \right)^+ = \left(\sum_{j=1}^{n} p_{ij} - d_k \right)^+$$

The total tardiness of S is therefore at most

$$\hat{T} + \sum_{k=1}^{s} \left(\sum_{j=1}^{n} p_{ij} - d_k \right)^+$$

from which (24) follows.

The proof uses the fact that the makespan $\hat{M}$ of $\hat{S}$ can be assumed to be at most M. A generally tighter bound could be obtained by assuming $\hat{M}$ is at most the makespan M^* of P's minimum tardiness solution. Unfortunately, this is not a valid assumption, since in exceptional cases $\hat{M} > M^*$ even though $\hat{P}$ has fewer tasks than P. Suppose for example that $n = 4$ and $s = 1$, with $r = (0, 0, 0, 0)$, $d = (5, 3, 3, 6)$, $p = (1, 2, 2, 4)$, $c = (2, 1, 1, 1)$, and $C = 2$. An optimal solution of P puts $t = (4, 0, 2, 0)$ with tardiness $T^* = 1$ and makespan $M^* = 5$, but the only optimal solution of the smaller problem $\hat{P}$ puts $(t_2, t_3, t_4) = (0, 0, 2)$ with tardiness $\hat{T} = 0$ and makespan $\hat{M} = 6 > M^*$.

Lemma 3 is applied as follows. As before let J_{hi} be the set of tasks assigned to facility i in a previous iteration h, and T^*_{hi} the corresponding minimum tardiness incurred by facility i. Since task j is removed from facility i when $x_{ij} = 0$, the resulting minimum tardiness on facility i is at least

$$T^*_{hi} - \sum_{k \in J_{hi}} \left(\sum_{j \in J_{hi}} p_{ij} - d_k \right)^+ (1 - x_{ik})$$

This yields the Benders cuts (b) in the master problem

$$\begin{aligned}
&\min \ T \\
&\text{s.t.} \ \sum_i x_{ij} = 1, \ \text{all } j && (a) \\
&\qquad T \geq \sum_i \left(T^*_{hi} - \sum_{k \in J_{hi}} \left(\sum_{j \in J_{hi}} p_{ij} - d_k \right)^+ (1 - x_{ik}) \right), \ \text{all } i, \ h = 1, \ldots, H-1 && (b) \\
&\qquad \text{relaxation of subproblem} && (c)
\end{aligned}$$

Unfortunately these cuts are weak and do not perform well in practice, due to the lack of a tighter bound on $\hat{M}$. It appears that stronger cuts require some internal information from the CP solver.

A second approach is to obtain stronger cuts by re-solving the subproblem repeatedly, as done for the min cost and min makespan problems. Several schemes are possible, but we found the following to be effective. Let $T_i(J)$ be the minimum tardiness on facility i that results when the tasks in J are assigned to facility i, so that $T_i(J_{hi}) = T^*_{hi}$. Let Z_{hi} be the set of tasks in J_{hi} that can be removed, one at a time, without reducing the min tardiness. That is,

$$Z_{hi} = \{j \in J_{hi} \mid T_i(J_{hi} \setminus \{j\}) = T^*_{hi}\}$$

Finally, let T^0_{hi} be the minimum tardiness that results from removing all the tasks in Z_{hi} at once, so that $T^0_{hi} = T_i(J_{hi} \setminus Z_{hi})$. Thus any or all tasks in Z_{hi} can be removed from facility i without reducing the min makespan below T^0_{hi}. This yields a bounding function $B_{x^h}(x) = \sum_i B^i_{x^h}(x)$, where

$$B^i_{x^h}(x) = \begin{cases} T^0_{hi} & \text{if } Z_{hi} \supset \{j \in J_{hi} \mid x_j \neq i\} \neq \emptyset \quad (a) \\ T^*_{hi} & \text{if } \{j \in J_{hi} \mid x_j \neq i\} = \emptyset \quad (b) \\ 0 & \text{otherwise} \quad (c) \end{cases} \tag{25}$$

The function $B_{x^h}(x)$ satisfies (B1) by construction. Case (b) of (25) ensures that it satisfies (B2) as well.

The master problem MILP now becomes

$$\begin{aligned} \text{minimize} \quad & T \\ \text{subject to} \quad & \sum_i x_{ij} = 1, \text{ all } j && (a) \\ & T \geq \sum_i T_i && (b) \\ & T_i \geq T^0_{hi} - T^0_{hi} \sum_{j \in J_{hi} \setminus Z_{hi}} (1 - x_{ij}), \quad \text{all } i,\ h = 1, \ldots, H-1 && (c1) \\ & T_i \geq T^*_{hi} - T^*_{hi} \sum_{j \in J_{hi}} (1 - x_{ij}), \quad \text{all } i,\ h = 1, \ldots, H-1 && (c2) \\ & \text{relaxation of subproblem} && (d) \end{aligned} \tag{26}$$

The cut (c2) is redundant and can be eliminated for a given h, i when $T^0_{hi} = T^*_{hi}$. This in fact substantially reduces the size of master problem, since computational testing suggests that $T^0_{hi} = T^*_{hi}$ very often.

We employed two relaxations for the master problem (26), since they seem to have somewhat complementary strengths. The first and simpler relaxation is again based on the

"areas" required by tasks. Recall that $J(t_1, t_2)$ is the set of jobs with time windows between t_1 and t_2.

Lemma 4 *Consider a minimum total tardiness problem in which tasks $j = 1, \ldots, n$ with time windows $[r_j, d_j]$ are scheduled on a single facility i, where $\min_j\{r_j\} = 0$. The total tardiness incurred by any feasible solution is bounded below by*

$$\left(\frac{1}{C_i} \sum_{j \in J(0,d_k)} p_{ij} c_{ij} - d_k \right)^+$$

for each $k = 1, \ldots, n$.

Proof. For any k, the last scheduled task in the set $J(0, d_k)$ can finish no earlier than time $\hat{T} = \frac{1}{C_i} \sum_{j \in J(0,d_k)} p_{ij} c_{ij}$. Since the last task has due date no later than d_k, its tardiness is no less than $(\hat{T} - d_k)^+$. Thus total tardiness is no less than $(\hat{T} - d_k)^+$.

This gives rise to a relaxation consisting of

$$\begin{aligned} & T \geq \sum_i T_i^L \\ & T_i^L \geq \frac{1}{C_i} \sum_{j \in J(0,d_k)} p_{ij} c_{ij} x_{ij} - d_k, \quad \text{all } i, k \\ & T_i^L \geq 0, \quad \text{all } i \end{aligned} \tag{27}$$

A second relaxation of the subproblem can be developed on basis of the following lemma. For each facility i let π_i be a permutation of $\{1, \ldots, n\}$ such that $p_{i\pi_i(1)} c_{i\pi_i(1)} \leq \cdots \leq p_{\pi_i(n)} c_{\pi_i(n)}$.

Lemma 5 *Consider a minimum tardiness problem in which tasks $1, \ldots, n$ with time windows $[r_j, d_j]$ are scheduled on a single facility i. Assume $\min_j\{r_j\} = 0$ and index the tasks so that $d_1 \leq \cdots \leq d_n$. Then the total tardiness T of any feasible solution is bounded below by $\underline{T} = \sum_{k=1}^n \underline{T}_k$, where*

$$\underline{T}_k = \left(\frac{1}{C_i} \sum_{j=1}^k p_{i\pi_i(j)} c_{i\pi_i(j)} - d_k \right)^+, \quad k = 1, \ldots, n$$

Proof. Consider any feasible solution of the one-facility minimum tardiness problem, in which tasks $1, \ldots, n$ are respectively scheduled at times $t_1, \ldots, t_n$. The minimum tardiness is

$$T^* = \sum_{k=1}^n (t_k + p_{ik} - d_k)^+ \tag{28}$$

Let $\sigma_0(1), \ldots, \sigma_0(n)$ be the order in which tasks are scheduled in this solution, so that $t_{\sigma_0(1)} \leq \cdots \leq t_{\sigma_0(n)}$. For an arbitrary permutation σ of $\{1, \ldots, n\}$ let

$$\underline{T}_k(\sigma) = \left(\frac{1}{C_i} \sum_{j=1}^{k} p_{i\pi_i(j)} c_{i\pi_i(j)} - d_{\sigma(k)} \right)^+ \tag{29}$$

and $\underline{T}(\sigma) = \sum_{k=1}^{n} \underline{T}_k(\sigma)$.

We show first that $T^* \geq \underline{T}(\sigma_0)$. Since σ_0 is a permutation we can write (28) as

$$T^* = \sum_{k=1}^{n} \left(t_{\sigma_0(k)} + p_{i\sigma_0(k)} - d_{\sigma_0(k)} \right)^+$$

We observe that

$$T^* \geq \sum_{k=1}^{n} \left(\frac{1}{C_i} \sum_{j=1}^{k} p_{i\sigma_0(j)} c_{i\sigma_0(j)} - d_{\sigma_0(k)} \right)^+ \geq \sum_{k=1}^{n} \left(\frac{1}{C_i} \sum_{j=1}^{k} p_{i\pi_0(j)} c_{i\pi_0(j)} - d_{\sigma_0(k)} \right)^+ = \underline{T}(\sigma_0)$$

where the first inequality is based on the areas required by tasks, and the second inequality is due to the definition of π_i.

Now suppose a bubble sort is performed on the integers $\sigma_0(1), \ldots, \sigma_0(n)$ so as to put them in increasing order, and let $\sigma_0, \ldots, \sigma_P$ be the resulting series of permutations. Thus $(\sigma_P(1), \ldots, \sigma_P(n)) = (1, \ldots, n)$, and σ_{p+1} is obtained from σ_p by swapping two adjacent terms $\sigma_p(k)$ and $\sigma_p(k+1)$, where $\sigma_p(k) > \sigma_p(k+1)$. This means σ_p and σ_{p+1} are the same except that $\sigma_{p+1}(k) = \sigma_p(k+1)$ and $\sigma_{p+1}(k+1) = \sigma_p(k)$. Since $T^* \geq \underline{T}(\sigma_0)$ and $\underline{T}(\sigma_P) = \underline{T}$, to prove the theorem it suffices to show $\underline{T}(\sigma_0) \geq \cdots \geq \underline{T}(\sigma_P)$.

Thus we consider any two adjacent permutations σ_p, σ_{p+1} and show that $\underline{T}(\sigma_p) \geq \underline{T}(\sigma_{p+1})$. We observe that

$$\begin{aligned} \underline{T}(\sigma_p) &= \sum_{j=1}^{k-1} \underline{T}_j(\sigma_p) + \underline{T}_k(\sigma_p) + \underline{T}_{k+1}(\sigma_p) + \sum_{j=k+2}^{n} \underline{T}_j(\sigma_p) \\ \underline{T}(\sigma_{p+1}) &= \sum_{j=1}^{k-1} \underline{T}_j(\sigma_p) + \underline{T}_k(\sigma_{p+1}) + \underline{T}_{k+1}(\sigma_{p+1}) + \sum_{j=k+2}^{n} \underline{T}_j(\sigma_p) \end{aligned} \tag{30}$$

Using (29), we note that $\underline{T}_k(\sigma_p) = (a - B)^+$, $\underline{T}_{k+1}(\sigma_p) = (A - b)^+$, $\underline{T}_k(\sigma_{p+1}) = (a - b)^+$, and $\underline{T}_{k+1}(\sigma_{p+1}) = (A - B)^+$ if we set

$$\begin{aligned} a &= \frac{1}{C_i} \sum_{j=1}^{k} p_{i\pi_i(j)} c_{i\pi_i(j)}, \quad A = \frac{1}{C_i} \sum_{j=1}^{k+1} p_{i\pi_i(j)} c_{i\pi_i(j)} \\ b &= d_{\sigma_p(k+1)}, \quad B = d_{\sigma_p(k)} \end{aligned}$$

Note that $a \leq A$. Also, $b \leq B$ since $\sigma_p(k) > \sigma_p(k+1)$ and $d_1 \leq \cdots \leq d_n$. From (30) we have

$$\underline{T}(\sigma_p) - \underline{T}(\sigma_{p+1}) = (a-B)^+ + (A-b)^+ - (a-b)^+ - (A-B)^+$$

It is straightforward to check that this quantity is always nonnegative when $a \leq A$ and $b \leq B$. The theorem follows.

The bound of Lemma 5 can be written in terms of the variables x_{ik}:

$$\sum_{k=1}^{n} \underline{T}'_{ik} x_{ik} \tag{31}$$

where

$$\underline{T}'_{ik} \geq \frac{1}{C_i} \sum_{j=1}^{k} p_{i\pi_i(j)} c_{i\pi_i(j)} x_{i\pi_i(j)} - d_k, \quad k = 1, \ldots, n$$

and $\underline{T}'_{ik} \geq 0$. The variable x_{ik} appears in (31) because the term T'_{ik} should occur only for those tasks j that are assigned to facility i (i.e., $x_{ik} = 1$). We linearize the bound by writing it as

$$\sum_{k=1}^{n} \underline{T}_{ik} \tag{32}$$

where

$$\underline{T}_{ik} \geq \frac{1}{C_i} \sum_{j=1}^{k} p_{i\pi_i(j)} c_{i\pi_i(j)} x_{i\pi_i(j)} - d_k - (1 - x_{ik}) U_{ik}, \quad k = 1, \ldots, n \tag{33}$$

and $\underline{T}_{ik} \geq 0$. The big-M term U_{ik} is given by

$$U_{ik} = \frac{1}{C_i} \sum_{j=1}^{k} p_{i\pi_i(j)} c_{i\pi_i(j)} - d_k$$

Note that although U_{ik} can be negative, the right-hand side of (33) is never positive when $x_{ik} = 0$. Finally, to obtain a relaxation of the subproblem, we sum (32) over all facilities and write

$$T \geq \sum_{i=1}^{m} \sum_{k=1}^{n} \underline{T}_{ik} \tag{34}$$

The relaxation (c) of the master problem now consists of two relaxations, the first of which is (27), and the second (34) together with (33) for $i = 1, \ldots, m$. The second relaxation is valid only when tasks are indexed so that $d_1 \leq \cdots \leq d_n$.

9 Problem Generation

Random instances were generated as follows for minimum cost and minimum makespan problems. The capacity limit was set to $C_i = 10$ for each facility i. For each task j, c_{ij} was assigned the same random value for all facilities i and drawn from a uniform distribution on $[1, 10]$. For instances with n tasks and m facilities, the processing time p_{ij} of each task j on facility i was drawn from a uniform distribution on $[i, 10i]$. Thus facility 1 tends to run about i times faster than facility i for $i = 1, \ldots, m$. Since the average of $10i$ over m facilities is $5(m + 1)$, the total processing time of all tasks is roughly proportional to $5n(m + 1)$, or about $5n(m + 1)/m$ per facility. The release dates were set to zero, and the deadline for every task was set to αL (rounded to the nearest integer), where $L = 5n(m + 1)/m$. We used $\alpha = \frac{1}{3}$ for the minimum cost and minimum makespan problem, which results in a deadline that is loose enough to permit feasible solutions but tight enough so that tasks are reasonably well distributed over the facilities in minimum cost solutions. In minimum cost problems, the cost F_{ij} is drawn from a uniform distribution on $[2(m-i+1), 20(m-i+1)]$, so that faster facilities tend to be more expensive.

No precedence constraints were used, which tends to make the scheduling portion of the problem more difficult. The wide variation in facility speeds tends to put the Benders approach at a disadvantage, since the faster facilities receive more tasks and require disproportionately more computation to solve the scheduling problem.

For minimum tardiness problems we set the release dates to zero and drew the due dates from a uniform distribution on the interval $[\frac{1}{4}\alpha L, \alpha L]$, again using $\alpha = \frac{1}{3}$. We found that when the processing time for facility i is drawn from $[i, 10i]$, the wide range of facility speeds often results in solutions in which nearly all tasks are assigned to the fastest facility. We therefore generated problems on three facilities in which the processing time is drawn from a uniform distribution on $[2, 20]$, $[2, 25]$ and $[2, 30]$ for the three facilities, respectively. For 22 or more tasks we used the intervals $[5, 20]$, $[5, 25]$ and $[5, 30]$ since otherwise the minimum tardiness tends to be zero in the larger problems.

To test the min cost and min makespan methods on problems with deadlines, we generated instances similar to the min tardiness problems but with $\alpha = \frac{1}{2}$ rather than $\alpha = \frac{1}{3}$ to avoid infeasible instances. We also generated instances in which the release times differ for each task. In these instances, the release time r is drawn uniformly from the interval $[0, \alpha L]$, and the deadline is drawn uniformly from $[r + \frac{1}{4}\alpha L, r + \alpha L]$.

10 Computational Results

We solved randomly generated problems with MILP (using CPLEX), CP (using the ILOG Scheduler), and the logic-based Benders method. All three methods were implemented with OPL Studio, using the OPL script language. The CP problems were solved with the

assignAlternatives and setTimes options, and the CP subproblems of the Benders method were solved with the setTimes option. Both result in substantially better performance.

Table 1 displays computational results for minimum cost and minimum makespan problems on 2, 3 and 4 facilities. The CP solver is consistently faster than MILP on minimum cost problems. However, CP is unable to solve most problems with more than 16 tasks within two hours of computation time.

The Benders method is substantially faster than both CP and MILP. Its advantage increases rapidly with problem size, reaching some three orders of magnitude relative to CP for 16 or 18 tasks. Presumably the advantage would be greater for larger problems.

On minimum makespan problems, CP and MILP appear to be roughly competitive overall, with CP generally faster on 2-facility problems and MILP on 3 and 4-facility problems. Again the Benders method is considerably faster that both CP and MILP, with speedups ranging between two and three orders of magnitude for problems small enough to be solved by CP or MILP. As before, the speedup increases with the problem size.

As the number of tasks increases, the Benders subroblems in minimum cost and makespan problems eventually reach a size at which the computation time for the scheduling subproblem explodes. This point is reached later when there are more facilities, since the subproblems are smaller when the tasks are spread over more facilities. In practice, precedence constraints or other side constraints could accelerate the solution of the scheduling subproblems and allow the solution of larger problems.

Table 2 reports tests of CP, MILP and Benders on (a) minimum cost problems in which the tasks have different deadlines, (b) minimum cost problems in which they have different release times and different deadlines, and (c) minimum makespan problems in which the tasks have different deadlines. MILP is generally faster than CP on these problems.

The advantage of the Benders method remains substantial. It is at least as great as before on problems (b). MILP terminated due to insufficient memory on several of the problems (a), thus removing the instances on which it is likely to run longest. The speedup appears somewhat less than before on problems (c), but this is due primarily to a single outlier with 20 tasks that Benders could not solve in two hours.

Table 3 investigates how the Benders method scales up to a larger number of facilities when the average number of tasks per facility is fixed to 5. The random instances are generated so that the fastest facility is roughly twice as fast as the slowest facility, with the other facility speeds spaced evenly in between. Since the subproblems remain relatively small as the problem size increases, it is possible to accommodate more tasks than in Tables 1 and 2. In this case, the master problem (rather than the subproblem) dominates the computational task in larger problems and eventually becomes too hard to solve.

Table 3 suggests that the Benders method scales up better for minimum cost problems than for minimum makespan problems. Yet even when it fails to solve a makespan problem optimally, it obtains a feasible solution and a fairly tight lower bound on the optimal value. The bound steadily improve as the algorithm runs. Table 4 displays the bounds obtained

Table 1: Computation times in seconds for minimum cost and minimum makespan problems, using MILP, CP, and logic-based Benders methods. Each time represents the average of 5 instances. Computation was cut off after two hours (7200 seconds), and a + indicates that this occurred for at least one of the five problems. The test problems are cNjMmK, where N is the number of tasks, M the number of facilities, and K the problem number ($K = 1, \ldots, 5$).

Facilities	Tasks	Min cost			Min makespan		
		MILP	CP	Benders	MILP	CP	Benders
2	10	1.9	0.14	0.08	3.4	0.80	0.24
	12	199	2.2	0.11	12	4.0	0.31
	14	1441	79	0.08	2572+	299	5.0
	16	3604+[1]	1511	1.8	5794+	3737	36
	18		7200+	9.0		7200+	233
	20			111			1268
	22			1805+			
3	10	0.86	0.13	0.09	3.9	0.85	0.23
	12	797	2.6	0.19	12	7.5	0.38
	14	114	35	0.20	524	981	1.4
	16	678[1]	1929	1.2	1716+	4414	7.6
	18		7200+	2.8	4619+	7200+	30
	20			2.5			8.7
	22			7.6			2012+
	24			19			
4	10	2.0	0.10	0.09	1.0	0.07	0.19
	12	7.2	1.4	0.15	5.0	1.9	0.43
	14	158	72	0.36	24	524	0.82
	16	906[1]	344	0.25	35	3898	1.0
	18		6343+	0.87	3931+	7200+	6.4
	20			0.96			4.4
	22			3.3			28
	24			39			945
	26			29			

[1]CPLEX ran out of memory on one or more problems, which are omitted from the average time.

Table 2: Computation times in seconds for minimum cost and minimum makespan problems, using MILP and the Benders method. Each time represents the average of 5 instances. Computation was cut off after two hours (7200 seconds), and a + indicates that this occurred for at least one of the five problems. The problems with different deadlines are de$N$jMmK, and those with different deadlines and release times are df$N$jMmK.

Tasks	Min cost Different deadlines			Min cost Different deadlines and release times			Min makespan Different deadlines		
	CP	MILP	Benders	CP	MILP	Benders	CP	MILP	Benders
14	1.2	93	0.17	0.95	0.82	0.25	7.1	223	4.4
16	676	6.7	1.8	7.2	1.0	0.28	1620+	853	5.1
18	47	230	0.28	69	4.9	0.75	1928+	350	2.9
20	5006+	87	2.7	55	31	0.82	7200+	7200+	1449+
22	5596+	104[1]	1.0	1180	1514+	1.9		7200+	388
24	7200+	744[1]	10.2	7200+	1924+	4.6		7200+	132
26		3616[3]	1560+	7200+	375	8.9			2930+
28		4102[2]	1681+		3032+	4.3			2709+

[1]CPLEX ran out of memory on one problem, which is omitted from the average time.
[2]CPLEX ran out of memory on two problems, which are omitted from the average time.
[3]CPLEX ran out of memory on three problems, which are omitted from the average time.

for the minimum makespan problems of Table 3 that were not solved to optimality.

Table 5 shows computational results for minimum tardiness problems on three facilities using CP, MILP and Benders. Since problem difficulty tends to increase rapidly with minimum tardiness, the table shows results for individual problem instances and indicates the minimum tardiness for each. The instances are listed in order of minimum tardiness for each problem size.

The minimum tardiness problems are more readily solved by MILP than CP. Nonetheless the Benders method is faster than MILP for nearly all problem instances. The advantage of the hybrid approach is not as great as for minimum cost and minimum makespan problems. Yet for larger problems (16 or more tasks) that the Benders method solved to optimality, the average speedup factor relative to MILP is 25. This is actually an underestimate, since the MILP solver was cut off after two hours. Table 6 shows that the relaxations based on Lemmas 4 and 5 are key to the success of the Benders method.

Despite its better performance, the Benders method does not significantly enlarge the class of tardiness problems that can be solved to optimality. It failed to solve 6 of the 40 instances to optimality, only a modest improvement over the 10 that were intractable for MILP. On problems too large solve optimally, however, it finds much better solutions

Table 3: Computation times in seconds for minimum cost and minimum makespan problems, using the Benders method. Each time represents the average of 5 instances. Computation was cut off after two hours (7200 seconds), and a + indicates that this occurred for at least one of the five problems. The test problems are eNjMmK.

Tasks	Facilities	Min cost	Min makespan
10	2	0.1	0.2
15	3	0.3	1.6
20	4	3.2	32
25	5	3.3	28
30	6	1.4	65
35	7	8.0	767
40	8	157	5944+
45	9	95	5762+
50	10	19	

than MILP finds within two hours. The Benders method also finds a lower bound on the optimal value that steadily increases with the number of iterations. The lower bounds are rather weak for the instances solved here, but they are tighter than the lower bound of zero obtained from the continuous relaxation of the MILP model.

11 Conclusions and Future Research

We find that logic-based Benders decomposition can improve substantially on the state of the art for the solution of planning and scheduling problems. In the case of minimum cost and minimum makespan problems, the Benders method is several orders of magnitude faster than either CP or MILP and solves larger problem instances to optimality.

The speedup is not nearly as great on minimum tardiness problems but significant nonetheless. The Benders method does not solve significantly larger problems to optimality but obtains much better solutions than MILP for instances too large to solve optimally.

In fact, an important advantage of the Benders approach is that it can be terminated early while still yielding a feasible solution, as well as a lower bound on the optimal value that improves steadily as the algorithm runs. (This does not apply to the minimum cost problems, for which all intermediate solutions are infeasible.)

Future research should address several issues. One is whether effective Benders cuts can be developed for minimum makespan and minimum tardiness problems in which tasks have different release dates. In addition, implementation of a branch-and-check approach to solving the master problem could significantly improve performance, particularly on problems in which the master problem dominates the computational burden.

Table 4: Best solution value and lower bound found after 7200 seconds of computation by the Benders method on the minimum makespan problems of Table 3 that were not solved to optimality. The test problems are eNjMmK, where K is the problem number shown in the last column.

Tasks	Facilities	Lower bound	Best solution value	Prob. no.
40	8	12	14	1
40	8	13	15	2
40	8	15	17	3
40	8	12	13	5
45	9	13	15	1
45	9	13	16	2
45	9	10	12	3
45	9	12	16	5

A fundamental issue is whether access to "dual" information from the CP scheduler (results of edge finding, etc.) would result in more effective Benders cuts, and how these cuts would be derived. This is particularly important for the tardiness problem, where the Benders cuts used here are rather weak.

A related issue is how to incorporate strong cuts in the master problem without generating a large number of constraints. Computational experience with the tardiness problems, for instance, suggests that stronger cuts could have been generated with little additional effort by solving a greater number of modified subproblems in each iteration. Each of the strong cuts, however, involves a logical condition that requires a large number of linear inequalities (many with "big-M" coefficients) in order to formulate it in the MILP master problem. Solution of the master problem bogs down as the cuts accumulate.

This suggests that rather than solving the master problem in the form of an MILP, it may be more effective to apply an algorithm that accommodates more complex logical constraints. This in combination with a branch-and-check strategy could result in a more robust class of Benders methods.

12 References

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

Cambazard, H., P.-E. Hladik, A.-M. Déplanche, N. Jussien, and Y. Trinquet. 2004. Decomposition and learning for a hard real time task allocation problem, in M. Wallace, ed.,

Table 5: Computational results for minimum tardiness problems on three facilities. Computation is terminated after two hours (7200 seconds). The problems are $ddNjMmK$, where K is the problem number shown in the last column.

Tasks	Time (sec)			Benders/	Best solution value found[1]		Benders	Prob.
	CP	MILP	Benders	MILP speedup	MILP	Benders	lower bound[2]	no.
10	13	4.7	2.6	1.8	10	10		2
	1.1	6.4	1.6	4.0	10	10		1
	1.4	6.4	1.6	4.0	16	16		4
	4.6	32	4.1	7.8	17	17		5
	8.1	33	22	1.5	24	24		3
12	4.7	0.7	0.2	3.5	0	0		5
	14	0.6	0.1	6.0	0	0		1
	25	0.7	0.2	3.5	1	1		3
	19	15	2.4	6.3	9	9		4
	317	25	12	2.1	15	15		2
14	838	7.0	6.1	1.2	1	1		2
	7159	34	3.7	9.2	2	2		3
	1783	45	19	2.4	15	15		5
	>7200	73	40	1.8	19	19		1
	>7200	>7200	3296	>2.2	(26)	26		4
16	>7200	19	1.4	14	0	0		2
	>7200	46	2.1	22	0	0		5
	>7200	52	4.2	12	4	4		4
	>7200	1105	156	7.1	20	20		3
	>7200	3424	765	4.5	31	31		1
18		187	2.8	67	0	0		5
		15	5.3	2.8	3	3		4
		46	49	0.9	5	5		3
		256	47	5.5	11	11		1
		>7200	1203	>6.0	(14)	11		2
20		105	18	5.8	0	0		1
		4141	23	180	1	1		5
		39	29	1.3	4	4		2
		1442	332	4.3	8	8		3
		>7200	>7200		(75)	(37)	9	4
22		6.3	19	0.3	0	0		4
		584	37	16	2	2		1
		>7200	>7200		(120)	(40)	7	3
		>7200	>7200		(162)	(46)	11	5
		>7200	>7200		(375)	(141)[3]	34	2
24		10	324	0.03	0	0		3
		>7200	94	>77	(20)	0		5
		>7200	110	>65	(57)	0		4
		>7200	>7200		(20)	(5)	3	2
		>7200	>7200		(25)	(7)	1	1

[1]Values in parentheses were not proved optimal.
[2]When omitted, the lower bound is equal to the optimal value shown in the previous column.
[3]Best known solution is 128, obtained using a slightly weaker relaxation.

Table 6: Effect of the subproblem relaxation on performance of the Benders method. Computation time in seconds is shown. The problems are $ddNjMmK$, where K is the problem number shown in the last column.

Tasks	Minimizing tardiness: with relaxation	without relaxation	Prob. no.
16	1.4	4.4	2
	2.1	6.5	5
	4.2	30	4
	156	199	3
	765	763	1
18	2.8	10	5
	5.3	17	4
	47	120	1
	49	354	3
	1203	5102	2
20	18	151	1
	23	1898	5
	29	55	2
	332	764	3
	>7200	>7200	4

Principles and Practice of Constraint Programming (CP2004), *Lecture Notes in Computer Science* **3258**, Springer, 153–167.

Corréa, A. I., A. Langevin, and L. M. Rousseau. 2004. Dispatching and conflict-free routing of automated guided vehicles: A hybrid approach combining constraint programming and mixed integer programming, in J. C. Régin and M. Rueher, eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004)*, *Lecture Notes in Computer Science* **3011**, Springer, 370–378.

Chu, Y., and Q. Xia. 2004. Generating Benders cuts for a class of integer programming problems, in J. C. Régin and M. Rueher, eds, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004)*, *Lecture Notes in Computer Science* **3011**, Springer, 127–141.

Chu, Y., and Q. Xia. 2005. A hybrid algorithm for a class of resource-constrained scheduling problems, in R. Barták and M. Milano, eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2005)*, *Lecture Notes in Computer Science* **3524**, Springer, 110–124.

Eremin, A., and M. Wallace. 2001. Hybrid Benders decomposition algorithms in constraint logic programming, in T. Walsh, ed., *Principles and Practice of Constraint Programming*

(CP 2001), *Lecture Notes in Computer Science* **2239**, Springer.

Geoffrion, A. M. 1972. Generalized Benders decomposition, *Journal of Optimization Theory and Applications* **10**: 237–260.

Harjunkoski, I., and I. E. Grossmann. 2001. A Decomposition approach for the scheduling of a steel plant production, *Computers and Chemical Engineering* **25**, 1647–1660.

Harjunkoski, I., and I. E. Grossmann. 2002. Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods, *Computers and Chemical Engineering* **26**, 1533–1552.

Hooker, J. N. 2000. *Logic-based Methods for Optimization: Combining Optimization and Constraint Satisfaction*, John Wiley & Sons.

Hooker, J. N. 2004. A hybrid method for planning and scheduling, in M. Wallace, ed., *Principles and Practice of Constraint Programming (CP 2004)*, *Lecture Notes in Computer Science* **3258**, Springer, 305–316.

Hooker, J. N. 2005. Planning and scheduling to minimize tardiness, in P. van Beek, ed., *Principles and Practice of Constraint Programming (CP 2005)*, *Lecture Notes in Computer Science* **3709**, Springer, 314–327.

Hooker, J. N. and G. Ottosson. 2003. Logic-based Benders decomposition, *Mathematical Programming* **96**: 33–60.

Hooker, J. N., and Hong Yan. 1995. Logic circuit verification by Benders decomposition, in V. Saraswat and P. Van Hentenryck, eds., *Principles and Practice of Constraint Programming: The Newport Papers*, MIT Press (Cambridge, MA) 267–288.

Jain, V., and I. E. Grossmann. 2001. Algorithms for hybrid MILP/CP models for a class of optimization problems, *INFORMS Journal on Computing* **13**: 258–276.

Maravelias, C. T., I. E. Grossmann. 2004. Using MILP and CP for the scheduling of batch chemical processes, in J. C. Régin and M. Rueher, eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004)*, *Lecture Notes in Computer Science* **3011**, Springer, 1–20.

Thorsteinsson, E. S. 2001. Branch-and-Check: A hybrid framework integrating mixed integer programming and constraint logic programming, *Lecture Notes in Computer Science* **2239**: 16–30.

Timpe, C. 2002. Solving planning and scheduling problems with combined integer and constraint programming, *OR Spectrum* **24**, 431–448.

Türkay, M., and I. E. Grossmann. 1996. Logic-based MINLP algorithms for the optimal synthesis of process networks, *Computers and Chemical Engineering* **20**: 959–978.