Logic-based Benders Decomposition for Planning and Scheduling: A Computational Analysis

Andre A. Ciré, Elvin Çoban and John N. Hooker

Version  Post-print/accepted manuscript


How to cite TSpace items

Always cite the published version, so the author(s) will receive recognition through services that track citation counts, e.g. Scopus. If you need to cite the page number of the author manuscript from TSpace because you cannot access the published version, then cite the TSpace version in addition to the published version using the permanent URI (handle) found on the record page.

This article was made openly accessible by U of T Faculty. Please tell us how this access benefits you. Your story matters.
Logic-based Benders Decomposition for Planning and Scheduling: A Computational Analysis

André A. Ciré
University of Toronto, Canada
AndreCire@rotman.utoronto.ca

Elvin Çoban
Özyeğin University, Istanbul, Turkey
elvin.coban@ozyegin.edu.tr

J. N. Hooker
Carnegie Mellon University, Pittsburgh, USA
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 (CP) 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 assigns 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. This paper is partially based on results presented by the same authors at CPAIOR 2013.

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 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.

Previous Work
Logic-based Benders decomposition was introduced by Hooker (1995) and Hooker and Yan (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 (2007b) and Hooker (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 or nonlinear programming problem. Logic-based Benders decomposition 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 MILP 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, Pape, and Nuijten 2001; Hooker 2007a). The combinatorial nature of the scheduling problem is no longer a barrier to generating Benders cuts.

The computational advantages of LBBD have been

We presented some of the computational results given here at a recent CPAIOR conference (Ciré, Çoban, and Hooker 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.

**Fundamentals**

Logic-based Benders decomposition is based on the concept of an inference dual. Consider an optimization problem

\[
\begin{align*}
\min & \quad f(x) \\
\text{subject to} & \quad C(x) \geq 0 \quad x \in D
\end{align*}
\]

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:

\[
\begin{align*}
\max & \quad v \\
\text{subject to} & \quad C(x)^P \vdash (f(x) \geq v) \\
& \quad v \in \mathbb{R}, \quad P \in \mathcal{P}
\end{align*}
\]

Here \( C(x)^P \vdash (f(x) \geq v) \) indicates that proof \( P \) deduces \( f(x) \geq v \) 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 linear programming (LP) problem \( \min \{ cx \mid Ax \geq b, \ x \geq 0 \} \), 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 \((u, c, g)\) of multipliers, and \( P \) deduces the bound \( cx \geq v \) when the surrogate \( uAx \geq ub \) dominates \( cx \geq v \); that is, \( uA \leq c \) and \( ub \geq v \). The dual therefore maximizes \( v \) subject to \( uA \leq c, \ ub \geq v, \) and \( u \geq 0 \). Equivalently, it maximizes \( ub \) subject to \( uA \leq c \) and \( u \geq 0 \), which is the classical LP dual.

Logic-based Benders decomposition applies to problems of the form

\[
\begin{align*}
\min & \quad f(x, y) \\
& \quad C(x, y) \geq 0 \\
& \quad x \in D_x, \ y \in D_y
\end{align*}
\]

Fixing \( x \) to \( \bar{x} \) defines the subproblem

\[
\begin{align*}
\min & \quad f(\bar{x}, y) \\
& \quad C(\bar{x}, y) \geq 0 \\
& \quad y \in D_y
\end{align*}
\]

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

\[
\begin{align*}
\min & \quad v \\
& \quad v \geq B_{\bar{x}}(x), \ i = 1, \ldots, k - 1 \\
& \quad x \in D_x
\end{align*}
\]

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 \( B_{\bar{x}}(x^i) \) 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 \( B_{\bar{x}}(x^i) \) is an upper bound.

Classical Benders decomposition is the result of applying logic-based Benders decomposition to a problem of the form

\[
\begin{align*}
\min & \quad f(x) + cy \\
& \quad g(x) + Ay \geq b \\
& \quad x \in D_x, \ y \geq 0
\end{align*}
\]

The subproblem is an LP:

\[
\begin{align*}
\min & \quad f(\bar{x}) + cy \\
& \quad Ay \geq b - g(\bar{x}) \\
& \quad y \geq 0
\end{align*}
\]

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

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

\[
\begin{align*}
v & \geq \begin{cases} 
v^* & \text{if } x_j = \bar{x}_j \text{ for all } j \in J \\
-\infty & \text{otherwise}
\end{cases} \\
\end{align*}
\]

where \( J \) is the index set for the \( x_j \)s. 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 \( j \in J \) 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 \( j \in J \).

**LBBB 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 \( \delta_{ij} \), where \( \delta_{ij} \) is 1 when task \( j \) is assigned to resource \( i \). The master problem (5) is

\[
\min \sum_{ij} c_{ij}\delta_{ij} \\
\sum_i \delta_{ij} = 1, \quad j = 1, \ldots, n
\]

Benders cuts

Subproblem relaxation

\( \delta_{ij} \in \{0, 1\}, \text{ all } i, j \)

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

\[
L_j \leq s_j \leq U_j - p_{ij}, \quad \text{all } j \in J_i \\
\text{cumulative}((s_j, j \in J_i), (p_{ij}, j \in J_i), (r_{ij}, j \in J_i), R_i)
\]

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

\[
\sum_{j \in J_i} (1 - \delta_{ij}) \geq 1
\]

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, Pape, and Nuijten 2001), or edge finding plus branching, one could observe the set \( J_i \) 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, 2005a; 2007c).

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 latest deadline:

\[
\sum_j p_{ij}\delta_{ij} \leq \max_j \{U_j\} - \min_j \{L_j\}
\]

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 (2005a; 2007c).

**Computational Results**

The problem instances are those used in Hooker (2004; 2005a; 2007c) 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 to 32 tasks on 2 to 4 resources, and “e” instances, which schedule 10 to 50 tasks on 2 to 10 resources.

The “c” instances are designed to be difficult for LBBB, 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 LBBB, 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 LBBB 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 2005a; 2007c). 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. LBBB is clearly superior to MIP, running at least 1000 times faster...
The absence of a relaxation not only generates more efficient Benders cuts, but no subproblem relaxation in the master problem, LBBD has a relaxation and simple nogood cuts, and LBBD with a relaxation and strengthened cuts.

<table>
<thead>
<tr>
<th>Size</th>
<th>MIP (CPLEX)</th>
<th>LBBD: strong cuts only</th>
<th>LBBD: relax + weak cuts</th>
<th>LBBD: relax + strong cuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>m n</td>
<td>5 0.1</td>
<td>5 0.2</td>
<td>5 0.1</td>
<td>5 0.1</td>
</tr>
<tr>
<td>2 10</td>
<td>5 0.2</td>
<td>5 0.2</td>
<td>5 0.1</td>
<td>5 0.0</td>
</tr>
<tr>
<td>2 12</td>
<td>5 0.1</td>
<td>5 0.4</td>
<td>5 0.1</td>
<td>5 0.0</td>
</tr>
<tr>
<td>2 14</td>
<td>5 0.4</td>
<td>5 2.0</td>
<td>5 0.2</td>
<td>5 0.3</td>
</tr>
<tr>
<td>2 16</td>
<td>5 2.0</td>
<td>5 19</td>
<td>5 0.5</td>
<td>5 0.7</td>
</tr>
<tr>
<td>2 20</td>
<td>5 388</td>
<td>5 120</td>
<td>5 2.0</td>
<td>5 8.0</td>
</tr>
<tr>
<td>2 22</td>
<td>4 1899</td>
<td>4 1899</td>
<td>5 617</td>
<td>5 955</td>
</tr>
<tr>
<td>2 24</td>
<td>2 4346+ 1 6341+ 4 1495+ 4 1936+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 26</td>
<td>1 6362+ 0 - 5 327 4 1642+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 28</td>
<td>2 4384+ 0 - 5 1004 5 1133</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 30</td>
<td>- 0 - 0 - 2 5391+ 2 5761+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 32</td>
<td>1 5813+ 0 - 2 4325+ 2 4325+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 10</td>
<td>5 0.0</td>
<td>5 0.2</td>
<td>5 0.1</td>
<td>5 0.1</td>
</tr>
<tr>
<td>3 12</td>
<td>5 0.4</td>
<td>5 0.5</td>
<td>5 0.1</td>
<td></td>
</tr>
<tr>
<td>3 14</td>
<td>5 0.3</td>
<td>5 1.2</td>
<td>5 0.3</td>
<td>5 0.2</td>
</tr>
<tr>
<td>3 16</td>
<td>5 1.3</td>
<td>5 5.6</td>
<td>5 2.7</td>
<td>5 0.8</td>
</tr>
<tr>
<td>3 18</td>
<td>5 5.4</td>
<td>5 22</td>
<td>5 7.8</td>
<td>5 1.4</td>
</tr>
<tr>
<td>3 20</td>
<td>4 1712+ 5 30</td>
<td>5 1.2</td>
<td>5 0.5</td>
<td></td>
</tr>
<tr>
<td>3 22</td>
<td>3 3674+ 5 59</td>
<td>5 7.5</td>
<td>5 2.6</td>
<td></td>
</tr>
<tr>
<td>3 24</td>
<td>2 4411+ 4 1739+ 5 15</td>
<td>5 5.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 26</td>
<td>0 - 4 3510+ 5 191</td>
<td>5 98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 28</td>
<td>2 5238+ 2 6645+ 5 270</td>
<td>5 209</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 30</td>
<td>- 0 - 0 - 2 4354+ 4 1856+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 32</td>
<td>0 - 0 - 2 4667+ 2 4751+</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

+ Computation terminated after 7200 sec for instances not solved to optimality.

The absence of a relaxation not only generates more efficient Benders cuts, but no subproblem relaxation in the master problem, LBBD has a relaxation and simple nogood cuts, and LBBD with a relaxation and strengthened cuts.

The superiority of LBBD is even clearer for the “e” instances, as revealed in Table 2 and the performance profile in Fig. 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 2 resources, and it brings rather limited improvement in the overall performance profile. Even when the instances with 2 resources are removed from the profile, as in Fig. 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 Fig. 2.

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.

on the larger instances with more than 2 resources. The advantage is less pronounced when there are only 2 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 Fig. 1.

The superiority of LBBD is even clearer for the “e” instances, as revealed in Table 2 and the performance profile...
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 in the "e" instances with only 2 or 3 resources, for which the master solution time tends to fail. Once the number of tasks reaches a certain point, solution of the master bogs down, and the method tends to fail. The results also reveal why the "c" instances are harder for LBBD because the assignment problem is relatively easy when there are only 2 or 3 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 per iteration, because the subproblem is solved repeatedly with only 2 or 3 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.

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 zero subproblem solution time, while strong cuts roughly equalize the master and subproblem time in most cases. LBBD is somewhat less efficient for instances with 8 and 9 resources, for which the master solution time reaches 7200 sec.

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.

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>Iters Master Subpr</th>
<th>Iters Master Subpr</th>
<th>Iters Master Subpr</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>18</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>0.1</td>
<td>0.1</td>
<td>5.0</td>
</tr>
<tr>
<td>14</td>
<td>19</td>
<td>0.1</td>
<td>0.3</td>
<td>1.8</td>
</tr>
<tr>
<td>16</td>
<td>41</td>
<td>0.5</td>
<td>1.5</td>
<td>2.0</td>
</tr>
<tr>
<td>18</td>
<td>149</td>
<td>5.7</td>
<td>14.2</td>
<td>2.4</td>
</tr>
<tr>
<td>20</td>
<td>107</td>
<td>3.5</td>
<td>11.7</td>
<td>3.6</td>
</tr>
<tr>
<td>22</td>
<td>340</td>
<td>70+</td>
<td>1782+</td>
<td>4.6</td>
</tr>
<tr>
<td>24</td>
<td>327+</td>
<td>67+</td>
<td>6263+</td>
<td>2.0+</td>
</tr>
<tr>
<td>26</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.8</td>
</tr>
<tr>
<td>28</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2.0</td>
</tr>
<tr>
<td>30</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>4.2+</td>
</tr>
<tr>
<td>32</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.2+</td>
</tr>
</tbody>
</table>

+Computation terminated for unsolved instances after the total computation time reaches 7200 sec.

Table 4: Computational analysis of logic-based Benders decomposition for "e" instances.

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>Iters Master Subpr</th>
<th>Iters Master Subpr</th>
<th>Iters Master Subpr</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>9.4</td>
<td>0.1</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>13</td>
<td>0.3</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>14</td>
<td>0.4</td>
<td>0.0</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>55</td>
<td>14.0</td>
<td>0.0</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>19</td>
<td>0.4</td>
<td>0.0</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>26</td>
<td>1.1</td>
<td>0.0</td>
</tr>
<tr>
<td>7</td>
<td>35</td>
<td>76</td>
<td>34.0</td>
<td>0.0</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>107+</td>
<td>1525+</td>
<td>0.0+</td>
</tr>
<tr>
<td>9</td>
<td>45</td>
<td>132</td>
<td>1048</td>
<td>0.0</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>39</td>
<td>43</td>
<td>0.0</td>
</tr>
</tbody>
</table>

+Computation terminated for unsolved instances after the total computation time reaches 7200 sec.
Conclusions

Based on the experiments described above, we conclude that logic-based Benders decomposition 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 mixed integer programming on the problem class studied here. This is despite the marked improvement of MIP solvers, already highly advanced, over the last few 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 decompose the scheduling problem itself using LBBD. This can be done by dividing the time horizon into segments and creating a subproblem for each segment. This approach has been successfully applied in the context of single-resource scheduling (Çoban and 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 and 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.

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 reaperately re-solving the subproblem to tease out dual information, but cuts based on the actual dual solution could be much more effective.

References


Figure 3: Performance profile for the 120 “c” instances with 3 and 4 resources.


Hooker, J. N. 2007b. Planning and scheduling by


