The Knowledge Engineering Review, Vol. 31:5, 440–451. © Cambridge University Press, 2016
doi:10.1017/S0269888916000254
First published online 20 December 2016

Logic-based Benders decomposition for planning and scheduling: a computational analysis

ANDRE A. CIRÉ[1], ELVIN ÇOBAN[2] and JOHN N. HOOKER[3]

[1]*Rotman School of Management, University of Toronto, 105 St. George St, Canada;*
e-mail: AndreCire@rotman.utoronto.ca;
[2]*Özyeğin University, Istanbul, Turkey;*
e-mail: elvin.coban@ozyegin.edu.tr;
[3]*Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, USA;*
e-mail: jh38@andrew.cmu.edu

Abstract

Logic-based Benders decomposition (LBBD) has improved the state of the art for solving a variety of planning and scheduling problems, in part by combining the complementary strengths of constraint programming and mixed integer programming (MIP). We undertake a computational analysis of specific factors that contribute to the success of LBBD, to provide guidance for future implementations. We study a problem class that assign tasks to multiple resources and poses a cumulative scheduling problem on each resource. We find that LBBD is at least 1000 times faster than state-of-the-art MIP on larger instances, despite recent advances in the latter. Further, we conclude that LBBD is most effective when the planning and scheduling aspects of the problem are roughly balanced in difficulty. The most effective device for improving LBBD is the inclusion of a subproblem relaxation in the master problem. The strengthening of Benders cuts also plays an important role when the master and subproblem complexity are properly balanced. These findings suggest future research directions.

1 Introduction

Logic-based Benders decomposition (LBBD) is a generalization of classical Benders decomposition that can be applied to a much wider variety of combinatorial optimization problems. LBBD is particularly attractive for planning and scheduling, where it can combine mixed integer programming (MIP) and constraint programming (CP) in a way that exploits their relative strengths. Implementations of this method have obtained computational results superior to those of state-of-the-art MIP and CP solvers, sometimes by several orders of magnitude.

However, a computational analysis of the precise factors responsible for this success has never been conducted. In particular, applications of LBBD typically strengthen the Benders cuts in various ways, as well as including a subproblem relaxation in the master problem, on the assumption that these techniques improve performance. We undertake here a systematic study of their actual impact. In addition, because MIP technology has improved markedly in recent years, we compare LBBD with a recent state-of-the-art commercial MIP solver (CPLEX) to determine whether the advantage of LBBD persists.

We focus attention on a basic planning and scheduling problem in which tasks are assigned to resources and then scheduled on those resources. The tasks assigned to a given resource can run concurrently so long as the total rate of resource consumption never exceeds a given maximum (cumulative scheduling). The assignment problem is solved by MIP and the scheduling problem by CP.

2 Previous work

LBBD was introduced by Hooker (1995) and a general theory was presented in Hooker (2000) and Hooker and Ottosson (2003). A guide to applying LBBD to planning and scheduling problems is provided in Hooker (2007, 2012).

Classical Benders decomposition derives Benders cuts from dual or Lagrange multipliers in the subproblem (Benders, 1962; Geoffrion, 1972). However, this presupposes that the subproblem is a linear programming (LP) or nonlinear programming problem. LBBD has the advantage that Benders cuts can, at least in principle, be obtained from a subproblem of any form by solving its *inference dual* (Hooker, 1996). The solution of the dual is a *proof* of optimality for fixed values of the master problem variables (whence the name 'logic based'). The core idea of Benders decomposition is that *this same proof* may establish a bound on the optimal value when the master problem variables take other values. The corresponding Benders cut enforces this bound in the master problem.

Logic-based Benders cuts must be designed specifically for each class of problems, but this provides an opportunity to exploit problem structure. The Benders framework is also natural for combining MIP and CP, because one method can be used to solve the master problem and the other the subproblem. This is particularly advantageous when the subproblem is a scheduling problem, for which CP methods are well suited (Baptiste *et al.*, 2001; Hooker, 2012). The combinatorial nature of the scheduling problem is no longer a barrier to generating Benders cuts.

The computational advantages of LBBD have been demonstrated in a number of studies, including Benini *et al.* (2005), Cambazard *et al.* (2004), Chu and Xia (2004), Çoban and Hooker (2013), Corréa *et al.* (2004), Fazel-Zarandi and Beck (2009), Harjunkoski and Grossmann (2001, 2002), Heching and Hooker (2016), Hooker (2004, 2005a, 2005b, 2006, 2007), Jain and Grossmann (2001), Maravelias and Grossmann (2004), Terekhov *et al.* (2007), Thorsteinsson (2001), Timpe (2002), and Yunes *et al.* (2010).

We presented some of the computational results given here at a recent CPAIOR conference (Ciré *et al.*, 2013). In this paper, we present more detailed results, including performance profiles, iteration counts, a computation time breakdown by master and subproblem, and the performance of Benders cuts without a subproblem relaxation. We also analyze the implications of these results.

3 Fundamentals

LBBD is based on the concept of an *inference dual*. Consider an optimization problem:

$$\begin{aligned} &\min f(x) \\ &C(x) \\ &x \in D \end{aligned} \tag{1}$$

where $C(x)$ represents a constraint set containing variables x, and D is the domain of x (such as $\mathbb{R}^n$ or $\mathbb{Z}^n$). The inference dual is the problem of finding the tightest lower bound on the objective function that can be deduced from the constraints:

$$\begin{aligned} &\max\ v \\ &C(x) \overset{P}{\vdash} (f(x) \geqslant v) \\ &v \in \mathbb{R},\ \ P \in \mathcal{P} \end{aligned} \tag{2}$$

Here $C(x) \overset{P}{\vdash} (f(x) \geqslant v)$ indicates that proof P deduces $f(x) \geqslant v$ from $C(x)$. The domain of variable P is a family $\mathcal{P}$ of proofs, and the dual solution is a pair (v, P). When the primal problem (1) is a feasibility problem with no objective function, the dual can be viewed as the problem finding a proof P of infeasibility.

If problem (1) is a LP problem $\min\{cx \mid Ax \geqslant b, x \geqslant 0\}$, the inference dual becomes the classical LP dual (assuming feasibility) for an appropriate proof family $\mathcal{P}$. Namely, each proof P corresponds to a tuple $u \geqslant 0$ of multipliers, and P deduces the bound $cx \geqslant v$ when the surrogate $uAx \geqslant ub$ dominates $cx \geqslant v$; that is, $uA \leqslant c$ and $ub \geqslant v$. The dual therefore maximizes v subject to $uA \leqslant c$, $ub \geqslant v$, and $u \geqslant 0$. Equivalently, it maximizes ub subject to $uA \leqslant c$ and $u \geqslant 0$, which is the classical LP dual.

LBBD applies to problems of the form:

$$\begin{aligned} &\min f(x, y) \\ &C(x, y) \\ &x \in D_x,\ y \in D_y \end{aligned} \tag{3}$$

Fixing x to $\bar{x}$ defines the subproblem:

$$\begin{aligned} &\min f(\bar{x}, y) \\ &C(\bar{x}, y) \\ &y \in D_y \end{aligned} \tag{4}$$

Let proof P solve the inference dual of the subproblem by deducing the bound $f(\bar{x}, y) \geqslant v^*$. A Benders cut $v \geqslant B_{\bar{x}}(x)$ is derived by identifying a bound $B_{\bar{x}}(x)$ that the same proof P deduces for any given x. Thus, in particular, $B_{\bar{x}}(\bar{x}) = v^*$. The kth master problem is

$$\begin{aligned} &\min v \\ &v \geqslant B_{x^i}(x),\ i = 1, \ldots, k-1 \\ &x \in D_x \end{aligned} \tag{5}$$

where $x^1, \ldots, x^{k-1}$ are the solutions of the first $k-1$ master problems. If the subproblem is a feasibility problem with no objective function, the Benders cut is a constraint that excludes $\bar{x}$ and perhaps other solutions that proof P shows to be infeasible.

At this point we solve the master problem and set $\bar{x}$ equal to an optimal solution of the master, whereupon the process repeats. The algorithm terminates when the optimal value v_k of the master problem is equal to the minimum of $B_{x^i}(x^i)$ over $i = 1, \ldots, k$. At any stage of the algorithm, v_k is a lower bound on the optimal value of (3), and each $B_{x^i}(x^i)$ is an upper bound.

Classical Benders decomposition is the result of applying LBBD to a problem of the form:

$$\begin{aligned} &\min f(x) + cy \\ &g(x) + Ay \geqslant b \\ &x \in D_x,\ y \geqslant 0 \end{aligned} \tag{6}$$

The subproblem is an LP:

$$\begin{aligned} &\min f(\bar{x}) + cy \\ &Ay \geqslant b - g(\bar{x}) \\ &y \geqslant 0 \end{aligned} \tag{7}$$

whose inference dual is the LP dual. Its solution u defines a surrogate $uAy \geqslant u(b - g(\bar{x}))$ that dominates $cy \geqslant v^*$ and therefore deduces that $f(\bar{x}) + cy \geqslant f(\bar{x}) + u(b - g(\bar{x}))$. The same u deduces $f(x) + cy \geqslant f(x) + u(b - g(x))$ for any x, and we have the classical Benders cut $v \geqslant f(x) + u(b - g(x))$. When the subproblem is infeasible, the dual has an extreme ray solution u that proves infeasibility because $uA \leqslant 0$ and $u(b - g(x)) > 0$. The Benders cut is therefore $u(b - g(x)) \leqslant 0$.

In practice, the solution of the subproblem inference dual is the proof of optimality obtained while solving the subproblem. The simplest type of Benders cut is a *nogood cut*, which states that the solution of the subproblem cannot be improved unless at least one x_j is fixed to a different value. This yields a nogood cut:

$$v \geqslant \begin{cases} v^* & \text{if } x_j = \bar{x}_j \text{ for all } j \in J \\ -\infty & \text{otherwise} \end{cases} \tag{8}$$

where J is the index set for the x_js. If the subproblem is infeasible, the nogood cut states simply that $x_j \neq \bar{x}_j$ for some $j \in J$.

Nogood cuts can be strengthened in various ways. The simplest examines the dual proof and observes that only the variable settings $x_j = \bar{x}_j$ for $j \in \bar{J}$ appear as premises. This yields a strengthened nogood cut by replacing J with $\bar{J}$ in (8). If the dual proof is not accessible, we can strengthen the cuts by re-solving the subproblem when some of the premises $x_j = \bar{x}_j$ are dropped, and checking whether the optimal subproblem value is the same. Further analysis of the optimality proof may yield *analytic* Benders cuts that provide useful bounds when $x_j \neq \bar{x}_j$ for some of the $j \in J$.

4 Logic-based Benders decomposition for planning and scheduling

The problem is to assign each of n tasks to one of m resources so as to minimize cost, subject to the condition that the tasks assigned to each resource can be feasibly scheduled on that resource. Each task j must be processed within time window $[L_j, U_j]$, has processing time p_{ij} on resource i, and consumes resource i at the rate r_{ij}. The total rate of resource i consumption can be at most L_i. It is assumed that there is a fixed cost c_{ij} for processing task j on resource i.

The master problem variables x become binary variables δ_{ij}, where δ_{ij} is 1 when task j is assigned to resource i. The master problem (5) is

$$\begin{aligned} &\min \sum_{ij} c_{ij}\delta_{ij} \\ &\sum_i \delta_{ij} = 1, \quad j = 1, \ldots, n \\ &\text{Benders cuts} \\ &\text{Subproblem relaxation} \\ &\delta_{ij} \in \{0, 1\}, \quad \text{all } i, j \end{aligned} \tag{9}$$

The subproblem decouples into a separate scheduling problem for each resource. The subproblem variables y are the start time s_j of task j. Let J_i be the set of tasks assigned to resource i in the solution of the master problem. Let the constraint cumulative(s, p, r, R) require that tasks be scheduled at start times $s = (s_1, \ldots, s_k)$ so that the resource consumption rate never exceeds R. The subproblem for resource i is the feasibility problem:

$$\begin{aligned} &L_j \leqslant s_j \leqslant U_j - p_{ij}, \quad \text{all } j \in J_i \\ &\text{cumulative}\left((s_j, j \in J_i), (p_{ij}, j \in J_i), (r_{ij}, j \in J_i), R_i\right) \end{aligned} \tag{10}$$

We generate a Benders cut for each resource for which the assignment is infeasible. A simple nogood cut is as described above, which in the present context becomes

$$\sum_{j \in J_i} (1 - \delta_{ij}) \geqslant 1 \tag{11}$$

Stronger cuts would ideally be obtained by examining the proof of infeasibility when the subproblem is solved. For example, if infeasibility is proved by edge finding (Baptiste *et al.*, 2001), or edge finding plus branching, one could observe the set $\bar{J}_i$ of tasks that actually play a role in the proof, and replace J_i with $\bar{J}_i$ in (11). Unfortunately, this kind of information is not available from the commercial CP software used to solve the scheduling problem. We therefore strengthen the nogood cut heuristically as indicated earlier. We remove elements from J_i one at a time and re-solve the subproblem until the scheduling problem becomes feasible. We then restore the last element removed, and the resulting $\bar{J}_i$ becomes the basis for the strengthened nogood cut (Hooker, 2005b, 2007).

We also tighten the master problem (9) with a relaxation of the subproblem. We use a very simple relaxation that requires that the processing times of tasks assigned to resource i fit between the earliest release time and the lastest deadline:

$$\sum_j p_{ij}\delta_{ij} \leqslant \max_j \{U_j\} - \min_j \{L_j\}$$

The subproblem is a feasibility problem because we are minimizing assignment cost only. An optimal solution is obtained as soon as the assignment is feasible on all resources. As a result, the Benders method yields no upper bound on the optimal value until it terminates with an optimal solution. This is not the case with other common objectives, such as minimizing makespan, minimizing the number of late tasks, or minimizing total tardiness. These objectives result in subproblems that are optimization problems, and they require different (and more complex) Benders cuts and subproblem relaxations. They are discussed in Hooker (2005b, 2007).

5 Computational results

The problem instances are those used in Hooker (2004, 2005b, 2007) and elsewhere. The instances and full documentation are available at http://web.tepper.cmu.edu/jnh/instances.htm

They consist of ‘c’ instances, which schedule 10–32 tasks on two to four resources, and ‘e’ instances, which schedule 10–50 tasks on 2–10 resources.

The ‘c’ instances are designed to be difficult for LBBD, because the processing times on different resources differ by as much as a factor of m when there are m resources. This causes many more tasks to be assigned to the more efficient resources, which results in a computational bottleneck on those resources. There are also fewer resources, resulting in less decoupling of the subproblem. The ‘e’ instances are more suited to LBBD, even though they are larger. Processing times differ by at most a factor of 1.5 across resources, so that the load is more evenly (and perhaps more realistically) balanced.

We implemented LBBD by solving the master problem with CPLEX 12.4.01 and the subproblems with IBM CP Optimizer 12.4.01 using extended filtering, DFS search, and default variable and value selection. We also solved the instances by pure MIP, using the best known formulation and CPLEX 12.4.01. We did not solve the instances with pure CP, because previous experience indicates that CP solvers are considerably slower than MIP on these instances (Hooker, 2005b, 2007). All tests are run on an Intel Xeon E5345 2.33 GHz (64 bits) in single core mode with 8 GB RAM.

Table 1 shows our results for the ‘c’ instances. LBBD is clearly superior to MIP, running at least 1000 times faster on the larger instances with more than two resources. The advantage is less pronounced when there are only two resources, which is not surprising because there is less decoupling of the subproblem. The superiority of LBBD is equally evident in the performance profile of Figure 1.

The superiority of LBBD is even clearer for the ‘e’ instances, as revealed in Table 2 and the performance profile in Figure 2.

The importance of the subproblem relaxation is equally evident in the results. If strengthened nogood cuts are used without a relaxation, the advantage of LBBD is substantially reduced, and it disappears completely in some instances.

Strengthening of the nogood cuts, however, is not always effective. It actually worsens performance in ‘c’ instances with two resources, and it brings rather limited improvement in the overall performance profile. Even when the instances with two resources are removed from the profile, as in Figure 3, the effect of strengthening is not as great as one might expect. On the other hand, cut strengthening is significantly more effective in the ‘e’ instances, as is evident in Figure 2.

6 Analysis of results

More detailed computational data can shed light on the results described above. Tables 3 and 4 show the average number of iterations for each instance size, as well as a breakdown of the average solution times by master problem and subproblem.

We can immediately see the effect of the subproblem relaxation. The lack of a relaxation results in a dramatic increase in the number of iterations and therefore in the solution time. This is intuitively reasonable, because a Benders method in effect computes the projection of the feasible set onto the master problem variables. As more Benders cuts are added, the projection is more accurately described. Eventually, enough cuts are added to describe the projection completely, and the problem can be solved by solving only the master problem. It is impractical to generate so many cuts, however, and the effectiveness of the Benders method rests on the fact that cut generation is guided by interim solutions of the master problem. Cuts tend to be generated only in the vicinity of the optimal solution, and this is enough to solve the problem. A relaxation of the subproblem is useful because it more tightly circumscribes the projection from the start, so that many fewer cuts are necessary to chip away regions near the optimum that are not part of the projection.

The absence of a relaxation not only generates more iterations, but each iteration requires longer to solve. This is because more iterations produce more Benders cuts, which makes the master problem larger and harder to solve. The tables suggest that a successful application of the Benders method tends to result in well under 100 iterations. After this point, solution of the master bogs down, and the method tends to fail.

Table 1 Computational results for the ‘c’ instances, which are designed to be difficult for logic-based Benders decomposition (LBBD)

Size		MIP (CPLEX)		LBBD: strong cuts only		LBBD: relax + weak cuts		LBBD: relax + strong cuts	
m	n	Solved	Seconds	Solved	Seconds	Solved	Seconds	Solved	Seconds
2	10	5	0.1	5	0.2	5	0.1	5	0.1
	12	5	0.2	5	0.2	5	0.1	5	0.0
	14	5	0.1	5	0.4	5	0.1	5	0.0
	16	5	28	5	2.0	5	0.2	5	0.3
	18	5	388	5	19	5	0.5	5	0.7
	20	4	1899	5	120	5	2.0	5	8.0
	22	3	3844$^+$	4	1852$^+$	5	617	5	955
	24	2	4346$^+$	1	6341$^+$	4	1495$^+$	4	1936$^+$
	26	1	6362$^+$	0	—	5	327	4	1642$^+$
	28	2	4384$^+$	0	—	5	1004	5	1133
	30	0	—	0	—	2	5391$^+$	2	5761$^+$
	32	1	5813$^+$	0	—	2	4325$^+$	2	4325$^+$
3	10	5	0.0	5	0.2	5	0.1	5	0.1
	12	5	0.1	5	0.4	5	0.5	5	0.1
	14	5	0.3	5	1.2	5	0.3	5	0.2
	16	5	13	5	5.6	5	2.7	5	0.8
	18	5	548	5	22	5	7.8	5	1.4
	20	4	1712$^+$	5	30	5	1.2	5	0.5
	22	3	3674$^+$	5	59	5	7.5	5	2.6
	24	2	4411$^+$	4	1739$^+$	5	15	5	5.7
	26	0	—	4	3510$^+$	5	191	5	98
	28	2	5238$^+$	2	6645$^+$	5	270	5	209
	30	0	—	0	—	4	2354$^+$	4	1856$^+$
	32	0	—	0	—	2	4667$^+$	2	4751$^+$
4	10	5	0.0	5	0.1	5	0.0	5	0.0
	12	5	0.1	5	0.2	5	0.1	5	0.1
	14	5	0.3	5	0.6	5	1.0	5	0.3
	16	5	1.0	5	0.6	5	0.4	5	0.1
	18	5	36	5	4.0	5	1.7	5	0.4
	20	5	523	5	11	5	1.1	5	0.3
	22	5	811	5	75	5	8.2	5	1.1
	24	1	6292$^+$	5	122	5	23	5	9.1
	26	0	—	3	3369$^+$	5	19	5	7.4
	28	1	5762$^+$	3	4623$^+$	5	36	5	11
	30	0	—	2	4841$^+$	5	430	5	61
	32	0	—	0	—	5	680	5	478

n Tasks are scheduled on m resources. The number of problem instances solved (out of five) and average computation time in seconds are shown. Results are shown for the CPLEX mixed integer programming (MIP) solver, LBBD with strengthened cuts but no subproblem relaxation in the master problem, LBBD with a relaxation and simple nogood cuts, and LBBD with a relaxation and strengthened cuts.

$^+$Computation terminated after 7200 seconds for instances not solved to optimality. In cases where CPLEX terminated prematurely due to lack of memory, the computation time was set at 7200 seconds when computing the average time for CPLEX.

The results also reveal why the ‘c’ instances are harder for LBBD. Once the number of tasks reaches a certain point, the subproblem solution time explodes, even while the master problem solution time remains small. This point is reached earlier when there are fewer resources, indicating that the key factor is the average number of tasks per resource. In fact, the break point is about 10 tasks/resource. Due to the uneven loads, this places subtantially more than 10 tasks on the fastest resource, whereupon the CP

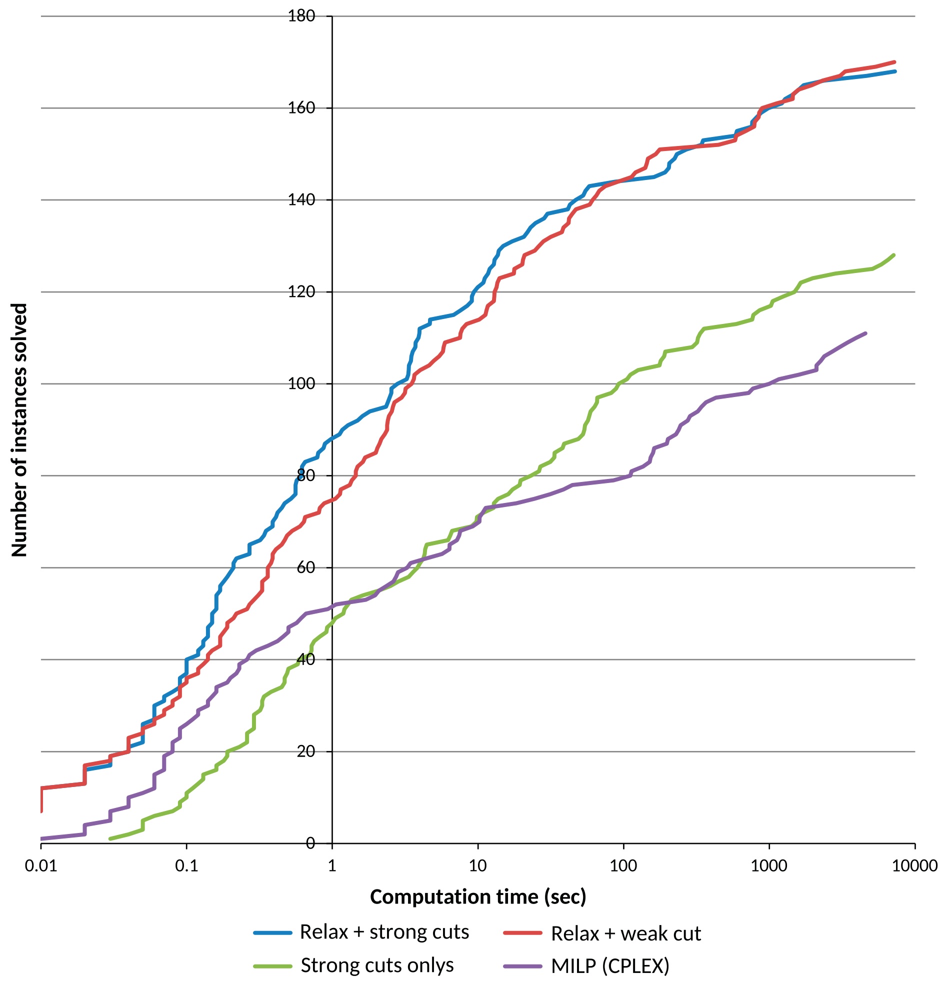

Figure 1 Performance profile for all 180 'c' instances

Table 2 Computational results for the 'e' instances

Size		MIP (CPLEX)		LBBD: relax + weak cuts		LBBD: relax + strong cuts	
m	*n*	Solved	Seconds	Solved	Seconds	Solved	Seconds
2	10	5	0.1	5	0.1	5	0.1
2	12	5	0.3	5	0.3	5	0.1
3	15	5	0.9	5	0.4	5	0.2
4	20	5	46	5	14	5	1.9
5	25	5	73	5	1.0	5	0.7
6	30	5	543	5	1.3	5	0.4
7	35	2	5122^{+}	5	36	5	2.7
8	40	1	7246^{+}	4	1527^{+}	5	80
9	45	0	—	5	1050	5	35
10	50	1	6983^{+}	5	45	5	5.4

LBBD = logic-based Benders decomposition; MIP = mixed integer programming.
$^{+}$Computation terminated after 7200 seconds for instances not solved to optimality.

problem becomes highly combinatorial, and solution time explodes. This phenomenon does not occur in the 'e' instances, where there are only five tasks per resource, and the resource loads are more evenly balanced.

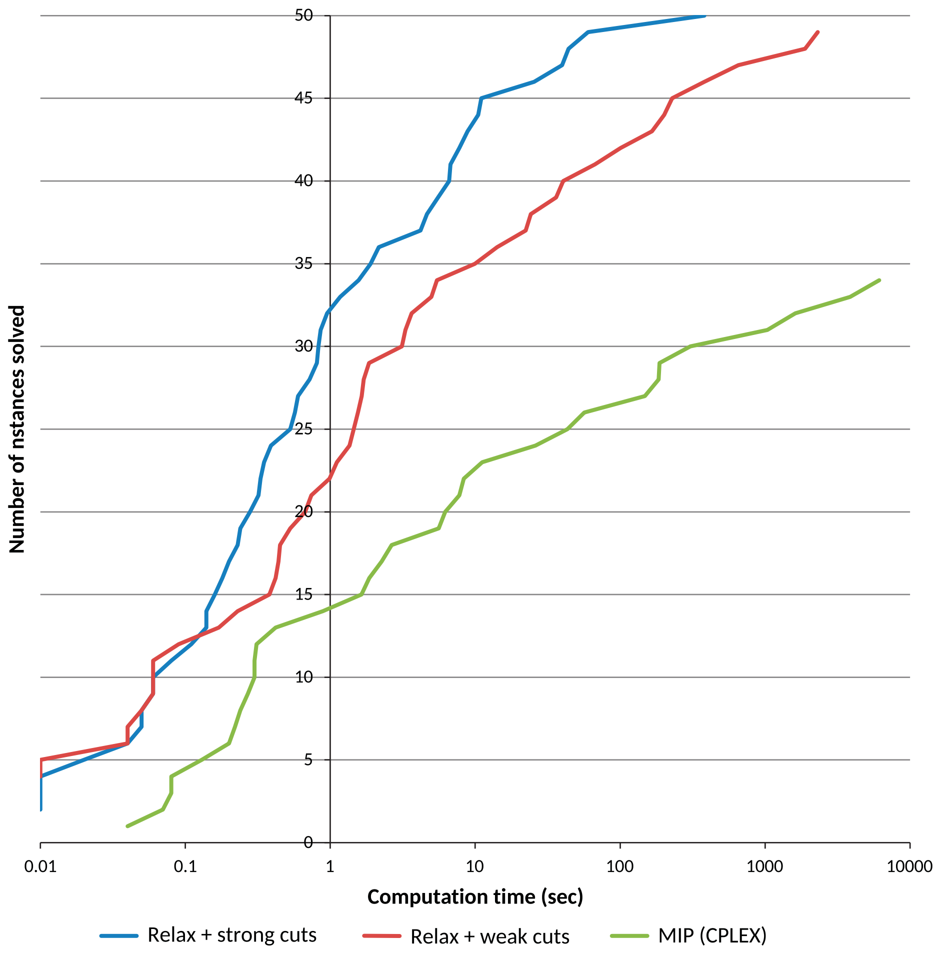

Figure 2 Performance profile for all 50 ‘e’ instances. MIP = mixed integer programming

From a broader perspective, the ‘c’ instances are less suitable for LBBD because the assignment problem is relatively easy when there are only two or three resources. Most of the combinatorial complexity is relegated to the scheduling subproblem, which can quickly become intractable as the instances scale up. In the ‘e’ instances, the assignment problem bears a more equal share of the combinatorial burden. Decomposition is more effective because the labor is more equally shared between master and subproblem.

It is less obvious how the effect of cut strengthening can be explained, but a pattern is visible. We first observe that strong cuts tend to result in greater subproblem solution time per iteration, because the subproblem is solved repeatedly in each iteration to tighten the cuts. However, this tends to be more than offset by the smaller number of iterations. As one might expect, stronger cuts remove more solutions that lie outside the projection, and fewer cuts are therefore necessary. The key observation is that when there are fewer resources relative to tasks, the subproblem relaxation already substantially reduces the number of iterations. This means that there is less room for further reduction due to strong cuts. This is particularly evident in the ‘c’ instances with only two or three resources (Table 3).

The data further suggest that cut generation should result in a rough balance between the total master solution time and the total subproblem solution time. This is particularly evident for the ‘e’ instances (Table 4), for which weak cuts result in nearly 0 subproblem solution time, while strong cuts roughly equalize the master and subproblem time in most cases. LBBD is somewhat less efficient for instances with eight and nine resources, for which the master solution time is significantly greater than the subproblem time. This suggests that one should invest more time to generate stronger cuts for these instances.

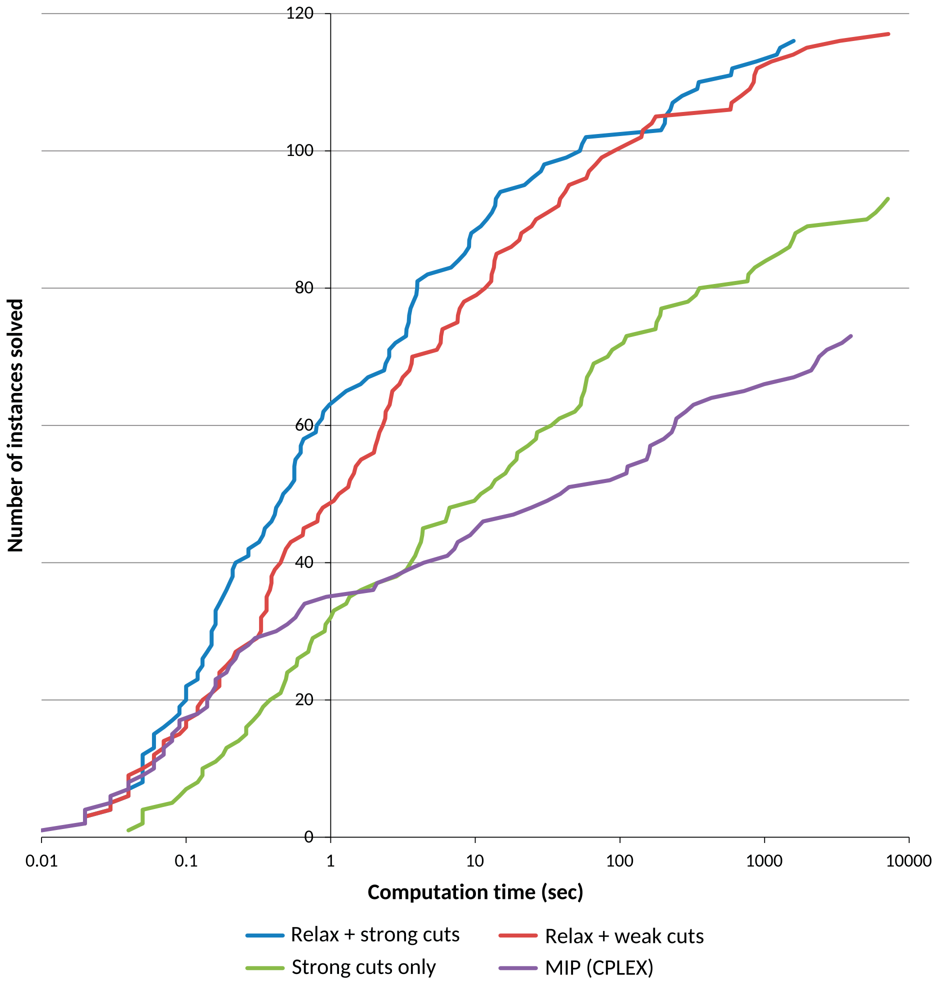

Figure 3 Performance profile for the 120 'c' instances with three and four resources. MIP = mixed integer programming

7 Conclusions

Based on the experiments described above, we conclude that LBBD is more effective when the problem in fact decomposes. The planning and scheduling components should both embody substantial shares of the problem's combinatorial complexity.

When the planning portion involves the assignment of tasks to resources, the average number of tasks per resource should be small enough that the assignment problem is roughly as hard as the scheduling problem. In particular, it should remain below the threshold at which solution time of the scheduling problem tends to explode. LBBD failure most often occurs when the CP solver blows up because of too many tasks are assigned to the resource. This is less likely to occur when no resource is significantly faster than others.

We also find that the most effective technique for reducing solution time is to include a relaxation of the subproblem in the master problem. This can dramatically reduce the number of iterations, as well as the solution time per iteration, because the master problems are smaller when there are fewer iterations.

The identification of stronger cuts can also significantly reduce the number of iterations, especially when the master and subproblem complexity are properly balanced. A rough guideline is to invest greater time in identifying strong cuts when the subproblem solution time per Benders iteration is less than the master problem solution time.

In any event, LBBD remains substantially faster than state-of-the art MIP on the problem class studied here. This is despite the marked improvement of MIP solvers, already highly advanced, over the last few

Table 3 Computational analysis of logic-based Benders decomposition for 'c' instances, showing the average number of iterations and the average computation time spent solving master problems and subproblems

		Strong cuts only			Relax + weak cuts			Relax + strong cuts		
m	*n*	Iterations	Master (seconds)	Subproblem (seconds)	Iterations	Master (seconds)	Subproblem (seconds)	Iterations	Master (seconds)	Subproblem (seconds)
2	10	18	0.1	0.1	9.8	0.1	0.0	4.8	0.0	0.0
	12	13	0.1	0.1	5.0	0.0	0.0	3.4	0.0	0.0
	14	19	0.1	0.3	1.8	0.0	0.0	1.8	0.0	0.0
	16	41	0.5	1.5	2.0	0.0	0.2	2.0	0.0	0.3
	18	149	5.7	14	2.4	0.0	0.5	2.4	0.0	0.7
	20	107	3.5	117	3.6	0.0	2.0	2.8	0.0	8.0
	22	340[+]	70[+]	1782[+]	4.6	0.0	617	4.4	0.0	955
	24	327[+]	67[+]	6263[+]	2.0[+]	0.0[+]	1495[+]	1.8[+]	0.0[+]	1936[+]
	26	—	—	—	1.8	0.0	327	1.6[+]	0.0[+]	1642[+]
	28	—	—	—	2.0	0.0	1004	1.8	0.0	1133
	30	—	—	—	4.2[+]	0.0[+]	5391[+]	1.0[+]	1452[+]	4309[+]
	32	—	—	—	1.2[+]	0.0[+]	4325[+]	1.0[+]	0.0[+]	4325[+]
3	10	13	0.0	0.1	9.8	0.1	0.0	4.4	0.0	0.0
	12	23	0.2	0.2	14	0.4	0.0	6.4	0.1	0.1
	14	42	0.7	0.5	13	0.2	0.1	6.8	0.1	0.1
	16	86	4.0	1.5	40	2.5	0.2	17	0.5	0.3
	18	183	19	3.0	61	7.3	0.5	23	1.0	0.5
	20	226	23	6.4	21	0.8	0.4	8.2	0.1	0.4
	22	340	49	10	49	2.9	4.6	16	0.4	2.3
	24	1222[+]	1689[+]	50[+]	55	12	3.5	22	1.6	4.1
	26	1854[+]	2723[+]	786[+]	130	33	158	22	0.6	97
	28	2113[+]	3283[+]	3363[+]	15	0.2	270	8.0	0.1	209
	30	—	—	—	80[+]	9.2[+]	2344[+]	21[+]	1.1[+]	1855[+]
	32	—	—	—	143[+]	64[+]	4602[+]	23[+]	1.7[+]	4750[+]
4	10	6.8	0.0	0.1	4.6	0.0	0.0	3.0	0.0	0.0
	12	12	0.1	0.1	6.2	0.1	0.0	4.4	0.0	0.0
	14	26	0.3	0.3	22	0.9	0.1	9.0	0.2	0.1
	16	27	0.2	0.3	12	0.3	0.1	5.6	0.1	0.1
	18	74	3.0	1.0	32	1.5	0.1	15	0.3	0.2
	20	130	9.0	2.3	26	1.0	0.2	11	0.1	0.2
	22	334	69	6.6	51	7.6	0.6	15	0.7	0.5
	24	407	104	18	96	20	3.1	37	3.4	5.6
	26	1351[+]	3315[+]	54[+]	83	11	7.5	32	2.0	5.4
	28	2042[+]	4091[+]	532[+]	27	1.7	34	12	0.5	11
	30	1408[+]	4665[+]	175[+]	117	395	35	41	41	20
	32	—	—	—	60	6.3	673	14	0.4	478

[+]Computation terminated for unsolved instances after the total computation time reaches 7200 seconds.

years. The advantage exceeds a factor of 1000 for larger instances. In fact, the LBBD method presented here is best conceived as an enhancement of MIP rather than a competitor, because MIP solves the master problem, and LBBD will therefore improve as MIP improves. LBBD also benefits from advancements in CP, used here to solve the subproblem.

On the other hand, LBBD is likely to fail when the imbalance or size of the master and subproblem result in more than 100 Benders iterations.

These results suggest several research directions. When the number of tasks per resource crosses the CP solver's tractability threshold, one option is to solve the scheduling subproblem with failure-directed search, which was recently introduced in the IBM CP optimizer. Another option is to decompose the scheduling problem itself using LBBD. This can be done by dividing the time horizon into segments and

Table 4 Computational analysis of logic-based Benders decomposition for ‘e’ instances

		Relax + weak cuts			Relax + strong cuts		
m	n	Iterations	Master (seconds)	Subproblem (seconds)	Iterations	Master (seconds)	Subproblem (seconds)
2	10	9.4	0.1	0.0	5.2	0.0	0.0
2	12	13	0.3	0.0	4.4	0.0	0.0
3	15	14	0.4	0.0	5.6	0.1	0.1
4	20	55	14	0.0	16	1.7	0.3
5	25	19	0.4	0.0	8.6	0.1	0.6
5	30	26	1.1	0.0	8.8	0.2	0.2
7	35	76	34	0.0	19	2.0	0.7
8	40	107^+	1525^+	0.0^+	31	78	2.1
9	45	132	1048	0.0	39	33	2.2
10	50	39	43	0.0	18	3.6	1.7

$^+$Computation terminated for unsolved instances after the total computation time reaches 7200 seconds.

creating a subproblem for each segment. This approach has been successfully applied in the context of single-resource scheduling (Çoban & Hooker, 2013).

The overriding importance of a subproblem relaxation suggests that further research should be conducted in this area. For example, some of the subproblem constraints could be incorporated verbatim into the master problem, along with the subproblem variables that occur within them. Normally, subproblem variables do not occur in the master, but their inclusion is consistent with the Benders mechanism so long as their solution values in the master are discarded before the next subproblem is solved. Their presence in the master serves only to restrict the possible values of the master variables. This strategy has been applied in a Benders-based solution of the home health-care delivery problem (Ciré & Hooker, 2012). A practice of decomposing problems while allowing the components to intermingle could lead to a research program that improves decomposition methods in general.

One characteristic of LBBD not exploited here is that the subproblem is easily parallelized by assignining each facility to a different processor. This could be a significant advantage if there are a large number of facilities, or if the scheduling subproblem is itself decomposed.

Finally, the generation of strong Benders cuts could benefit from access to dual information in the subproblem solver. That is, the solver should report how it proved optimality or infeasibility. The cuts used here were strengthened by repeatedly re-solving the subproblem to tease out dual information, but cuts based on the actual dual solution could be much more effective.

References

Baptiste, P., Le Pape, C. & Nuijten, W. 2001. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer.

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

Benini, L., Bertozzi, D., Guerri, A. & Milano, M. 2005. Allocation and scheduling for MPSoCs via decomposition and no-good generation. In *Principles and Practice of Constraint Programming (CP 2005)*, 107–121. Springer.

Cambazard, H., Hladik, P.-E., Déplanche, A.-M., Jussien, N. & Trinquet, Y. 2004. Decomposition and learning for a hard real time task allocation problem. In *Principles and Practice of Constraint Programming (CP 2004)*, Wallace, M. (eds). Springer, 153–167.

Chu, Y. & Xia, Q. 2004. Generating Benders cuts for a class of integer programming problems. In *CPAIOR 2004 Proceedings*, Regin, J. C. & Rueher, M. (eds). Springer, 127–141.

Ciré, A., Coban, E. & Hooker, J. N. 2013. Mixed integer programming vs logic-based Benders decomposition for planning and scheduling. In *CPAIOR 2013 Proceedings*, Gomes, C. & Sellmann, M. (eds). Springer, 325–331.

Ciré, A. & Hooker, J. N. 2012. A heuristic logic-based Benders method for the home health care problem. In *Presented at Matheuristics 2012*.

Çoban, E. & Hooker, J. N. 2013. Single-facility scheduling by logic-based Benders decomposition. *Annals of Operations Research* **210**, 245–272.

Corréa, A. I., Langevin, A. & Rousseau, L. M. 2004. Dispatching and conflict-free routing of automated guided vehicles: a hybrid approach combining constraint programming and mixed integer programming. In *CPAIOR 2004 Proceedings*, Regin, J. C. & Rueher, M. (eds). Springer, 370–378.

Fazel-Zarandi, M. M. & Beck, J. C. 2009. Solving a location-allocation problem with logic-based Benders decomposition. In *Principles and Practice of Constraint Programming (CP 2009)*, Gent, I. P. (ed.). Springer, 344–351.

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

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

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

Heching, A. & Hooker, J. N. 2016. Scheduling home hospice care with logic-based Benders decomposition. In *CPAIOR 2016 Proceedings*, Gendron, B. (ed.). Springer, 187–197.

Hooker, J. N. 1995. Logic-based Benders decomposition. In *INFORMS National Meeting (INFORMS 1995).*

Hooker, J. N. 1996. Inference duality as a basis for sensitivity analysis. In *Principles and Practice of Constraint Programming (CP 1996)*, Freuder, E. C. (ed.). Springer, 224–236.

Hooker, J. N. 2000. *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. Wiley.

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

Hooker, J. N. 2005a. Planning and scheduling to minimize tardiness. In *Principles and Practice of Constraint Programming (CP 2005)*, 314–327. Springer.

Hooker, J. N. 2005b. A hybrid method for planning and scheduling. *Constraints* **10**, 385–401.

Hooker, J. N. 2006. An integrated method for planning and scheduling to minimize tardiness. *Constraints* **11**, 139–157.

Hooker, J. N. 2007. Planning and scheduling by logic-based Benders decomposition. *Operations Research* **55**, 588–602.

Hooker, J. N. 2012. *Integrated Methods for Optimization*, 2nd edition. Springer.

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

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

Maravelias, C. T. & Grossmann, I. E. 2004. Using MILP and CP for the scheduling of batch chemical processes. In *CPAIOR 2004 Proceedings*, Regin, J. C. & Rueher, M. (eds). Springer, 1–20.

Terekhov, D., Beck, J. C. & Brown, K. N. 2007. Solving a stochastic queueing design and control problem with constraint programming. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI2007)*, **1**, 261–266. AAAI Press.

Thorsteinsson, E. 2001. Branch and check: a hybrid framework integrating mixed integer programming and constraint logic programming. In *Principles and Practice of Constraint Programming (CP2001)*, Walsh, T. (ed.). Springer, 16–30.

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

Yunes, T. H., Aron, I. & Hooker, J. N. 2010. An integrated solver for optimization problems. *Operations Research* **58**, 342–356.