Checking-up on Branch-and-Check

J. Christopher Beck

Department of Mechanical & Industrial Engineering
University of Toronto
Toronto, Ontario M5S 3G8, Canada
jcb@mie.utoronto.ca

Abstract. Branch-and-Check, introduced ten years ago, is a generalization of logic-based Benders decomposition. The key extension is to solve the Benders sub-problems at each feasible solution of the master problem rather than only at an optimal solution. We perform the first systematic empirical comparison of logic-based Benders decomposition and branch-and-check. On four problem types the results indicate that either Benders or branch-and-check may perform best, depending on the relative difficulty of solving the master problem and the sub-problems. We identify a characteristic of the logic-based Benders decomposition runs, the proportion of run-time spent solving the master problem, that is valuable in predicting the performance of branch-and-check. We also introduce a variation of branch-and-check to address difficult sub-problems. Empirical results show that this variation leads to more robust performance than both logic-based Benders decomposition and branch-and-check on the problems investigated.

1 Introduction

Logic-based Benders decomposition (LBBD) [1, 2] has been proposed as a framework for hybrid techniques that combine mixed-integer programming (MIP) and constraint programming (CP). Informally, LBBD requires the decomposition of a problem into a master problem and a set of sub-problems. The solution approach solves the master to optimality, solves the sub-problems, and then adds constraints ("Benders cuts") to the master problem based on the sub-problem results. The approach iterates between solving the master problem and the sub-problems until a globally optimal solution is found and proved.

Branch-and-Check (B&C) [1, 3] is a generalization of LBBD where the sub-problems are solved *during* the search for a solution to the master problem. In Thorsteinsson's formulation [3], the sub-problems are solved every time a feasible master solution is found, and the cuts are added to the master problem, if necessary. B&C, therefore, is essentially a branch-and-cut search with the Benders sub-problems being the source of the cuts.

Despite the increasing interest in LBBD and speculation (e.g., [4]) that B&C could result in a significant improvement over LBBD, there does not appear to have been a systematic evaluation of B&C. We do not have a clear picture

of how B&C performs on different problems nor what problem characteristics contribute to its behavior. The contributions of this paper, therefore, are:

1. The first systematic empirical comparison of logic-based Benders decomposition and branch-and-check.
2. The identification of a characteristic of the behavior of LBBD that appears to be correlated to whether an improvement in performance can be expected from B&C.
3. The introduction and evaluation of a variation on B&C that addresses the poor performance of B&C when the sub-problems are difficult.

The following section provides the necessary background. We then turn to the empirical investigations which result in the first two contributions of this paper. Section 4 proposes and evaluates the variation of B&C while Section 5 discusses our results and concludes.

2 Background

In this section, logic-based Benders decomposition and branch-and-check are defined, the literature on branch-and-check is reviewed in detail, and the formal definitions of the problems studied in this paper are presented.

2.1 Logic-based Benders Decomposition and Branch-and-Check

Logic-based Benders decomposition [2] is a generalization of classical Benders decomposition that is based on the division of a problem into a master problem (MP) and a set of sub-problems (SPs). The MP is a projection of the global model to a subset of decision variables, denoted y, and the constraints and objective function components that involve only y. The rest of the decision variables, x, define the sub-problems. Solving a problem by Benders decomposition involves iteratively solving the MP to optimality and using the solution to fix the y variables, generating the sub-problems. The duals of the SPs are solved to find the tightest bound on the (global) cost function that can be derived from the original constraints and the current MP solution. If this bound is less than or equal to the current MP solution (assuming a minimization problem), then the MP solution and the SP solutions constitute a globally optimal solution. Otherwise, a constraint, called a "Benders cut," is added to the MP to express the violated bound and another iteration is performed.

Branch-and-check (B&C) [3] moves the SP solving into a branch-and-cut search to solve the MP. Rather than waiting until the MP is solved to optimality, the B&C algorithm solves the SPs at each feasible MP solution, generates the Benders cuts, and immediately adds them to the branch-and-cut tree. The current feasible MP solution is therefore rejected and search continues. Because only globally feasible MP solutions are accepted, the MP is solved only once: there are no MP-SP iterations as in LBBD.

Intuitively, B&C may out-perform LBBD because the MP is not repeatedly solved from scratch. Furthermore, a cut introduced based on one sub-optimal MP solution may cut-off other sub-optimal MP solutions whereas in LBBD these sub-optimal solutions may be enumerated in each iteration.

2.2 Literature Review

Thorsteinsson [3] introduced the name *branch-and-check* and performed experiments on a planning and scheduling problem. A significant speed-up was shown when compared to an LBBD model due to Jain & Grossmann [5]. These results were attributed to the claim that the MP was hard to solve, relative to the SPs, and therefore adding the SP cuts based on feasible MP solutions (rather than waiting for optimality) sped-up the overall solving process. A significant weakness of the work, however, is that the B&C implementation did not actually solve the SPs during the MP branch-and-cut search: each time the SPs were solved, the MP search was *restarted*.

Bockmayr & Pisaruk [6] adopt an approach very similar to B&C, except that cuts are added at each node in the MP branch-and-cut tree rather than only at integer feasible nodes.[1] The relaxed solution at each node is used to derive bounds that are then used to define the SPs. Computational results show that solving the SPs more often results in an improvement over the results of Jain & Grossmann. No comparison is done against Thorsteinsson's approach.

Sadykov & Wolsey [7] address the same scheduling problem as the above authors with a B&C approach. They state that solving the SPs at integral MP nodes only is an important feature distinguishing their algorithm from Bockmayr & Pisaruk. However, Sadykov & Wolsey use a tighter MP formulation than Bockmayr & Pisaruk and so it is not possible to attribute their improved performance solely to the more frequent solution of the SPs. In fact, Sadykov & Wolsey state that they believe that the main reason for their performance is the tighter MP model. Follow-up work [8] solves a one-machine minimum weighted number of late activities problem using B&C: the SP is solved at each feasible MP solution and the MP branch-and-cut search continues with the added Benders cuts. However, no LBBD algorithm is used for comparison.

In summary, we have been able to find only four papers [3, 6–8] that have implemented a B&C-like approach. Thorsteinsson and Bockmayr & Pisaruk use the same model as Jain & Grossmann and so these three papers vary only on the frequency with which the sub-problems are solved. Jain & Grossmann solve the sub-problems the least, only when an optimal MP solution is found. Thorsteinsson solves the SP at each feasible MP solution, restarting the MP search each time. Bockmayr & Pisaruk solve the SPs most often, at each node in the MP tree. Both Thorsteinsson and Bockmayr & Pisaruk show better performance than Jain & Grossmann but their approaches have not been compared to each other. The fourth work, Sadykov & Wolsey, also solves the SPs at each feasible

[1] The idea of solving the sub-problems *more often* than at each feasible MP solution appears in Hooker's original formulation [1].

MP solution and shows stronger results than Bockmayr & Pisaruk but uses a tighter MP formulation.

It appears, therefore, that the most we can conclude from previous work is that for a specific scheduling problem, solving the SPs more frequently than is done in LBBD leads to improved performance. We have not been able to find any work that directly compares B&C (without restarting the MP search) to LBBD. Moreover, work on B&C has been restricted to a single problem type, limiting the generality of the resulting conclusions.

2.3 Problems and Models

In this section, we define the four problems used in this study. Each of these problems has an existing LBBD model in the literature. We study three planning and scheduling problems and one location-allocation problem.

CostMinUnary CostMinUnary is the problem studied by Jain & Grossmann, Thorsteinsson, and Bockmayr & Pisaruk. The problem is defined by a set of jobs, $j \in J$, each with an individual release date, R_j, and deadline, S_j, which must be scheduled on a set of resources, I. A job can be assigned to any resource; however, its processing time, p_{ij}, and cost, f_{ij}, depend on the resource, $i \in I$, to which it is assigned. The objective is to assign the jobs to resources so that they can execute within their time-windows $[R_j, S_j]$, no jobs on the same resource overlap, and the cost of the resource assignment is minimized.

Following existing work [1, 5, 4], the master problem in an LBBD model can be defined as follows, with y_{ij} being a 0-1 variable expressing whether job j is assigned to resource i:

$$\text{minimize} \sum_{ij} f_{ij} y_{ij} \tag{1}$$

$$\text{s.t.} \sum_{i} y_{ij} = 1 \qquad \text{all } j \tag{2}$$

$$\sum_{j} p_{ij} y_{ij} \leq \max_{j}\{S_j\} - \min_{j}\{R_j\} \quad \text{all } i \tag{3}$$

$$\sum_{j \in J_{hi}} (1 - y_{ij}) \geq 1 \qquad \text{all } i, h = 1, \ldots H-1 \tag{4}$$

$$y_{ij} \in \{0, 1\} \qquad \text{all } i, j$$

The objective function (1) minimizes the cost of assigning jobs to resources, subject to the constraint that all jobs must be assigned to exactly one resource (2). Constraint (3) is a relaxation of the sub-problem expressing that the sum of the durations of the jobs assigned to any resource must be less than or equal to the time between the minimum release date and maximum due date. Constraints (4) are the Benders cuts, where J_{hi} is the set of jobs assigned to resource i in iteration h and that led to an infeasibility in the sub-problem. The cut simply expresses that, in order to form a feasible schedule, at least one job in J_{hi} must be assigned to a different resource.

The sub-problems are then straightforward to define using constraint programming: they are single-machine scheduling problems with release dates and due dates where the goal is to find a feasible schedule. Explicitly, if t_j is the start time of job j, the sub-problem for resource i for all j with $y_{ij} = 1$ is:

$$t_j \geq R_j \quad (5)$$
$$t_j + p_{ij} \leq S_j \quad (6)$$
$$\textbf{cumulative}(t_j, p_{ij}, \mathbf{1}, 1) \quad (7)$$

The global constraint **cumulative** [9] represents a single-machine scheduling problem to assign values to all start times, t_j, taking into account the durations of each job and the capacities. The capacity required by each job is represented in the vector $\mathbf{1}$ and the capacity of the resource is 1.

CostMinMulti The CostMinMulti problem is the same as CostMinUnary except that the resources are no longer unary and each job may require more than one unit of the resource. The model [4] has the objective function (1), constraint (2), and the Benders cuts (4) as in CostMinUnary. The sub-problem relaxation (3) is different to account for the discrete resource capacity and the fact that all the problems solved here have a release date of 0 and the same due date, represented by d_0. Letting C_i be the capacity of resource i and c_{ij} the amount of resource i required by job j during its processing time, the sub-problem relaxation expresses that the area of resource availability (i.e., capacity multiplied by the time horizon: $C_i \times (d_0 - 0)$) must be greater than or equal to the sum of the areas of the jobs assigned to i ($p_{ij}c_{ij}y_{ij}$). Therefore, constraint (8) replaces constraint (3).

$$\frac{1}{C_i} \sum_j p_{ij} c_{ij} y_{ij} \leq d_0, \text{ all } i \quad (8)$$

The sub-problem formulation is also changed to reflect the discrete capacity. Thus, constraint (7) becomes:

$$\textbf{cumulative}(t_j, p_{ij}, c_{ij}, C_i) \quad (9)$$

MkspMinMulti MkspMinMulti is a multi-capacity planning and scheduling problem with the objective of makespan minimization. In Hooker's model [4], M represents the makespan and the master problem is defined as follows:

$$\text{minimize } M \quad (10)$$
$$\text{s.t. } \sum_i y_{ij} = 1 \qquad \text{all } j \quad (11)$$
$$M \geq \frac{1}{C_i} \sum_j p_{ij} c_{ij} y_{ij} \qquad \text{all } i \quad (12)$$
$$M \geq M^*_{hi} - \sum_{j \in J_{hi}} (1 - y_{ij}) p_{ij} \quad \text{all } i, h = 1, \ldots H - 1 \quad (13)$$
$$y_{ij} \in \{0, 1\} \qquad \text{all } i, j$$

The differences from CostMinMulti are the sub-problem relaxation (12) and the Benders cut (13). The sub-problem relaxation is, in fact, a restatement of constraint (8) with a variable end of horizon, M, rather than the fixed one, d_0, and as such is based on exactly the same reasoning. The Benders cut makes use of M^*_{hi}, the minimum makespan on resource i in iteration h. The expression that is subtracted from M^*_{hi} relies on the fact that the maximum reduction in makespan that can come from removing job j from resource i (i.e., by setting y_{ij} to 0) is the duration of that job, p_{ij}.

Unlike the other two scheduling problems, in MkspMinMulti, the sub-problem is an optimization problem as follows:

$$\begin{aligned} \text{minimize } & M_i && (14)\\ \text{s.t. } & M_i \geq t_j + p_{ij} && (15)\\ & t_j \geq 0 && (16)\\ & \textbf{cumulative}(t_j, p_{ij}, c_{ij}, C_i) && (17) \end{aligned}$$

LocAlloc The LocAlloc problem is a facility location, customer allocation, and truck allocation problem. Given the set J of potential sites and the set I of clients, the goal is to choose which sites to open, to assign each customer to a single open site, to assign a number of trucks to each site, and to assign each customer to a single truck. Multiple customers can be assigned to the same truck provided the sum of their travel distances is less than a given maximum distance for the truck. For each site j there is an opening cost, f_j, and a capacity, b_j. The demand of the clients, d_i, assigned to a site must be less than or equal to the site capacity. Each vehicle has a fixed utilization cost, u, and a maximum total driving distance, ℓ. Serving client i from site j generates a driving distance, t_{ij}, for the vehicle performing the service and has an associated cost, c_{ij}. The available vehicles at a site are indexed in set K with parameter $\overline{k} \geq |K|$ being the maximum number of vehicles at any site.

In the LBBD model presented by Fazel-Zarandi & Beck [10], the master problem determines the open sites, the assignment of customers to sites, and the number of trucks at each site. The sub-problems are then separate feasibility problems which attempt to assign the customers to trucks.

The master problem decision variables are:

$$p_j = \begin{cases} 1, \text{ if site } j \text{ is opened} \\ 0, \text{ otherwise} \end{cases}$$

$$x_{ij} = \begin{cases} 1, \text{ if client } i \text{ is served by site } j \\ 0, \text{ otherwise} \end{cases}$$

$numVeh_j$ = number of vehicles assigned to facility j

The master problem can then be modeled as:

$$\text{minimize} \sum_{j \in J} f_j p_j + \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} + u \sum_{j \in J} numVeh_j \tag{18}$$

$$\text{s.t.} \sum_{j \in J} x_{ij} = 1 \qquad i \in I \tag{19}$$

$$\sum_{i \in I} t_{ij} x_{ij} \leq \ell \cdot \overline{k} \qquad j \in J \tag{20}$$

$$t_{ij} x_{ij} \leq \ell \qquad i \in I, j \in J \tag{21}$$

$$\sum_{i \in I} d_i x_{ij} \leq b_j p_j \qquad j \in J \tag{22}$$

$$numVeh_j \geq \left\lceil \frac{\sum_{i \in I} t_{ij} x_{ij}}{\ell} \right\rceil \qquad j \in J \tag{23}$$

$$numVeh_j \geq numVeh^*_{jh} - \sum_{i \in I_{jh}} (1 - x_{ij}) \qquad j \in J_h \tag{24}$$

$$x_{ij} \leq p_j \qquad i \in I, j \in J \tag{25}$$

$$x_{ij}, p_j \in \{0, 1\} \qquad i \in I, j \in J \tag{26}$$

The objective (18) is to minimize the total cost of opening facilities, serving customers from a site, and allocating vehicles to a site. Constraint (19) ensures that all clients are served by exactly one facility. The distance limitations are defined by constraints (20) and (21). Constraint (22) limits the demand assigned to facility j. Constraint (23) defines the minimum number of vehicles assigned to each site.

Constraint (24) is the Benders cut. Inspired by the makespan cut in the MkspMinMulti problem, this cut makes use of the optimal number of trucks at facility j in iteration h, $numVeh^*_{jh}$, and subtracts from it an upper-bound on the reduction in the number of trucks that can arise from removing one customer.

The sub-problem is a feasibility problem to determine if the customers can be feasibly assigned to the allocated trucks. In order to generate a cut, however, we must solve the optimization version of the problem, finding the minimum number of vehicles that the assigned clients can be packed into. This is a bin-packing problem that can be modeled in CP as follows:

$$\min numVehBinPacking_j$$

$$\text{s.t. } \mathbf{pack}(load_k, truck_i, dist_i) \tag{27}$$

$$numVeh_j \leq numVehBinPacking_j < numVehFFD_j \tag{28}$$

The variables of the sub-problem are $load_k$, the total travel distance for truck k based on its assigned clients and $truck_i$, the index of the truck assigned to client i. The distances between site j and client i are represented in the data vector $dist_i$. The **pack** global constraint (27) maintains the load of the vehicles given the distances and assignments of clients to vehicles [11]. The upper and lower bounds on the number of vehicles are represented by constraint (28). These bounds are derived from the MP solution ($numVeh_j$) and heuristic preprocessing ($numVehFFD_j$).

3 A Systematic Evaluation of Branch-and-Check

The next sub-section describes the problem instances for each of our problems as well as providing the experimental details. We then compare logic-based Benders decomposition and branch-and-check experimentally and present insights into the performance comparison through a deeper analysis of the results.

3.1 Experimental Setup

We use existing problem instances in all of our experiments. For the two multi-capacity scheduling problems we use Hooker's instances [4]: 75 instances with 2 resources, 60 instances with 3 resources, and 60 instances with 4 resources. The number of jobs varies between 10 and 38. The unary capacity problem instances are generated by modifying the multi-capacity instances by setting the capacity equal to 1 and modifying the time windows of each activity. The overall scheduling horizon is extended by a factor of 3.6, a value chosen after experimentation in order to guarantee that all instances have a feasible solution. The horizon change resulted in two other changes: as in Hooker's work the possible window for an activity is set to one-third of the (now extended) horizon and, unlike Hooker, the release of each job was drawn with uniform probability from the first two-thirds of the horizon. All other parameters (cost, processing times, etc.) are exactly as in Hooker's instances.

For the Location-Allocation problems, 300 instances are taken from Fazel-Zarandi & Beck [10]. In half of these instances, the cost of serving a customer from a specific location is correlated with the distance to the location, while in the remaining half, distance and cost are uncorrelated. The problem sizes (i.e., number of possible facilities $\times$ number of clients) are: $\{20 \times 10, 30 \times 15, 40 \times 20\}$.

All experiments were run with a 7200-second time limit on a Duo Core AMD 720 CPU with 1 MB cache, 4 GB of memory, running Red Hat Enterprise 4. The MIP solver is CPLEX 12.1 and the CP solver is ILOG Solver/Scheduler 6.7.

3.2 Logic-based Benders Decomposition vs. Branch-and-Check

The comparison of logic-based Benders decomposition with branch-and-check is shown in Table 1.[2] For each problem set, we present the mean and median difference in CPU time (LBBD minus B&C). This formulation means that positive numbers favor B&C (i.e., it has a lower mean CPU time) and negative entries favor LBBD. Using a bootstrap paired-t test [12], we also indicate the statistical significance at $p \leq 0.005$.

Our results are consistent with previous work on the CostMinUnary problems: B&C shows a clear benefit, especially with an increased number of resources. However, the advantage for B&C disappears for the other scheduling problems to the point that LBBD shows significantly lower mean run-time overall and on three of the six subsets of CostMinMulti and MkspMinMulti. Finally, for LocAlloc, B&C again shows a significant advantage over LBBD.

[2] The OPT15 columns are discussed in Section 4.1.

		B&C		OPT15	
Problem	Set	Mean	Median	Mean	Median
CostMinUnary	2	*110.9	0.1	*110.9	0.1
	3	*164.8	1.2	*200.2	1.2
	4	*1049.7	19.7	*1038.7	19.7
	all	*416.3	0.7	*423.8	0.7
CostMinMulti	2	†-206.4	0	†-207.3	0
	3	-194.8	0.1	†-224.6	0.1
	4	-15.1	0.9	-8.6	0.9
	all	†-144.0	0	†-151.5	0
MkspMinMulti	2	-106.8	0	-58.7	0
	3	†-361.7	0	-215.2	0
	4	†-804.3	-0.1	-163.9	0.2
	all	†-400.0	0	-139.2	0
LocAlloc	cor	*999.4	66.8	*948.1	12.9
	uncor	*812.5	11.4	*848.5	11.7
	all	*905.9	23.9	*898.3	12.4

Table 1. Summary of B&C and OPT15 Performance. Mean and Median are the corresponding average differences in run-time (in seconds) between B&C and LBBD and between OPT15 and LBBD. A negative value indicates that LBBD achieves a lower run-time. The symbols * and † indicate a significant difference in mean run-time at $p \leq 0.005$, for the corresponding B&C variation and LBBD, respectively.

3.3 A Deeper Analysis

Table 2 presents further data: the number of iterations and the percentage of the run-time spent on the master problem and the sub-problems.

The statistics for LBBD indicate that it has significantly different behavior on the four problems. Using the median, we see that in CostMinUnary, LBBD spends over 97% of the run-time solving the MP and does 23 iterations. This is a substantial difference from the other two scheduling problems: a median 11% and 21% of their run-times is spent on the MP and they perform a median of 7 and 13 iterations, respectively. LocAlloc is different again, spending 100% of the run-time on the master problem but only requiring two iterations.

The branch-and-check results show a large increase in both the number of times that the SPs are solved and a corresponding increase in the proportion of CPU time spent solving them. This pattern is not seen for LocAlloc as, though there is a substantial increase in the number of SP iterations, most of the run-time is still spent solving the master problem.

The positive differences in CPU time in Table 1 correspond to problem sets where a significant portion of the run-time is spent on the master problems. Figure 1 plots the mean difference in run-time between LBBD and B&C against the proportion of time spent solving the MP by LBBD. We have aggregated the latter data into 10 buckets corresponding to intervals of size 0.1. The pattern that can be observed is that unless LBBD spends about 80% or more of its time solving the master problem, the benefits from branch-and-check are rare. In

		LBBD			B&C		
Problem	Set	Iterations	% MP	% SP	SP Iterations	% MP	% SP
CostMinUnary	2	62.8 (6)	75.9 (91.5)	22.8 (7.9)	165.4 (11)	32.0 (28.0)	57.3 (64.5)
	3	138.9 (24)	90.4 (95.1)	9.6 (4.9)	1047.5 (40)	44.1 (42.7)	54.2 (57.1)
	4	258.2 (81.5)	96.2 (99.3)	3.8 (0.7)	1022.4 (160)	53.5 (50.8)	46.5 (49.2)
	all	146.3 (23)	86.6 (97.0)	12.9 (2.8)	700.5 (50)	42.4 (42.5)	53.0 (55.6)
CostMinMulti	2	2.6 (1)	12.8 (0)	81.9 (100)	4.1 (2)	8.1 (0)	90.5 (100)
	3	23.6 (14.5)	51.3 (66.5)	48.7 (33.5)	47.2 (26.5)	15.9 (3.7)	84.1 (96.2)
	4	35.2 (22)	68.8 (93.4)	31.2 (6.6)	69.4 (45.5)	22.7 (18.1)	77.3 (81.9)
	all	19.1 (7)	41.9 (11.2)	56.1 (73.5)	37.4 (17)	15.0 (0.5)	84.5 (99.4)
MkspMinMulti	2	18.9 (5)	9.0 (0)	91.0 (100)	20.8 (6)	3.7 (0)	96.3 (100)
	3	59.0 (25.5)	37.4 (33.3)	62.6 (66.7)	69.9 (28.5)	13.0 (1.2)	87.0 (98.8)
	4	51.3 (20)	55.5 (60.0)	44.5 (40.0)	70.2 (36)	17.4 (8.7)	82.6 (91.4)
	all	41.2 (13)	32.0 (21.5)	68.0 (78.5)	51.1 (20)	10.8 (0.3)	89.2 (99.7)
LocAlloc	cor	7.2 (1.5)	99.9 (100)	0.1 (0)	70.6 (19)	99.4 (100)	0.6 (0)
	uncor	5.9 (2)	99.9 (100)	0.1 (0)	87.2 (23.5)	98.4 (100)	1.6 (0)
	all	6.5 (2)	99.9 (100)	0.1 (0)	78.9 (21.5)	98.9 (100)	1.1 (0)

Table 2. Details of the logic-based Benders decomposition vs. Branch-and-Check experiment. On the left-hand side, we present the mean (and median) number of master problem iterations and the percentage of time spent solving the master problem (% MP) and the sub-problems (% SP). On the right-hand side, branch-and-check data is presented: the number of iterations of the sub-problem (i.e., the number of times that the *set* of sub-problems is solved–recall that the master problem is solved only once), as well as the percentages of the run-time spent on the master problem and sub-problems.

contrast, with master run-time proportions approaching 1, both the magnitude and the frequency of benefits from using B&C are much higher.

These results can be understood by noting that LBBD and B&C embody different expectations with respect to relative sub-problem difficulty. In LBBD, the SPs are solved once for every (optimal) MP solution. In B&C, the SPs are solved at each feasible solution to the MP. If the MP is much harder to solve than the SPs, solving the MP once and using the cuts that are generated inexpensively from repeated SP solutions should result in lower overall run-time. In contrast, if the SPs are not easily solved, then frequently solving them is counter-productive. It would be better to solve the SPs only when necessary: when an optimal master problem needs to be either confirmed or cut-off. This is precisely the link between the results in Tables 1 and 2.

The generality and analytical understanding of this pattern remain to be explored. However, we believe that, as a broad measure, the portion of run-time spent by LBBD on solving the master problem is a promising indicator of the benefit that can accrue from applying B&C. Minimally, it can be employed by practitioners when they are deciding if spending the time to implement B&C is likely to be worthwhile.

4 A Variation on Branch-and-Check

The experiments above indicate that the difficulty in solving sub-problems is important to the performance differences between LBBD and B&C. Additionally, we make two observations.

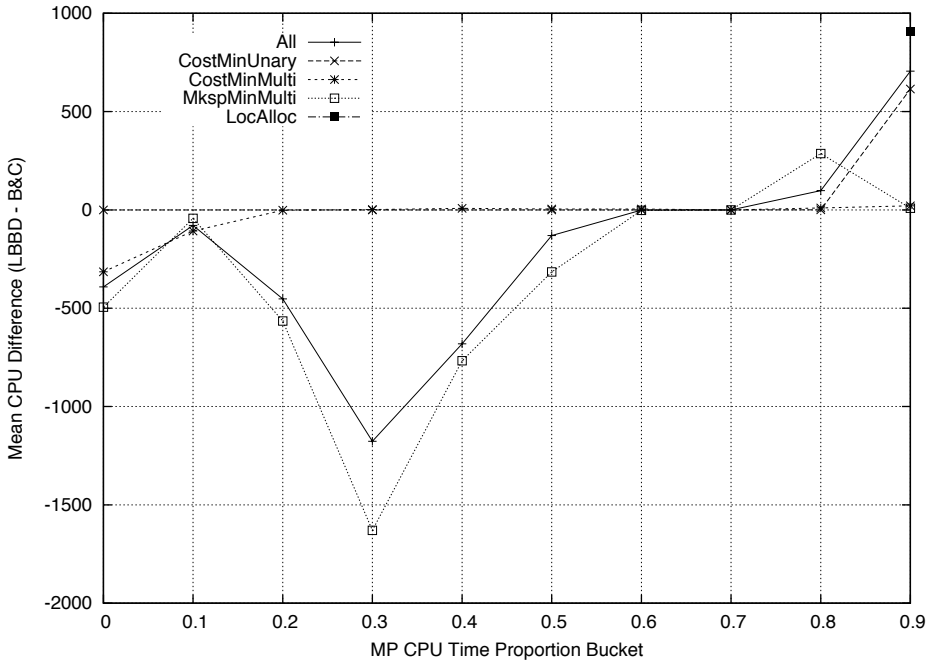

Fig. 1. A plot of the proportion of run-time spent solving the master problem, aggregated into 10 buckets ([0 0.1), [0.1 0.2) ... [0.9 1.0]), against the mean difference in CPU time (LBBD minus B&C). Note that the LocAlloc data all fall into the final bucket and therefore form only a single point.

1. A feasible MP solution may be very different from an optimal one. The cuts that are generated to remove the former may be irrelevant to cutting off optimal MP solutions that are not globally feasible and, therefore, the work of solving the SPs and generating cuts may be wasted.
2. The sub-problems in a given problem instance are not equally difficult. For example, on the scheduling problems we have observed poor but feasible MP solutions that place most or all activities on one machine, inducing the worst-case in terms of SP difficulty.

These observations suggest that the avoidance of difficult and irrelevant sub-problems may lead to better performance. Therefore, we propose a variation of B&C that solves the SPs more frequently than LBBD but less often than B&C by filtering the feasible MP solutions for which it solves the SPs. Our simple idea, denoted OPT15, is as follows: within the B&C algorithm, rather than solving SPs for each feasible MP solution, we solve the SPs corresponding to feasible

MP solutions with an optimality gap of less than 15%.[3] Feasible solutions with larger gaps are accepted as globally feasible.

The completeness of B&C is compromised by this change unless a feasible MP solution with a gap of less than 15% is subsequently found *and* proved to be globally feasible. If such a new MP solution is not found, we preserve completeness by running a second iteration of B&C without the 15% threshold. The second iteration has two significant advantages over the first iteration: all the cuts from solving the SPs in the first iteration are incorporated and the warm-start functionality of the MIP solver typically allows a good initial feasible MP solution to be found in the pre-solve phase.

The choice of 15% is arbitrary and based on examination of preliminary experiments. No tuning was done to investigate different choices for the threshold.

4.1 Experimental Evaluation

The right-hand side of Table 1 in Section 3.2 presents the mean and median run-time difference between LBBD and OPT15. The problem instances and experimental setup are the same as described in Section 3.1.

The empirical results indicate that OPT15 performs more robustly than LBBD or B&C. On the problems where B&C does significantly better than LBBD (CostMinUnary and LocAlloc), OPT15 performs about the same as B&C, achieving a statistically significant difference when compared to LBBD run-time and achieving equivalent performance in terms of mean run-time as B&C. The only statistic that is significantly different is the median run-time for LocAlloc, which is considerably smaller for OPT15. On problems where B&C performs poorly compared to LBBD, OPT15 performs approximately the same as B&C on CostMinMulti and much better on MkspMinMulti. In fact, there is no statistically significant difference between LBBD and OPT15 on the latter set.

A different perspective on this data is presented in Table 3. The table presents two pieces of data: the mean and median number of times that the set of SPs is solved (SP Iter.) and the percentage of problem instances for which the algorithm achieved a run-time within 10 seconds of the best run-time achieved by any algorithm. For example, on the two-machine instances of CostMinUnary, LBBD is within the 10 seconds of the best run-time on 85.3% of the problem instances, while B&C and OPT15 are within the threshold on all instances.

The SP iteration data demonstrate that, indeed, OPT15 tends to solve the sub-problems less frequently than B&C. The difference, however, is small.

The % Best data indicates that OPT15 is seldom best on a given subset: it is alone with the highest percentage on two subsets (CostMinMulti/4 and LocAlloc/uncor), while LBBD is uniquely the best on 5 sets and B&C on 2. However, it is never the worst performer while LBBD and B&C have poor results on different problem sets. Overall, the performance of OPT15 results in it being within the 10-second threshold on 84.7% the problems compared to 82.7% and 67.7% for B&C and LBBD, respectively.

[3] This gap is between the cost of the incumbent MP solution and the best current lower bound on the MP solution.

		LBBD		B&C		OPT15	
Problem	Set	% Best	SP Iter.	% Best	SP Iter.	% Best	SP Iter.
CostMinUnary	2	85.3	62.8 (6)	**100.0**	165.4 (11)	**100.0**	165.0 (11)
	3	75.0	138.9 (24)	**98.3**	1047.5 (40)	**98.3**	1008.3 (34)
	4	41.7	258.2 (81.5)	**96.7**	1022.4 (160)	91.7	1045.7 (139)
	all	68.7	146.3 (23)	**98.4**	700.5 (50)	96.9	695.4 (40)
CostMinMulti	2	**98.7**	2.6 (1)	88.0	4.1 (2)	88.0	4.1 (2)
	3	**83.3**	23.6 (14.5)	76.7	47.2 (26.5)	78.3	45.9 (26.5)
	4	80.0	35.2 (22)	81.7	69.4 (45.5)	**85.0**	74.5 (41.5)
	all	**88.2**	19.1 (7)	82.6	37.4 (17)	84.1	38.6 (16)
MkspMinMulti	2	**90.7**	18.9 (5)	85.3	20.8 (6)	86.7	19.5 (6)
	3	**85.0**	59.0 (25.5)	68.3	69.9 (28.5)	80.0	62.6 (26.5)
	4	**88.3**	51.3 (20)	66.7	70.2 (36)	85.0	71.6 (26)
	all	**88.2**	41.2 (13)	74.4	51.1 (20)	84.1	48.8 (17)
LocAlloc	cor	35.3	7.2 (1.5)	**82.0**	70.6 (19)	71.3	62.7 (15.5)
	uncor	45.3	5.9 (2)	74.0	87.2 (23.5)	**84.0**	69.3 (20)
	all	40.3	6.5 (2)	**78.0**	78.9 (21.5)	77.7	66.0 (18)
All		67.7	47.8 (6)	82.7	200.9 (23)	**84.7**	194.9 (19)

Table 3. The mean (and median) number of sub-problem iterations (SP Iter.) and percentage of problem instances (% Best) for which the algorithm's run-time was within 10 seconds of the best run-time. Bold entries indicate the highest percentage in a row.

5 Discussion and Conclusion

This paper has presented the first systematic comparison of logic-based Benders decomposition and branch-and-check. Using four different problems from the scheduling and facility location literature, we have demonstrated that B&C can lead to a significant improvement over LBBD but that the improvement is dependent on the difficulty of solving the sub-problems relative to that of solving the master problem. For problems where the sub-problem is difficult, B&C can result in significantly longer run-times than LBBD. We have also shown that the proportion of run-time used in LBBD to solve the master problem is a good measure of the likelihood of the benefit from implementing a B&C algorithm. Our results show that unless at least 80% of the LBBD run-time is spent on the master problem, benefits from B&C are small and rare.

The generality of these conclusions is still in question, as we have only evaluated four problem types, three of which are related scheduling problems. It would be interesting to perform similar experiments with radically different problems. Despite the problem similarities, however, three different behaviors were observed in terms of the proportion of CPU time spent solving the sub-problems and the number of master problem iterations (see Table 2). Furthermore, our results are consistent with our understanding of the increased emphasis on solving sub-problems that is embodied by B&C. We are, therefore, optimistic that the conclusions here will be confirmed in follow-up research.

In our experiments, the sub-problem relaxation and the Benders cut were not independent variables. These are two critical components of an LBBD-style

algorithm and changing these model components may change the relative performance of LBBD and B&C on a given problem. However, we conjecture that the fundamental conclusion regarding the proportion of effort in solving the master problem versus the sub-problems would still be valid. Specifically, a tighter, harder-to-compute Benders cut should result in fewer iterations but may result in much more expensive sub-problems. Depending on the strength and computational cost of the cut, the new model may spend either a higher or lower proportion of its run-time on the sub-problems. Our results suggest that if the new cut shifts the run-time toward the master problem then B&C has an improved likelihood of out-performing LBBD when compared with the weaker cut. The reverse is true if the new cut results in a larger proportional effort on the sub-problems. In contrast, a tighter, more expensive sub-problem relaxation should increase the effort required in solving the master problem while reducing the number of times that the sub-problems must be solved. This shift suggests that a tighter sub-problem relaxation would tend to favour B&C over LBBD.

This paper also introduces OPT15, a B&C variation that achieves more robust performance by avoiding sub-problems that are difficult and irrelevant to cutting off optimal master solutions. It achieves this goal by only solving sub-problems for master problem solutions with an optimality gap of 15% or less.

The relative performance of OPT15 and B&C depends on the quality of the feasible MP solutions that are found. Experiments using an earlier version of CPLEX (version 11.0) demonstrated significantly worse B&C performance and correspondingly larger OPT15 improvement on the CostMinMulti and MkspMinMulti problems. The performance change with CPLEX 12.1 was due to an improvement in the quality of the first feasible MP solution found. With CPLEX 11.0, the initial MP solutions often induced worst-case SPs that, by themselves, exhausted the 7200-second time limit. It would be interesting to repeat the above experiments with different CPLEX settings (e.g., preferring optimal to feasible solutions) and with other MIP solvers to further investigate the importance of the initial feasible MP solution. We expect the performance of OPT15 to increase when the initial feasible MP solutions are of poorer quality.

OPT15 investigates "middle ground" between solving sub-problems only for optimal MP solutions versus solving them for each feasible MP solution. As a relatively simple idea, it is unclear if OPT15 specifically deserves further development. However, as an example of a technique that interpolates between LBBD and B&C, it opens the possibility for more sophisticated approaches. Our choice of using a threshold on the optimality gap and the specific choice of that threshold were arbitrary. One might instead set a small time-limit on sub-problem searches. For easy sub-problems, the performance would be identical to B&C while for harder sub-problems, performance may approach that of LBBD. One could consider adaptively learning such time limits for given problems or problem instances.

Acknowledgments This research was supported in part by the Natural Sciences and Engineering Research Council of Canada, Canadian Foundation for Innova-

tion, Ontario Ministry for Research and Innovation, Alcatel-Lucent, and IBM. Thanks to Daria Terekhov for comments on earlier versions.

References

1. Hooker, J.N.: Logic-based Methods for Optimization. Wiley (2000)
2. Hooker, J., Ottosson, G.: Logic-based Benders decomposition. Mathematical Programming **96** (2003) 33–60
3. Thorsteinsson, E.S.: Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In: Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming (CP2001), Springer (2001) 16–30
4. Hooker, J.: A hybrid method for planning and scheduling. Constraints **10** (2005) 385–401
5. Jain, V., Grossmann, I.E.: Algorithms for hybrid MILP/CP models for a class of optimization problems. INFORMS Journal on Computing **13**(4) (2001) 258–276
6. Bockmayr, A., Pisaruk, N.: Detecting infeasibility and generating cuts for mixed integer programming using constraint programming. Computers & Operations Research **33** (2006) 2777–2786
7. Sadykov, R., Wolsey, L.A.: Integer programming and constraint programming in solving a multimachine assignment scheduling problem with deadlines and release dates. INFORMS Journal on Computing **18**(2) (2006) 209–217
8. Sadykov, R.: A branch-and-check algorithm for minimizing the weighted number of late jobs on a single machine with release dates. European Journal of Operational Research **189** (2008) 1284–1304
9. Baptiste, P., Le Pape, C., Nuijten, W.: Constraint-based Scheduling. Kluwer Academic Publishers (2001)
10. Fazel-Zarandi, M.M., Beck, J.C.: Solving a location-allocation problem with logic-based Benders decomposition. In: Proceedings of the Fifteenth International Conference on Principles and Practice of Constraint Programming (CP2009). (2009) 344–351
11. Shaw, P.: A constraint for bin packing. In: Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP04). (2004) 648–662
12. Cohen, P.R.: Empirical Methods for Artificial Intelligence. The MIT Press, Cambridge, Mass. (1995)