A Hybrid Method for Planning and Scheduling

J. N. Hooker (john@hooker.tepper.cmu.edu)
Tepper School of Business, Carnegie Mellon University, Pittsburgh, USA

Abstract.
We combine mixed integer linear programming (MILP) and constraint programming (CP) to solve 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 minimum cost problems, as well as minimum makespan problems in which all tasks have the same release date and deadline. We obtain computational speedups of several orders of magnitude relative to the state of the art in both MILP and CP.

We address a fundamental class of planning and scheduling problems for manufacturing and supply chain management. Tasks must be assigned to facilities and scheduled subject to release dates and deadlines. Tasks may run in parallel on a given facility provided the total resource consumption at any time remains with limits (cumulative scheduling). In our study the objective is to minimize cost or minimize makespan.

The problem naturally decomposes into an assignment portion and a scheduling portion. We exploit the relative strengths of mixed integer linear programming (MILP) and constraint programming (CP) by applying MILP to the assignment problem and CP to the scheduling problem. We then link the two with a logic-based Benders algorithm.

We obtain speedups of several orders of magnitude relative to the existing state of the art in both mixed integer programming (CPLEX) and constraint programming (ILOG Scheduler). As a result we solve larger instances to optimality than could be solved previously. In minimum makespan problems, the Benders method provides a feasible solution and a lower bound on the optimal makespan even when it is terminated before finding a provably optimal solution.

1. The Basic Idea

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. Solution of the subproblem generates a *Benders cut* (a type of nogood) that the primary variables must satisfy in all subsequent solutions enumerated. 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.

In this paper, the primary variables define the assignment of tasks to facilities, and the master problem is the assignment problem augmented with Benders cuts. The subproblem is the set of cumulative scheduling problems (one for each facility) that result from a given assignment.

In classical Benders decomposition [2, 6], the subproblem is always a continuous linear or nonlinear programming problem, and there is a standard way to obtain Benders cuts. In a logic-based Benders method, the subproblem is an arbitrary optimization problem, and a specific scheme for generating cuts must be devised for each problem class by solving the *inference dual* of the subproblem. In the present context, the Benders cuts must also be linear inequalities, since the master problem is an MILP. It is also important in practice to augment the master problem with a linear relaxation of the subproblem.

The main contribution of this paper is to develop effective linear Benders cuts and subproblem relaxations for (a) minimum cost problems with cumulative scheduling, and (b) minimum makespan problems with cumulative scheduling in which all tasks have the same release date and deadline.

2. Previous Work

Logic-based Benders decomposition was introduced by Hooker and Yan [10] in the context of logic circuit verification. The idea was formally developed in [7] and applied to 0-1 programming by Hooker and Ottosson [9].

Jain and Grossmann [11] successfully applied logic-based Benders to minimum-cost planning and scheduling problems in which the subproblems are one-machine disjunctive (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. Two goals of the present paper are (a) to accommodate cumulative scheduling, and (b) to develop Benders cuts when the subproblem is an optimization problem, as in the case of minimum makespan problems.

In related work, we observed in [7] that the master problem need only be solved once by a branching algorithm that accumulates Benders cuts as they are generated. Thorsteinsson [17] 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 Benders. 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.

A shorter version of this paper appeared as [8]. In the same proceedings volume, a paper by Cambazard et al. [3] applies a logic-based Benders method to real-time allocation and scheduling of computing resources. The master problem allocates tasks to networked processors, and the subproblem schedules the tasks so as to meet deadlines. Since both the master problem and subproblem are solved with constraint technology, the algorithm represents a purely CP rather than a hybrid approach.

Classical Benders decomposition can also be useful in a CP context, as shown by Eremin and Wallace [5].

3. 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 zero and deadline d_j (viewed as a due date in the case of minimum tardiness problems). The tasks assigned to facility i must be given start times t_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.

We investigate two objective functions. If we let x_j be the facility assigned to task j, the *cost* objective is $g(x,t) = \sum_j f_{x_j j}$, where f_{ij} is the fixed cost of processing task j on facility i. The *makespan* objective is $g(x,t) = \min_j \{t_j + p_{x_j j}\}$.

4. Constraint Programming Formulation

The problem is succinctly written using the constraint cumulative(t, p, c, C), which requires that tasks be scheduled at times $t = (t_1, \dots, t_n)$ 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 $J_t = \{j \mid t_j \leq t \leq t_j + p_j\}$ is the set of tasks underway at time t.

The planning and scheduling problem becomes

$$\begin{array}{ll} \text{minimize} & g(x,t) \\ \text{subject to} & \text{cumulative}((t_j | x_j = i), (p_{ij} | x_j = i), (c_{ij} | x_j = i), C_i), \text{ all } i \\ & r_j \leq t_j \leq d_j - p_{x_j j}, \text{ all } j \end{array} \tag{1}$$

where $g(x,t)$ is the desired objective function and $(t_j|x_j = i)$ denotes the tuple of start times for tasks assigned to facility i. The second constraint enforces the time windows.

5. 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 is

$$\begin{array}{lll} \min & g(x,t) & \\ \text{subject to} & \sum_{it} x_{ijt} = 1, \text{ all } j & (a) \\ & \sum_j \sum_{t' \in T_{ijt}} c_{ij} x_{ijt} \leq C_i, \text{ all } i,t & (b) \\ & x_{ijt} = 0, \text{ all } j,t \text{ with } d_j - p_{ij} < t \leq r_j \text{ or } t > n - p_{ij} + 1 & (c) \end{array} \tag{2}$$

where $T_{ijt} = \{t' \mid t - p_{ij} < t' \leq t\}$ 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 cost objective is $g(x,t) = \sum_{ijt} f_{ij} x_{ijt}$. The makespan objective is $g(x,t) = z$, together with the constraints $z \geq \sum_{it}(t + p_{ij})x_{ijt}$ for all j.

Due to the size of (2), we also investigated a smaller discrete event model suggested by [18], which uses continuous time. However, it proved to be much harder to solve than (2). We therefore omitted the discrete event model from the computational studies described below.

6. Logic-based Benders Decomposition

Logic-based Benders decomposition applies to problems of the form

$$\begin{array}{ll} \text{minimize} & f(x,t) \\ \text{subject to} & C(x,t) \\ & x \in D_x,\ t \in D_t \end{array} \tag{3}$$

where $C(x,t)$ is a set of constraints containing variables x, t. 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:

$$\begin{array}{ll} \text{minimize} & f(\bar{x},t) \\ \text{subject to} & C(\bar{x},t) \\ & t \in D_t \end{array} \tag{4}$$

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

The *inference dual* of (4) is the problem of inferring the tightest possible lower bound on $f(\bar{x}, t)$ from $C(\bar{x}, t)$. It can be written

$$\begin{array}{ll} \text{maximize} & v \\ \text{subject to} & C(\bar{x}, t) \Longrightarrow f(\bar{x}, t) \geq v \\ & v \in I\!R \end{array} \tag{5}$$

where $\Longrightarrow$ means "implies" (see [7] for details).

The solution of the dual can be viewed as a derivation of the tightest possible bound $\hat{v}$ on $f(x, t)$ when $x = \bar{x}$. For purposes of Benders decomposition, we wish to derive not only a bound when $x = \bar{x}$ but a function $B_{\bar{x}}(x)$ that provides a valid lower bound on $f(x, t)$ for any given $x \in D_x$. In particular, $B_{\bar{x}}(\bar{x}) = \hat{v}$. If z is the objective function value of (3), this bounding function provides the valid inequality $z \geq B_{\bar{x}}(x)$, which we call a *Benders cut*.

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

$$\begin{array}{ll} \min & z \\ \text{subject to} & z \geq B_{x^h}(x),\ h = 1, \ldots, H-1 \\ & z \in I\!R,\ x \in D_x \end{array} \tag{6}$$

Here $x^1, \ldots, x^{H-1}$ are the solutions of the previous $H-1$ master problems. Then the solution $\bar{x}$ of (6) defines the next subproblem (4).

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 $v^* = \min\{v_1^*, \ldots, v_{H-1}^*\}$. It is shown in [7, 9] that the algorithm converges finitely to an optimal solution under fairly weak conditions, which hold in the present case. At any point in the algorithm, z_H^* and v^* provide lower and upper bounds on the optimal value of the problem.

In the planning and scheduling problem (1), any assignment $\bar{x}$ of tasks to facilities creates the subproblem:

$$\begin{array}{ll} \min & g(\bar{x}, t) \\ \text{subject to} & \text{cumulative}((t_j|\bar{x}_j = i), (p_{ij}|\bar{x}_j = i), (c_{ij}|\bar{x}_j = i), C_i), \text{ all } i \\ & r_j \leq t_j \leq d_j - p_{\bar{x}_j j},\ \text{all } j \end{array} \tag{7}$$

which decomposes into a separate scheduling problem for each facility. After solving the subproblem we generate a Benders cut that becomes part of the master problem (6).

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. Unfortunately, only the primal solution (the schedule itself) is available from the commercial CP solver. Thus for purposes of computational testing we present Benders cuts that require only this information. We indicate in Section 8, however, how stronger cuts can be deduced using "dual" information from the subproblem solution.

7. Minimizing Cost

The cost objective presents the simplest case, since cost can be computed in terms of master problem variables, and the subproblem is a feasibility problem. 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, then J_{hi} is a *conflict set*, or a set that results in infeasibility. The most obvious Benders cut simply rules out any assignment of tasks to facility i that contains the conflict set J_{hi}. In this case $B_{x^h}(x)$ takes the value ∞ when there is an infeasibility and the value $\sum_j f_{x_j j}$ otherwise. The master problem (5), written as a 0-1 programming problem, becomes

$$\begin{aligned} \text{minimize} \quad & \sum_{ij} f_{ij} y_{ij} \\ \text{subject to} \quad & \sum_i y_{ij} = 1, \text{ all } j & (a) \\ & \sum_{j \in J_{hi}} (1 - y_{ij}) \geq 1, \text{ all } i, \ h = 1, \ldots, H-1 & (b) \\ & \text{relaxation of subproblem} & (c) \end{aligned} \tag{8}$$

where $y_{ij} \in \{0, 1\}$, and $y_{ij} = 1$ indicates $x_j = i$. 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, $t_1 \leq r_j$ and $d_j \leq t_2$. 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

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

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 1. $O(n^3)$ 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 2. $O(n)$ 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 $T^i(r_0, r_0) = 0$.

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

$$R^i(\bar{r}_j, \bar{d}_k), \quad j = 1, \ldots, n_r, \ k = 1, \ldots, n_d \tag{10}$$

for each facility i. These inequalities serve as the relaxation (c) in (8).

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

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

be 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 $T^i(t_1, t_2) \geq T^i(u_1, u_2)$. A set of undominated inequalities can be generated for each machine using the algorithm of Fig. 1. 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 (8) consists of

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

for each facility i. The redundant inequalities can be eliminated running the simple $O(n)$ algorithm of Fig. 2 for each facility. Similarly, one can suppose that the deadlines are all $d_0 = \max_j\{d_j\}$ and use the inequalities $R^i(\bar{r}_j, d_0)$.

8. Stronger Benders Cuts

The simple Benders cuts described above can be strengthened. One way to strengthen them is to re-solve the infeasible scheduling problems with various subsets of jobs removed, in order to identify a conflict set $\bar{J}_{hi}$ that is smaller than J_{hi}. One can then replace (8b) with stronger Benders cuts

$$\sum_{j \in \bar{J}_{hi}} (1 - y_{ij}) \geq 1, \quad \text{all } i, h = 1, \ldots, H-1 \tag{11}$$

De Sequeira and Puget [4] state an algorithm for finding a conflict set that is minimal in the sense that any proper subset is feasible. Junker [12] provides a faster algorithm for the same task using a bisection search. This approach was used by Cambazard et al. [3] and can be practical when the scheduling subproblems are quickly solved.

It may be more efficient, however, to construct a stronger cut on the basis of "dual information" obtained from the solution of the scheduling problem; that is, on the basis of an explanation or proof of infeasibility. One straightforward approach is to keep track of which tasks actually play a role in the proof of infeasibility, and let these tasks comprise the conflict set $\bar{J}_i^h$ in (11). We indicate how to do this when domain filtering is based on one type of edge finding. The idea can be extended to other forms of domain reduction.

Edge finding is a procedure for identifying jobs that must precede, or that must follow, a set S of other jobs. For the scheduling problem on a given facility i, let the current domain of each start time t_j be the interval $[E_j, L_j - p_{ij}]$, where E_j is the earliest start time and L_j the latest *finish* time for task j. For any subset S of the tasks assigned to facility i, let $E_S = \min_{j \in S} E_j$ and $L_S = \max_{j \in S} L_j$. If we can identify an S for which

$$\Big(L_{S \cup \{k\}} - E_S\Big) C < \sum_{j \in S \cup \{k\}} p_{ij} c_{ij} \tag{12}$$

then task k must start before any task in S starts. Similarly, if

$$\Big(L_S - E_{S \cup \{k\}}\Big) C < \sum_{j \in S \cup \{k\}} p_{ij} c_{ij}$$

then task k must finish after all the tasks in S finish.

If edge finding determines that task k must start before the tasks in S, then we can bound how late task k can finish. The available capacity to run any subset $S' \subset S$ of tasks is $(L_{S'} - E_{S'})C$. We can be sure that these tasks do not constrain the finish time of task k if $\Delta_{S'} \leq 0$, where

$$\Delta_{S'} = \sum_{j \in S'} p_j c_j - (L_{S'} - E_{S'})(C - c_k)$$

However, if $\Delta_{S'} > 0$, then task k must finish no later than $\bar{L} = L_{S'} - \Delta_{S'}/c_{ik}$. If $\bar{L} < L_k$, we can tighten L_k by reducing it to $\bar{L}$. Similarly, if task k must finish after the tasks in S finish, then whenever $\Delta_{S'} > 0$ for $S' \subset S$ we infer that task k can start no earlier than $\bar{E} = E_{S'} + \Delta_{S'}/c_{ik}$. If $\bar{E} > E_k$, we can tighten E_k by raising it to $\bar{E}$. An $O(n^2)$ algorithm is presented in [1] that computes, for any given t_k, the smallest interval $[E_k, L_k]$ that can be deduced in this fashion by edge finding.

To keep track of which tasks are involved in a proof of infeasibility, we associate with task j a set R_j of tasks that help to reduce t_j's domain. Initially, $R_j = \{j\}$. If edge finding determines that task k precedes or follows the tasks in S and tightens E_k or L_k as a result, then the tasks in $\bigcup_{j \in S} R_j$ are added to R_k. When the edge finding process is finished and the domain of some t_j is found to be empty, we conclude that R_j is a conflict set.

A somewhat sharper analysis can be obtained by associating with each E_j a set R_j^E of tasks that help tighten that particular bound, and a similar set R_j^L with each L_j. Then when the domain of some t_j is reduced to the empty set, we conclude that $R_j^E \cup R_j^L$ is a conflict set. This approach requires a detailed analysis of the algorithm in [1] to determine under what circumstances each bound is tightened.

In practice, edge finding and other filtering mechanisms are combined with branching search. For instance, one can branch on which task starts first (among those whose position is not already fixed by prior branching). The search proves infeasibility by deducing, at each leaf node ℓ of the branching tree, an empty domain for at least one variable t_j. The conflict set $R_{\ell j}$ of tasks involved in deriving this empty domain can be obtained, in the manner just described, by examining the edge finding operations along the path from the root of the tree down to ℓ. A task that branching places before certain other tasks is treated like a task determined by edge finding to precede the other tasks.

At this point a conflict set can be collected by taking a union of conflict sets over the leaf nodes. For each leaf node ℓ we select a variable

t_{j_ℓ} with empty domain. Then

$$\bar{J}_{hi} = \bigcup_{\ell} R_{\ell j_\ell}$$

is a conflict set for the scheduling problem on facility i.

An interesting characteristic of feasibility problems is that there are often several proofs of infeasibility when there is one proof; in the present case, there may be several conflict sets at each node of the search tree. These proofs can be regarded as alternate dual solutions and are analogous to dual "degeneracy" in linear programming. Some of these proofs or explanations may be more useful or perspicuous than others. In the context of Benders decomposition, it is obviously advantageous to choose dual solutions that result in stronger Benders cuts. Thus one should select j_ℓ at each leaf node ℓ in such a way that $\bar{J}_{hi}$ is as small as possible.

9. Minimizing Makespan

This case is less straightforward than minimizing cost 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 1. Consider a minimum tardiness problem P in which tasks $1, \dots, n$ with release time 0 and deadlines $d_1, \dots, d_n$ 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 $1, \dots, s$ are removed. Then

$$M^* - \hat{M} \leq \Delta + \max_{j \leq s}\{d_j\} - \min_{j \leq s}\{d_j\} \tag{13}$$

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 $1, \dots, s$ sequentially after $\hat{M}$. That is, for $k = 1, \dots, s$ let task k start at time $\hat{M} + \sum_{j=1}^{k-1} p_{ij}$. The makespan of S is $\hat{M} + \Delta$. If $\hat{M} + \Delta \leq \min_{j \leq s}\{d_j\}$, then S is clearly feasible

for P, so that $M^* \leq \hat{M} + \Delta$ and the lemma follows. Now suppose $\hat{M} + \Delta > \min_{j \leq s}\{d_j\}$. This implies

$$\hat{M} + \Delta + \max_{j \leq s}\{d_j\} - \min_{j \leq s}\{d_j\} > \max_{j \leq s}\{d_j\} \tag{14}$$

Since $M^* \leq \max_{j \leq s}\{d_j\}$, (14) implies (13), 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 $(r_1, r_2, r_3) = (0, 0, 0)$, $(d_1, d_2, d_3) = (2, 1, \infty)$, $(p_{i1}, p_{i2}, p_{i3}) = (1, 1, 2)$, and $(c_{i1}, c_{i2}, c_{i3}) = (2, 1, 1)$. 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 tardiness incurred by facility i. The solution of the current master problem removes task $j \in J_{hi}$ from facility i when $y_{ij} = 0$. Thus by Lemma 1, the resulting minimum makespan for facility i is reduced by at most

$$\sum_{j \in J_{hi}} p_{ij}(1 - y_{ij}) \tag{15}$$

This yields a bounding function

$$B_{y^h}(y) = M^*_{hi} - \sum_{j \in J_{hi}} p_{ij}(1 - y_{ij})$$

that provides a lower bound on the optimal makespan of each facility i. We can now write Benders cuts (b) in the master problem:

$$\begin{aligned}
&\text{minimize} \quad M \\
&\text{subject to} \quad \sum_i y_{ij} = 1, \text{ all } j && (a) \\
&\qquad M \geq M^*_{hi} - \sum_{j \in J_{hi}} (1 - y_{ij}) p_{ij}, \\
&\qquad\qquad \text{all } i,\ h = 1, \ldots, H-1 && (b) \\
&\qquad M \geq \frac{1}{C_i} \sum_j c_{ij} p_{ij} y_{ij}, \text{ all } i && (c)
\end{aligned} \tag{16}$$

where $y_{ij} \in \{0, 1\}$. 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

$$M^*_{hi} - \left(\sum_{j \in J_{hi}} p_{ij}(1 - y_{ij}) + \max_{j \in J_{hi}}\{d_j\} - \min_{j \in J_{hi}}\{d_j\} \right)$$

if one or more tasks are removed, and is M^*_{hi} otherwise. This lower bounding function can be linearized to obtain the Benders cuts

$$\left.\begin{aligned} & M \geq M^*_{hi} - \sum_{j \in J_{hi}} (1 - y_{ij}) p_{ij} - w_{hi} \\ & w_{hi} \leq \left(\max_{j \in J_{hi}}\{d_j\} - \min_{j \in J_{hi}}\{d_j\} \right) \sum_{j \in J_{hi}} (1 - y_{ij}) \\ & w_{hi} \leq \max_{j \in J_{hi}}\{d_j\} - \min_{j \in J_{hi}}\{d_j\} \end{aligned}\right\} \text{ all } i,\ h = 1, \ldots, H-1 \tag{17}$$

Thus (17) replaces (16b) when the deadlines differ and all release times are equal. For purposes of computational testing, however, we focus on the case in which all time windows are the same, since this seems to be the more important case, and it permits simpler and stronger Benders cuts.

The Benders cuts in (16) and (17) can be strengthened much as in the minimum cost algorithm. The CP solver determines the minimum makespan on a given facility by finding the largest upper bound $\overline{M}$ on makespan for which the scheduling problem is infeasible; in this case the minimum makespan is $\overline{M} + 1$. By analyzing the infeasibility proof as in minimum cost problem, we may find a conflict set $\bar{J}_{hi}$ that is smaller than J_{hi}. We can then obtain stronger cuts by replacing J_{hi} with $\bar{J}_{hi}$ in the Benders cuts of (16) and (17).

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

Random instances were generated as follows. 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 $5\alpha n(m+1)/m$ (rounded to the nearest integer). We used $\alpha = 1/3$, 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.

Table I displays computational results for 2, 3 and 4 facilities as the number of tasks increases. The CP solver is consistently faster than MILP, and in fact MILP is not shown for the makespan problems due to its relatively poor performance. 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 tasks. Presumably the advantage would be greater for larger problems.

As the number of tasks increases into the 20s, the Benders subroblems reach a size at which the computation time for the scheduling subproblem dominates and eventually 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 often accelerate the solution of the scheduling subproblems. Easier subproblems could allow the Benders method to deal with larger numbers of tasks. This hypothesis was tested by adding precedence constraints to the problem instances described above; see Table II. This resulted in fast solution of the scheduling subproblems, except in the three largest makespan instances, which we omit because they do not test the hypothesis. Easier subproblems in fact allow solution of somewhat larger instances.

Table III investigates how the Benders method scales up to a larger number of facilities, without precedence constraints. 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.

Table I. 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.

Facilities	Tasks	Min cost			Min makespan	
		MILP	CP	Benders	CP	Benders
2	10	1.9	0.14	0.09	0.80	0.08
	12	199	2.2	0.06	4.0	0.39
	14	1441	79	0.04	299	7.8
	16	3604+[a]	1511	1.1	3737	30
	18		7200+	7.0	7200+	461
	20			85		2656
	22			1674+		
3	10	0.86	0.13	0.37	0.85	0.06
	12	797	2.6	0.55	7.5	0.3
	14	114	35	0.34	981	0.7
	16	678[a]	1929	4.5	4414	6.5
	18		7200+	15	7200+	13.3
	20			2.9		34
	22			23		3084+
	24			53		
4	10	2.0	0.10	0.6	0.07	0.09
	12	7.2	1.4	4.0	1.9	0.09
	14	158	72	2.8	524	0.8
	16	906[a]	344	0.8	3898	0.9
	18		6343+	5.2	7200+	14
	20			2.6		25
	22			22		472
	24			114		1131
	26			76		

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

Since the subproblems remain relatively small as the problem size increases, it is possible for Benders to accommodate more tasks than in Table I. The advantage relative to CP or MILP is substantial, since the Benders approach extends solubility from about 16 tasks to 40 or so in the case of minimum cost problems, and to 30 or so in the case of minimum makespan. However, the number of iterations tends to increase with the number of facilities. Since each iteration adds more Benders cuts to the master problem, the computation time for solving the master problem dominates in larger problems.

Table II. Computational results for minimum cost and minimum makespan problems on two facilities with precedence constraints, using the Benders method. Computation time and number of iterations are shown for individual problem instances. Computation was cut off after 600 seconds. Minimum makespans are also given, except when computation is terminated prematurely, in which case lower and upper bounds are shown. In such cases a feasible solution with makespan equal to the upper bound is obtained.

Tasks	Minimum Cost		Minimum Makespan		
	sec	iter.	makespan	sec	iter.
12	0.02	3	17	0.2	14
14	0.05	5	13	0.3	17
16	0.5	24	27	1.6	32
18	0.02	15	27	25	152
20	0.9	39	37	0.7	13
22	0.7	34	26–27	600	480
24	7.7	115	32	13	92
26	2.1	24	37	442	268
28	21	162	35-37	600	351
30	73	261	50-53	600	206
32	>600	>666			
34	>600	>602			
36	235	318			

Table III 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 IV displays the bounds obtained for the minimum makespan problems of Table III that were not solved to optimality.

11. Conclusions and Future Research

We find that logic-based Benders decomposition can substantially improve on the state of the art when solving minimum-cost and minimum-makespan planning and scheduling problems, in the latter case when all tasks have the same release date and deadline.

In the case of minimum makespan problems, the Benders approach has the additional advantage that it can be terminated early while still

Table III. Computation times in seconds and number of iterations for minimum cost and minimum makespan problems, using the Benders method. Each figure represents the average of 5 instances.

Tasks	Facilities	Min cost	Min makespan
10	2	0.1	0.2
15	3	0.7	1.6
20	4	50[a]	13
25	5	2.9	213
30	6	4.8	3373+
35	7	128	6404+
40	8	1792+	7200+

[a]Includes one outlier that ran for 240 sec and 191 iterations.

Table IV. Best solution value and lower bound found after 7200 seconds of computation by the Benders method on the minimum makespan problems of Table III that were not solved to optimality.

Tasks	Facilities	Best solution value	Lower bound
30	6	13	12
35	7	11	10
35	7	15	13
40	8	14	11
40	8	15	12
40	8	16	13
40	8	10	9
40	8	13	11

yielding both a feasible solution and a lower bound on the optimal makespan. The bound improves steadily as the algorithm runs.

Several issues remain for future research. One is whether effective Benders cuts be developed for minimum makespan problems in which tasks have different release dates. Another is whether a Benders method can be extended to other objectives, such as minimum tardiness or minimum number of late tasks. A third is whether a branch-and-check approach to solving the Benders master problem would significantly

improve performance on planning and scheduling problems. The experience of Thorsteinsson [17] suggests that it would.

We indicated how access to "dual" information from the CP solver (results of edge finding, etc.) can result in more effective Benders cuts. A solver can provide this kind of information if it develops an explanation for a solution along with the solution itself. Explanations have attracted recent interest in the CP community (e.g. [13, 14, 15, 16]), and their use as nogoods in Benders and other search methods provides an additional reason to pursue this line of research.

References

1. Baptiste, P., C. Le Pape, and W. Nuijten, *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*, Kluwer (2001).
2. Benders, J. F., Partitioning procedures for solving mixed-variables programming problems, *Numerische Mathematik* **4** (1962): 238-252.
3. Cambazard, H., P.-E. Hladik, A.-M. Déplanche, N. Jussien, and Y. Trinquet, Decomposition and learning for a hard real time task allocation algorithm, in M. Wallace, ed., *Principles and Practice of Constraint Programming (CP 2004)*, *Lecture Notes in Computer Science* **3258**, Springer (2004).
4. De Siqueira N., J. L., and J.-F. Puget, Explanation-based generalisation of failures, *European Conference on Artificial Intelligence (ECAI-88)*, Pitman Publishers (Munich, 1988) 339-344.
5. Eremin, A., and M. Wallace, 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 (2001). 318-325.
6. Geoffrion, A. M., Generalized Benders decomposition, *Journal of Optimization Theory and Applications* **10** (1972): 237–260.
7. Hooker, J. N., *Logic-based Methods for Optimization: Combining Optimization and Constraint Satisfaction*, John Wiley & Sons (2000).
8. Hooker, J. N., 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 (2004).
9. Hooker, J. N. and G. Ottosson. Logic-based Benders decomposition, *Mathematical Programming* **96** (2003) 33–60.
10. Hooker, J. N., and Hong Yan, 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, 1995) 267–288.
11. Jain, V., and I. E. Grossmann, Algorithms for hybrid MILP/CP models for a class of optimization problems, *INFORMS Journal on Computing* **13** (2001) 258–276.
12. Junker, U., Quickxplain: Conflict detection for arbitrary constraint propagation algorithms, *IJCAI01 Workshop on Modeling and Solving Problems with Constraints (CONS-1)* (Seattle, 2001).

13. Jussien, N., The versatility of using explanations withing constraint programming, research report 03-04-INFO, École des Mines de Nantes, France (2003).
14. Jussien, N., and S. Ouis, User-friendly explanations for constraint programming, In *ICLP01 11th Workshop on Logic Programming Environments (WLPE01)*, Paphos, Cyprus, 2001.
15. Ouis, S., N. Jussien, and P. Boizumault, k-relevant explanations for constraint programming, *Sixteenth international Florida Artificial Intelligence Research Society Conference (FLAIRS 03)*, AAAI Press (2003).
16. Sqalli, M. H., and E. C. Freuder, Inference-based constraint satisfaction supports explanation, *National Conference on Artificial Intelligence (AAAI 96)* (1996).
17. Thorsteinsson, E. S., Branch-and-Check: A hybrid framework integrating mixed integer programming and constraint logic programming, T. Walsh, ed., *Principles and Practice of Constraint Programming (CP 2001)*, *Lecture Notes in Computer Science* **2239**, Springer (2001) 16–30.
18. Türkay, M., and I. E. Grossmann, Logic-based MINLP algorithms for the optimal synthesis of process networks, *Computers and Chemical Engineering* **20** (1996) 959–978.