An Integrated Method for Planning and Scheduling to Minimize Tardiness

J. N. Hooker

Carnegie Mellon University
john@hooker.tepper.cmu.edu

Abstract. We combine mixed integer linear programming (MILP) and constraint programming (CP) to minimize tardiness in planning and scheduling. Tasks are allocated to facilities using MILP and scheduled using CP, and the two are linked via logic-based Benders decomposition. We consider two objectives: minimizing the number of late tasks, and minimizing total tardiness. Our main theoretical contribution is a relaxation of the cumulative scheduling subproblem, which is critical to performance. We obtain substantial computational speedups relative to the state of the art in both MILP and CP. We also obtain much better solutions for problems that cannot be solved to optimality.

We address a 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 due dates. Tasks assigned to a given facility may run in parallel if desired, subject to a resource constraint (cumulative scheduling). We consider two objectives: minimizing the number of late tasks, and minimizing total tardiness.

The problem can be formulated entirely as a constraint programming (CP) problem or a mixed integer/linear programming (MILP) problem. However, these models are hard to solve. By linking CP and MILP in an integrated method, we obtain significant speedups relative to the state of the art in both MILP and CP. The linkage is achieved by logic-based Benders decomposition. The facility assignment problem becomes the master problem and is solved by MILP, while the scheduling problem becomes the subproblem (slave problem) and is solved by CP.

The primary theoretical contribution of this paper is a linear relaxation of the cumulative scheduling subproblem. We find that including such a relaxation in the master problem is essential to the success of the Benders method.

We solve problem instances in which tasks have the same release date and different due dates, although the method is valid for different release dates as well. The method also accommodates precedence constraints between tasks that are constrained to run on the same facility.

We obtain substantial speedups on nearly all instances relative to MILP (as represented by CPLEX), which in turn is generally faster than CP (as represented by the ILOG Scheduler). On larger instances, the integrated method

generally achieves speedups of two or three orders of magnitude when minimizing the number of late tasks, and it solves significantly more problems to optimality. There is a lesser but still significant speedup when minimizing total tardiness, and even when the hybrid method cannot obtain provably optimal solutions, it obtains much better solutions than provided by MILP in the same time period.

1 Previous Work

Classical Benders decomposition [1, 6] 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 this paper, the primary variables define the allocation of tasks to facilities.) Solution of the subproblem generates a nogood or *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 a subproblem dual. The next set of values of 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 the subproblem converge in value.

The classical Benders approach is inappropriate for the planning and scheduling 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.

Logic-based Benders decomposition was introduced in [13] for purposes of logic circuit verification. The idea was later formally developed in [8] and applied to 0-1 programming in [12]. In logic-based Benders, the Benders cuts are obtained by solving an "inference dual" of the subproblem, of which the linear programming dual is a special case. Although logic-based Benders cuts may take any form, in the present context they must be linear inequalities, since the master problem is an MILP.

The application of logic-based Benders to planning and scheduling was proposed in [8]. Jain and Grossmann [15] successfully applied such a method to minimum-cost planning and scheduling problems in which the subproblems are disjunctive scheduling problems, where tasks must run one at a time, rather than cumulative scheduling problems. The Benders cuts are particularly simple in this case because the subproblem is a feasibility problem rather than an optimization problem. These results are extended to multistage problems in [7].

The results of [15] can be improved upon using a "branch-and-check" method, proposed in [8] and applied by [17] to the instances solved in [15]. Branch and check solves the master problem only once, updating the solution each time a Benders cut is generated. 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.

It is less obvious how to define Benders cuts when the subproblem is an optimization problem. We showed in [9] how to derive effective Benders cuts for at least one such case, minimum makespan problems. The cuts are valid for cumulative as well as disjunctive scheduling, provided all tasks have the same release date. Computational tests showed the hybrid method to be 100 to 1000 times faster than MILP or CP when all tasks have the same deadline.

Logic-based Benders methods have also been adapted to solving integer programming problems [12, 3] and the propositional satisfiability problem [8, 12]. Similar ideas have been applied to minimal dispatching of automated guided vehicles [4], steel production scheduling [7], real-time scheduling of computer processors [2], traffic diversion [20], batch scheduling in a chemical plant [16], and polypropylene batch scheduling in particular [18]. 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 in [5].

In this paper we address minimum tardiness problems, in which the subproblem is an optimization problem. We obtain effective cuts by repeatedly solving the subproblem with slightly different task assignments. The idea is related to finding "minimal conflict sets" of tasks, or small sets of tasks that create infeasibility when assigned to a particular facility. Cambazard et al. [2] applied such an approach to real-time scheduling of computing resources. Here we develop cuts for an optimization rather than a feasibility subproblem. A shorter version of this paper appeared in [11].

As observed in [9, 17], the success of hybrid methods in planning and scheduling relies on including a relaxation of the scheduling subproblem in the master problem. We find that deriving a useful relaxation requires deeper analysis when minimizing total tardiness than when minimizing cost or makespan. A relaxation of the cumulative scheduling problem is presented in [14], but it is expressed in terms of the start time variables, rather than the assignment variables as required for the Benders master problem. We derive here a very different relaxation in terms of 0-1 assignment variables, which is suitable for the MILP master problem.

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 due date d_j. 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

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

where x_j, s_j are the variables and $J_{it}(x) = \{j \mid x_j = i, \ s_j \le t \le s_j + p_{ij}\}$ is the set of tasks underway at time t in facility i.

Precedence constraints may be imposed on tasks that are assigned to the same machine. Thus 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 $s_j + p_{x_j j} \le s_k$.

We investigate two objective functions:

- *number of late tasks*, given by $g(x,s) = \sum_j \delta(s_j + p_{x_j j} - d_j)$, where $\delta(\alpha)$ is 1 if $\alpha > 0$ and 0 otherwise.
- *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.

3 Constraint Programming Formulation

The problem can be formulated using the cumulative constraint as follows:

$$\begin{array}{ll} \text{minimize} & g(x,s) \\ \text{subject to} & r_j \le s_j, \text{ all } j \\ & \text{cumulative}((s_j|x_j = i), (p_{ij}|x_j = i), (c_{ij}|x_j = i), C_i), \text{ all } i \end{array} \tag{2}$$

where $(s_j|x_j = i)$ denotes the tuple of start times for tasks assigned to facility i. When minimizing the number of late tasks, $g(x,s) = \sum_j L_j$ where L_j is binary, and the constraint $(s_j + p_{x_j j} > d_j) \Rightarrow (L_j = 1)$ is added for each j. When minimizing total tardiness, $g(x,s) = \sum_j T_j$, and the constraints $T_j \ge s_j + p_{x_j j} - d_j$ and $T_j \ge 0$ are added for each j.

For purposes of computational testing we formulated (2) using the modeling language of OPL Studio. The essential part of the OPL model appears in Fig. 1. We used the assignAlternatives and setTimes search options specify a branching method that results in substantially better performance than the default method.

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 formulation for minimizing

```
[declarations]

scheduleHorizon = TimeHorizon;
DiscreteResource facility[i in I](C[i]);
AlternativeResources facilitySet(facility);
Activity scheduleTask[j in J];

minimize
    sum(j in J) late[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;
    forall(j in J) {
        scheduleTask[j].start >= task[j].r;
        scheduleTask[j].end > task[j].d => late[j]=1 ;
    };
};

search {
    assignAlternatives;
    setTimes;
};
```

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

the number of late tasks is

$$\begin{aligned}
\min \quad & \sum_j L_j \\
\text{subject to} \quad & NL_j \geq \sum_i (t + p_{ij}) x_{ijt} - d_j, \ \text{ all } j, t && (a) \\
& \sum_{it} x_{ijt} = 1, \ \text{ all } j && (b) \\
& \sum_j \sum_{t' \in T_{ijt}} c_{ij} x_{ijt'} \leq C_i, \ \text{ all } i, t && (c) \\
& x_{ijt} = 0, \ \text{ all } j, t \text{ with } t < r_j \text{ or } t > N - p_{ij} && (d)
\end{aligned} \tag{3}$$

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

$$T_j \geq \sum_i (t + p_{ij}) x_{ijt} - d_j, \ \ T_j \geq 0, \ \text{ all } j, t$$

We also investigated a smaller discrete event model suggested by [19], which uses continuous time. However, it proved much harder to solve than (3).

5 Hybrid Method for Minimizing Late Tasks

The Benders approach formulates a *master problem* that assigns tasks to facilities and a *subproblem* that schedules the tasks assigned to each facility. We write the master problem using an MILP model that minimizes the number of late tasks. In iteration h of the Benders algorithm, the master problem is

$$\begin{array}{ll} \text{minimize} & L \\ \text{subject to} & \displaystyle\sum_i x_{ij} = 1, \text{ all } j \qquad (a) \\ & \text{Benders cuts generated in iterations } 1, \ldots, h-1 \quad (b) \\ & \text{relaxation of subproblem} \qquad (c) \end{array} \tag{4}$$

Here the binary variable x_{ij} is 1 when task j is assigned to facility i. The Benders cuts and relaxation will be described shortly.

Once an assignment $\bar{x}_{ij}$ of tasks to facilities is determined by solving the master problem, a cumulative scheduling subproblem is solved by CP. The subproblem decouples into a separate scheduling problem on each facility i:

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{j \in J_{hi}} L_j \\ \text{subject to} & (s_j + p_{ij} > d_j) \Rightarrow (L_j = 1), \text{ all } j \in J_{hi} \\ & r_j \le s_j, \text{ all } j \in J_{hi} \\ & \text{cumulative}((s_j | j \in J_{hi}), (p_{ij} | j \in J_{hi}), (c_{ij} | j \in J_{hi})) \end{array} \tag{5}$$

where J_{hi} is the set of tasks for which $\bar{x}_{ij} = 1$ (i.e., the tasks assigned to facility i in the master problem solution). If L^*_{hi} is the optimal value of (5), then $\sum_i L^*_{hi}$ is the minimum number of late tasks across all facilities.

At this point we know that whenever the tasks in J_{hi} (perhaps among others) are assigned to facility i, the number of late tasks on facility i is at least L^*_{hi}. This allows us to write a valid lower bound $\underline{L}_{hi}$ on the number of late tasks in facility i for any assignment of tasks to machines. Since $x_{ij} = 0$ when task j is not assigned to facility i, we have

$$\begin{array}{l} \displaystyle\underline{L}_{hi} \ge L^*_{hi} - L^*_{hi} \sum_{j \in J_{hi}} (1 - x_{ij}), \text{ all } i \\ \underline{L}_{hi} \ge 0, \text{ all } i \end{array} \tag{6}$$

By summing over all facilities, we have a lower bound on the total number L of late tasks:

$$L \ge \sum_i \underline{L}_{hi} \tag{7}$$

Let $J^0_{hi} = J_{hi}$.
For all $j \in J_{hi}$:
    Compute $L_i(J^0_{hi} \setminus \{j\})$ by re-solving the subproblem on facility i.
    If $L_i(J^0_{hi} \setminus \{j\}) = L^*_{hi}$ then let $J^0_{hi} = J^0_{hi} \setminus \{j\}$.
Let $J^1_{hi} = J^0_{hi}$.
For all $j \in J^0_{hi}$:
    Compute $L_i(J^1_{hi} \setminus \{j\})$ by re-solving the subproblem on facility i.
    If $L_i(J^1_{hi} \setminus \{j\}) = L^*_{hi} - 1$ then let $J^1_{hi} = J^1_{hi} \setminus \{j\}$.

Fig. 2. Algorithm for generating Benders cuts when minimizing the number of late tasks.

The inequality (7), together with (6), provides a *Benders cut* for iteration h. The cut says that the number of late tasks will be at least the number obtained in the subproblem unless a different assignment of tasks to facilities is used.

In iteration h, the Benders cuts (b) in the master problem (4) consist of inequalities (6)–(7) obtained in iterations $1, \ldots, h-1$. Let v_k be the optimal value of the subproblem in iteration k. The algorithm terminates when the optimal value of the master problem equals $\min_{k \in \{1,\ldots,h-1\}}\{v_k\}$. At any point in the algorithm, a feasible solution of the subproblem is a feasible solution of the original problem, and the optimal value of the master problem is a lower bound on the optimal value of the original problem.

Unfortunately the Benders cuts (6)–(7) are weak and do not perform well in practice. The cuts can be strengthened by identifying, for each facility i, a smaller set J_{hi} of tasks that result in the same number of late tasks. One way to do this is to track which tasks actually play a role in the determining the minimum number of late tasks, as suggested in [10]. However, since this information is not available from commercial CP solvers, the information must be obtained indirectly by repeatedly solving subproblems with different assignments of tasks to facilities.

The following approach was found to yield effective cuts with a modest amount of computation. Let $L_i(J)$ be the minimum number of late tasks on facility i when the tasks in J are assigned to facility i. First identify a set $J^0_{hi} \subseteq J_{hi}$ of tasks that, when assigned to facility i, result in a minimum of L^*_{hi} late tasks; that is, a set J^0_{hi} such that $L_i(J^0_{hi}) = L^*_{hi}$. This is done via the simple greedy algorithm in Fig. 2. Then identify a set $J^1_{hi} \subseteq J^0_{hi}$ of tasks such that $L_i(J^1_{hi}) = L^*_{hi} - 1$, again using the algorithm of Fig. 2. The inequalities (6) can now be replaced by the generally stronger inequalities

$$\begin{aligned}
&\underline{L}_{hi} \geq L^*_{hi} - L^*_{hi} \sum_{j \in J^0_{hi}} (1 - x_{ij}), \text{ all } i \\
&\underline{L}_{hi} \geq L^*_{hi} - 1 - L^*_{hi} \sum_{j \in J^1_{hi}} (1 - x_{ij}), \text{ all } i \\
&\underline{L}_{hi} \geq 0, \text{ all } i
\end{aligned} \tag{8}$$

Table 1. Data for a planning and scheduling problem instance with 4 tasks and 2 facilities. The release times r_j are all zero, and each facility i has capacity $C_i = 3$.

		Facility 1			Facility 2		
j	d_j	p_{1j}	c_{1j}	$p_{1j}c_{1j}$	p_{2j}	c_{2j}	$p_{2j}c_{2j}$
1	2	2	3	6	4	3	12
2	3	4	2	8	5	2	10
3	4	5	1	5	6	1	6
4	5	6	3	18	5	3	15

These cuts remain valid for any set of additional constraints that may be added to the subproblems. The cut generation requires modest computation time but significantly reduce the number of iterations. One could define $J^2_{hi}, J^3_{hi}, \ldots$ and generate cuts for them in similar fashion, but they require greater computation time in exchange for a smaller reduction in iterations.

As an example, consider the problem of Table 1, which involves 4 tasks and 2 facilities. Suppose that at some iteration h in the Benders algorithm, tasks 1–3 are assigned to facility 1 and task 4 to facility 2. The optimal schedule for facility 1 is shown in Fig. 3, while the optimal schedule for facility 2 simply schedules task 4 at time 0. The minimum number of late jobs on facility 1 is $L^*_{h1} = 2$. The execution of the algorithm of Fig. 2 appears in Table 2. Since $J^0_{h1} = \{2, 3\}$ and $J^1_{h1} = \emptyset$, the Benders cuts (8) for facility 1 are

$$\begin{aligned} \underline{L}_{h1} &\geq 2 - 2(1 - x_{12}) - 2(1 - x_{13}) \\ \underline{L}_{h1} &\geq 1 \\ \underline{L}_{h1} &\geq 0 \end{aligned}$$

The cut for facility 2 is simply $\underline{L}_{h2} \geq 0$.

6 Relaxation for Minimizing Late Tasks

It is straightforward to relax the subproblem when minimizing the number of late tasks. (It will be harder when minimizing total tardiness.) A task j assigned to facility i consumes *energy* $c_{ij}p_{ij}$, and its running time is a least the *task interval* $c_{ij}p_{ij}/C_i$. Let $J(t_1, t_2)$ be the set of tasks whose time windows are contained in $[t_1, t_2]$. Thus $J(t_1, t_2) = \{j \mid [r_j, d_j] \subseteq [t_1, t_2]\}$. When executed on facility i, the running time of these tasks is at least the task interval

$$M = \frac{1}{C_i} \sum_{j \in J(t_1, t_2)} c_{ij} p_{ij} \tag{9}$$

Table 2. Calculation of J^0_{h1} and J^1_{h1} in the example of Table 1. Here $L^*_{h1} = 2$.

Calculation of J^0_{h1}, which is initially $\{1, 2, 3\}$:

j	$L_1(J^0_{h1} \setminus \{j\})$	New J^0_{h1}
1	2	$\{2, 3\}$
2	1	$\{2, 3\}$
3	1	$\{2, 3\}$

Calculation of J^1_{h1}, which is initially $\{2, 3\}$:

j	$L_1(J^1_{h1} \setminus \{j\})$	New J^1_{h1}
2	1	$\{3\}$
3	1	$\emptyset$

If $M > t_2 - t_1$ then at least one task is late, and in fact the number of late tasks on facility i is at least

$$\frac{M - (t_2 - t_1)}{\max\limits_{j \in J(t_1, t_2)} \{p_{ij}\}} \tag{10}$$

rounded up to the nearest integer.

Returning to the example of Table 1, suppose that tasks 1–3 are assigned to facility 1. We need only consider the tasks in $J(0, d_j)$ for $j = 1, 2, 3$. For $J(0, d_1) = J(0, 2) = \{1\}$, we have $M = (1/3)c_{11}p_{11} = 2$. The lower bound (10) is $(2 - 2)/2 = 0$. For $J(0, d_2) = \{1, 2\}$, the bound is $\frac{5}{12}$, and for $J(0, d_3) = \{1, 2, 3\}$ it is $\frac{7}{15}$. Thus at least $\lceil \frac{7}{15} \rceil = 1$ task is late (in fact, 2 are late in the optimal solution).

More generally, define $\bar{r}_1, \ldots, \bar{r}_{n_r}$ to be the distinct values among the release times $r_1, \ldots, r_n$ in increasing order, and similarly for $\bar{d}_1, \ldots, \bar{d}_{n_d}$. Then from (9) and (10) we have the following relaxation:

$$\begin{aligned}
&L \geq \sum_i L_i \\
&L_i \geq \frac{\dfrac{1}{C_i} \displaystyle\sum_{\ell \in J(\bar{r}_j, \bar{d}_k)} c_{i\ell} p_{i\ell} x_{i\ell} - \left(\bar{d}_k - \bar{r}_j\right)}{\max\limits_{\ell \in J(\bar{r}_j, \bar{d}_j)} \{p_{i\ell}\}}, \quad j = 1, \ldots, n_r, \ k = 1, \ldots, n_d, \text{ all } i \\
&L_i \geq 0, \quad \text{all } i
\end{aligned} \tag{11}$$

which becomes (c) in the master problem (4).

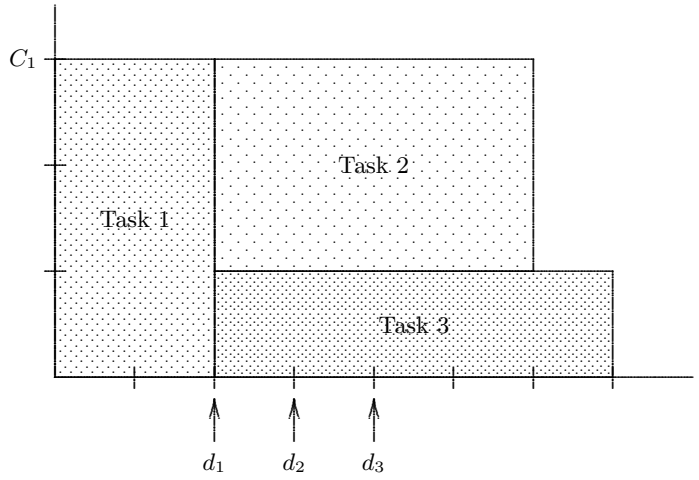

Fig. 3. *Schedule for tasks 1–3 on facility 1 that minimizes the number of late jobs (2) and minimizes total tardiness (6).*

In the example, the relaxation (11) is

$$
\begin{aligned}
& L \geq L_1 + L_2 \\
& L_1 \geq x_{11} - 1, \;\; L_1 \geq \tfrac{1}{2}x_{11} + \tfrac{2}{3}x_{12} - \tfrac{3}{4}, \;\; L_1 \geq \tfrac{2}{5}x_{11} + \tfrac{8}{15}x_{12} + \tfrac{1}{3}x_{13} - \tfrac{4}{5}, \\
& \qquad L_1 \geq \tfrac{1}{3}x_{11} + \tfrac{4}{9}x_{12} + \tfrac{5}{18}x_{13} + \tfrac{1}{3}x_{14} - \tfrac{5}{6} \\
& L_2 \geq x_{21} - \tfrac{1}{2}, \;\; L_2 \geq \tfrac{4}{5}x_{11} + \tfrac{2}{3}x_{22} - \tfrac{3}{5}, \;\; L_2 \geq \tfrac{2}{3}x_{21} + \tfrac{5}{9}x_{22} + \tfrac{1}{3}x_{23} - \tfrac{2}{3}, \\
& \qquad L_2 \geq \tfrac{2}{3}x_{21} + \tfrac{5}{9}x_{22} + \tfrac{1}{3}x_{23} + \tfrac{5}{6}x_{24} - \tfrac{5}{6} \\
& L_1, L_2 \geq 0
\end{aligned}
$$

The solution of the initial master problem (4) assigns tasks 1, 3 and 4 to facility 1 and task 2 to facility 2.

7 Hybrid Method for Minimizing Total Tardiness

In iteration h of the Benders method, the master problem for minimizing total tardiness is

$$
\begin{aligned}
\text{minimize} \quad & T \\
\text{subject to} \quad & \sum_i x_{ij} = 1, \text{ all } j && (a) \\
& \text{Benders cuts for iterations } 1, \ldots, h-1 && (b) \\
& \text{relaxation of subproblem} && (c)
\end{aligned}
\tag{12}
$$

The subproblem again decouples into a cumulative scheduling problem for each facility i:

$$\begin{aligned} \text{minimize} \quad & \sum_{j \in J_i} T_j \\ \text{subject to} \quad & T_j \geq s_j + p_{ij} - d_j, \quad \text{all } j \in J_i \\ & r_j \leq s_j, \quad \text{all } j \in J_i \\ & \text{cumulative}((s_j | j \in J_i), (p_{ij} | j \in J_i), (c_{ij} | j \in J_i)) \end{aligned} \tag{13}$$

We found the following scheme to generate effective Benders cuts. As before let J_{hi} be a set of tasks assigned to facility i in iteration h, and let T^*_{hi} be the resulting minimum tardiness on facility i. 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 minimum 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 the tasks in Z_{hi} all 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 minimum tardiness below T^0_{hi}. This yields the following Benders cuts in iteration h:

$$\begin{aligned} T &\geq T^0_{hi} - T^0_{hi} \sum_{j \in J_{hi} \setminus Z_{hi}} (1 - x_{ij}), \quad \text{all } i \\ T &\geq T^*_{hi} - T^*_{hi} \sum_{j \in J_{hi}} (1 - x_{ij}), \quad \text{all } i \end{aligned} \tag{14}$$

The second cut 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. These cuts are again valid for any set of additional constraints that may be added to the subproblem.

Suppose in the example of Table 1 that tasks 1–3 are again assigned to facility 1 and task 4 to facility 2 in iteration h. Figure 3 shows the minimum tardiness solution on facility 1, which has total tardiness $T^*_{h1} = 6$. Since the removal of any one task from facility 1 reduces the minimum tardiness, $Z_{h1} = \emptyset$ and the cuts (14) are both

$$T \geq 6 - 6(1 - x_{11}) - 6(1 - x_{12}) - 6(1 - x_{13})$$

The cut for facility 2 is $T \geq 0$.

8 Relaxation for Minimizing Total Tardiness

Our relaxation of the minimum tardiness scheduling subproblem has two parts. The first and simpler part is similar to the relaxation obtained for minimizing the number of late tasks. It is based on the following lemma. Recall that $J(t_1, t_2)$ is the set of jobs with time windows between t_1 and t_2.

Lemma 1. *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 $t = \frac{1}{C_i} \sum_{j \in J(0,d_k)} p_{ij} c_{ij}$. Since the last task has due date no later than d_k, its tardiness is no less than $(t - d_k)^+$. Thus total tardiness is no less than $(t - d_k)^+$.

In the example of Table 1, if tasks 1–3 are assigned to facility 1, the bounds of Lemma 1 are

$$\begin{aligned}
&\text{for } J(0, d_1) = \{1\}: && \left(\tfrac{1}{3}(6) - 2\right)^+ = 0 \\
&\text{for } J(0, d_2) = \{1, 2\}: && \left(\tfrac{1}{3}(6 + 8) - 3\right)^+ = 1\tfrac{2}{3} \\
&\text{for } J(0, d_3) = \{1, 2, 3\}: && \left(\tfrac{1}{3}(6 + 8 + 5) - 4\right)^+ = 2\tfrac{1}{3}
\end{aligned}$$

Since the data are integral, the minimum tardiness is at least $\lceil 2\frac{1}{3} \rceil = 3$ (it is 6 in the optimal schedule).

Lemma 1 gives rise to a relaxation consisting of

$$\begin{aligned}
& T \geq \sum_i T_i^L \\
& T_i^L \geq \frac{1}{C_i} \sum_{j \in J(0,d_k)} p_{ij} c_{ij} x_{ij} - d_k, \quad \text{all } i, k \\
& T_i^L \geq 0, \quad \text{all } i
\end{aligned} \tag{15}$$

and $T \geq 0$. In the example, the relaxation (15) becomes

$$\begin{aligned}
& T \geq T_1^L + T_2^L \\
& T_1^L \geq 2x_{11} - 2, \;\; T_1^L \geq 2x_{11} + \tfrac{8}{3}x_{12} - 3, \;\; T_1^L \geq 2x_{11} + \tfrac{8}{3}x_{12} + \tfrac{5}{3}x_{13} - 4, \\
& \qquad T_1^L \geq 2x_{11} + \tfrac{8}{3}x_{12} + \tfrac{5}{3}x_{13} + 2x_{14} - 5 \\
& T_2^L \geq 4x_{21} - 2, \;\; L_2 \geq 4x_{11} + \tfrac{10}{3}x_{22} - 3, \;\; T_2^L \geq 4x_{21} + \tfrac{10}{3}x_{22} + 2x_{23} - 4, \\
& \qquad L_2 \geq 4x_{21} + \tfrac{10}{3}x_{22} + 2x_{23} + 5x_{24} - 5 \\
& T_1^L, T_2^L \geq 0
\end{aligned} \tag{16}$$

The second part of the relaxation can be developed on basis of the following lemma. For each facility i let π_i be a permutation of $\{1, \ldots, n\}$ that orders the tasks by increasing energy on facility i; that is, $p_{i\pi_i(1)} c_{i\pi_i(1)} \leq \cdots \leq p_{\pi_i(n)} c_{\pi_i(n)}$. One can obtain a lower bound on the tardiness incurred by the job with the kth latest deadline by supposing that it finishes no sooner than the task interval of the k jobs with the smallest energies. More precisely:

Lemma 2. *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

$$T^* = \sum_{k=1}^{n} (t_k + p_{ik} - d_k)^+ \tag{17}$$

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

$$\underline{T}_k(\sigma) = \left(\frac{1}{C_i} \sum_{j=1}^{k} p_{i\pi_i(j)} c_{i\pi_i(j)} - d_{\sigma(k)} \right)^+ \tag{18}$$

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 (17) as

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

We observe that

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

where the first inequality is based on the energy 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

$$\begin{aligned}
\underline{T}(\sigma_p) &= \sum_{j=1}^{k-1} \underline{T}_j(\sigma_p) + \underline{T}_k(\sigma_p) + \underline{T}_{k+1}(\sigma_p) + \sum_{j=k+2}^{n} \underline{T}_j(\sigma_p) \\
\underline{T}(\sigma_{p+1}) &= \sum_{j=1}^{k-1} \underline{T}_j(\sigma_p) + \underline{T}_k(\sigma_{p+1}) + \underline{T}_{k+1}(\sigma_{p+1}) + \sum_{j=k+2}^{n} \underline{T}_j(\sigma_p)
\end{aligned} \tag{19}$$

Using (18), 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 (19) 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.

In the example, suppose again that tasks 1–3 are assigned to facility 1. The permutation π_1 is $(\pi_1(1), \pi_1(2), \pi_1(3)) = (3, 1, 2)$. The lower bound $\underline{T}$ of Lemma 2 is $\underline{T}_1 + \underline{T}_2 + \underline{T}_3$, where

$$\begin{aligned}
\underline{T}_1 &= \left(\tfrac{1}{3}(5) - 2\right)^+ = 0 \\
\underline{T}_2 &= \left(\tfrac{1}{3}(5+6) - 3\right)^+ = \tfrac{2}{3} \\
\underline{T}_3 &= \left(\tfrac{1}{3}(5+6+8) - 4\right)^+ = 2\tfrac{1}{3}
\end{aligned}$$

The bound is $\underline{T} = 3$, which in this case is slightly stronger than the bound of $2\frac{1}{3}$ obtained from Lemma 1.

The bound of Lemma 2 can be written in terms of the variables x_{ik}:

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

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 $\underline{T}'_{ik} \geq 0$. We linearize the bound by writing it as

$$\sum_{k=1}^{n} \underline{T}_{ik} \tag{20}$$

where

$$\underline{T}_{ik} \geq \frac{1}{C_i} \sum_{j=1}^{k} p_{i\pi_i(j)} c_{i\pi_i(j)} x_{i\pi_i(j)} - d_k - (1 - x_{ik}) U_{ik}, \quad k = 1, \ldots, n \tag{21}$$

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 (21) is never positive when $x_{ik} = 0$. Finally, to obtain a relaxation of the subproblem, we sum (20) over all facilities and write

$$T \geq \sum_{i=1}^{m} \sum_{k=1}^{n} \underline{T}_{ik} \tag{22}$$

The relaxation (c) of the master problem now consists of (15), (22), and (21) for $i = 1, \ldots, m$. The relaxation is valid only when tasks are indexed so that $d_1 \leq \cdots \leq d_n$. In the example, the relaxation consists of (16) and the following:

$$\begin{aligned}
&T \geq \underline{T}_{11} + \underline{T}_{12} + \underline{T}_{13} + \underline{T}_{21} + \underline{T}_{22} + \underline{T}_{23} \\
&\underline{T}_{11} \geq \tfrac{5}{3}x_{11} - 2 + \tfrac{1}{3}(1 - x_{11}) \\
&\underline{T}_{12} \geq \tfrac{5}{3}x_{11} + 2x_{12} - 3 - \tfrac{2}{3}(1 - x_{12}) \\
&\underline{T}_{13} \geq \tfrac{5}{3}x_{11} + 2x_{12} + \tfrac{8}{3}x_{13} - 4 - \tfrac{7}{3}(1 - x_{13}) \\
&\underline{T}_{14} \geq \tfrac{5}{3}x_{11} + 2x_{12} + \tfrac{8}{3}x_{13} + 6x_{14} - 5 - \tfrac{22}{3}(1 - x_{14}) \\
&\underline{T}_{21} \geq 2x_{21} - 2 \\
&\underline{T}_{22} \geq 2x_{21} + \tfrac{10}{3}x_{12} - 3 - \tfrac{7}{3}(1 - x_{22}) \\
&\underline{T}_{23} \geq 2x_{21} + \tfrac{10}{3}x_{12} + 4x_{13} - 4 - \tfrac{16}{3}(1 - x_{23}) \\
&\underline{T}_{24} \geq 2x_{21} + \tfrac{10}{3}x_{12} + 4x_{13} + 5x_{24} - 5 - \tfrac{28}{3}(1 - x_{24}) \\
&\underline{T}_{ik} \geq 0
\end{aligned}$$

The solution of the initial master problem assigns tasks 1–3 to facility 1 and task 4 to facility 2.

9 Problem Generation

Random instances were generated as follows. We set the number of facilities at 3, and the number of tasks at $n = 10, 12, \ldots, 24$. 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 $[1, 10]$. The processing time p_{ij} was drawn from a uniform distribution on $[2, 20]$, $[2, 25]$ and $[2, 30]$ for facilities $i = 1, 2, 3$, 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. The release dates were set to zero and the due date drawn from a uniform distribution on $[\beta n/4, \beta n]$. We used $\beta = 20/9$, partly since this was consistent with parameter settings used in earlier research, and partly because it leads to reasonable results (a few late tasks in most instances, and no late tasks in a few instances). No precedence constraints were used, which tends to make the scheduling portion of the problem more difficult.

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.

Table 3 shows computational results for minimizing the number of late tasks on three facilities using CP, MILP and the hybrid method. Since problem difficulty tends to increase with the minimum number of late tasks, the instances are ordered accordingly for each problem size. The problem instance identifier k appears in the last column. The instances are named dd$n$j3mk, where n is the number of tasks and k the instance identifier. The instances are available at the web site web.tepper.cmu.edu/jnh/planning.htm.

On all but two problem instances the hybrid method is faster than MILP, which in turn is generally faster than CP. The advantage of the hybrid method becomes greater as the instances grow in size. The speedup is generally two or three orders of magnitude for instances with 16 or more tasks. The average speedup factor relative to MILP is 295 for these instances. This is almost certainly a substantial underestimate for the instances averaged, since the MILP solver was cut off after two hours. (The average omits instances in which the hybrid method was also cut off.) In addition MILP failed to solve 10 instances, while the hybrid method failed to solve only one instance.

Table 4 shows computational results for minimizing total tardiness. Again the hybrid method is almost always faster than MILP, which is faster than CP. The advantage of the hybrid approach is not as great as in the previous table, but the speedup factor is still significant on instances with 16 or more tasks. The average speedup factor on these instances is 25, which is again an underestimate for these instances. (The average omits instances for which the hybrid method was also cut off.)

The hybrid method failed to solve 6 of the 40 instances to optimality, only a modest improvement over the 10 that were intractable for MILP. However, when the hybrid method failed to find provably optimal solutions, it obtained much better feasible solutions than obtained by MILP in the same two-hour period. In most cases these solutions were found very early in the solution process. Table 4 also shows the lower bounds obtained from the master problem, which in these instances are not very tight.

Table 5 illustrates the importance of relaxations in the hybrid approach, particularly when minimizing total tardiness. Lemmas 1 and 2 are clearly critical

Table 3. Computational results for minimizing the number of late tasks on three facilities. Computation is terminated after two hours (7200 seconds). The test instances are dd$N$jMmK, where N is the number of tasks, M the number of facilities, and K the instance number shown in the last column.

Tasks	Time (sec)			Hybrid/	Best solution		Instance
	CP	MILP	Hybrid	MILP speedup	value found[1] MILP	Hybrid	
10	0.09	0.48	0.05	9.6	1	1	1
	2.5	0.51	0.17	3.0	1	1	2
	0.28	0.46	0.27	1.7	2	2	5
	0.15	0.41	0.93	0.4	3	3	4
	1.7	3.9	3.0	1.3	3	3	3
12	0.01	0.73	0.07	10	0	0	1
	0.01	0.70	0.22	3.2	0	0	5
	0.02	0.64	0.06	11	1	1	3
	3.2	1.4	0.18	7.8	1	1	4
	1.6	1.7	0.34	5.0	1	1	2
14	1092	5.8	0.52	11	1	1	3
	382	8.0	0.69	12	1	1	2
	265	3.2	0.69	4.6	2	2	1
	85	2.6	1.3	2.0	2	2	5
	5228	1315	665	2.0	3	3	4
16	304	2.7	0.51	5.3	0	0	2
	?[2]	31	0.24	129	1	1	4
	310	22	0.41	54	1	1	5
	4925	29	2.7	11	2	2	3
	19	5.7	24	0.2	4	4	1
18	>7200	2.0	0.11	18	0	0	5
	?[2]	8.0	0.21	38	1	1	4
	>7200	867	8.5	102	1	1	2
	>7200	6.3	1.4	4.5	2	2	3
	>7200	577	3.4	170	2	2	1
20		97	0.37	262	0	0	1
		>7200	2.3	>3130	(1)	1	5
		219	5.0	44	1	1	2
		>7200	11	>655	(2)	2	3
		843	166	5.1	3	3	4
22		16	1.3	12	0	0	4
		>7200	3.7	>1946	(1)	1	1
		>7200	49	>147	(3)	2	5
		>7200	3453	>2.1	(5)	2	3
		>7200	>7200		(6)	(6)	2
24		25	0.8	31	0	0	3
		>7200	18	>400	(1)	0	5
		>7200	62	>116	(2)	0	4
		>7200	124	>58	(3)	1	1
		>7200	234	>31	(2)	1	2

[1]Values in parentheses are not proved optimal.
[2]Computation terminates with a segmentation fault.

Table 4. Computational results for minimum tardiness problems on three facilities. Computation is terminated after two hours (7200 seconds). The test instances are dd$N$jMmK, where N is the number of tasks, M the number of facilities, and K the instance number shown in the last column.

Tasks	Time (sec)			Hybrid/	Best solution		Benders	
	CP	MILP	Hybrid	MILP	value found[1]		lower	Instance
				speedup	MILP	Benders	bound[2]	
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 are 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 5. Effect of relaxations on performance of the hybrid method. Computation time in seconds is shown.

Tasks	Minimizing late tasks: with relaxation	without relaxation	Minimizing tardiness: with relaxation	without relaxation	Instance
16	0.5	2.6	1.4	4.4	2
	0.4	1.5	2.1	6.5	5
	0.2	1.3	4.2	30	4
	2.7	4.2	156	199	3
	24	18	765	763	1
18	0.1	1.1	2.8	10	5
	0.2	0.7	5.3	17	4
	3.4	3.3	47	120	1
	1.4	15	49	354	3
	8.5	11	1203	5102	2
20	0.4	88	18	151	1
	2.3	9.7	23	1898	5
	5.0	63	29	55	2
	11	19	332	764	3
	166	226	>7200	>7200	4

to the success of the hybrid method, especially when there are more than 16 tasks or so.

11 Conclusions

We find that integrating CP and MILP through a Benders scheme can substantially improve on the state of the art in planning and scheduling to minimize tardiness. The hybrid method is often two or three orders of magnitude faster than CP or MILP when minimizing the number of late tasks, and it solves significantly more problems. It is significantly faster when minimizing total tardiness, and when it fails to solve the problem to optimality, it nonetheless finds a much better feasible solution in the same time period.

The problems become hard for all the methods examined when there are more than a few late tasks in the optimal solution. However, in such cases it is probably best to relax some of the time windows so as to reflect scheduling priorities, perhaps by postponing due dates for less critical tasks. This makes the problem easier to solve and yields a more meaningful compromise solution in practice.

References

1. J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.
2. H. Cambazard, P.-E. Hladik, A.-M. Déplanche, N. Jussien, and Y. Trinquet. Decomposition and learning for a hard real time task allocation problem. In M. Wallace, editor, *Principles and Practice of Constraint Programming (CP2004)*, volume 3258 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2004.

3. Y. Chu and Q. Xia. Generating benders cuts for a class of integer programming problems. In J. C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004)*, volume 3011 of *Lecture Notes in Computer Science*, pages 127–141. Springer, 2004.
4. A. I. Corréa, A. Langevin, and L. M. Rousseau. 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, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004)*, volume 3011 of *Lecture Notes in Computer Science*, pages 370–378. Springer, 2004.
5. A. Eremin and M. Wallace. Hybrid Benders decomposition algorithm in constraint logic programming. In T. Walsh, editor, *Principles and Practice of Constraint Programming (CP2001)*, volume 2239 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2001.
6. A. M. Geoffrion. Generalized benders decomposition. *Journal of Optimization Theory and Applications*, 10:237–260, 1972.
7. I. Harjunkoski and I. E. Grossmann. A decomposition approach for the scheduling of a steel plant production. *Computers and Chemical Engineering*, 25:1647–1660, 2001.
8. J. N. Hooker. *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. Wiley, New York, 2000.
9. J. N. Hooker. A hybrid method for planning and scheduling. In M. Wallace, editor, *Principles and Practice of Constraint Programming (CP2004)*, volume 3258 of *Lecture Notes in Computer Science*, pages 305–316. Springer, 2004.
10. J. N. Hooker. A hybrid method for planning and scheduling. *Constraints*, 10:385–401, 2005.
11. J. N. Hooker. Planning and scheduling to minimize tardiness. In *Principles and Practice of Constraint Programming (CP2005)*, Lecture Notes in Computer Science. Springer, 2005.
12. J. N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96:33–60, 2003.
13. J. N. Hooker and H. Yan. Logic circuit verification by benders decomposition. In V. Saraswat and P. Van Hentenryck, editors, *Principles and Practice of Constraint Programming: The Newport Papers*, pages 267–288, Cambridge, MA, 1995. MIT Press.
14. J. N. Hooker and H. Yan. A relaxation for the cumulative constraint. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming*, volume 2470 of *Lecture Notes in Computer Science*, pages 686–690. Springer, 2002.
15. V. Jain and I. E. Grossmann. Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS Journal on Computing*, 13:258–276, 2001.
16. C. T. Maravelias and I. E. Grossmann. Using MILP and CP for the scheduling of batch chemical processes. In J. C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004)*, volume 3011 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2004.
17. E. Thorsteinsson. Branch and check: A hybrid framework integrating mixed integer programming and constraint logic programming. In T. Walsh, editor, *Principles and Practice of Constraint Programming (CP2001)*, volume 2239 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2001.

18. C. Timpe. Solving planning and scheduling problems with combined integer and constraint programming. *OR Spectrum*, 24:431–448, 2002.
19. M. Türkay and I. E. Grossmann. Logic-based MINLP algorithms for the optimal synthesis of process networks. *Computers and Chemical Engineering*, 20:959–978, 1996.
20. Q. Xia, A. Eremin, and M. Wallace. Problem decomposition for traffic diversions. In J. C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004)*, volume 3011 of *Lecture Notes in Computer Science*, pages 348–363. Springer, 2004.