Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Automatic Logic-Based Benders Decomposition with MiniZinc

Toby O. Davies, Graeme Gange, Peter J. Stuckey
Department of Computing and Information Systems
The University of Melbourne, Victoria 3010, Australia

Abstract

Logic-based Benders decomposition (LBBD) is a powerful hybrid optimisation technique that can combine the strong dual bounds of mixed integer programming (MIP) with the combinatorial search strengths of constraint programming (CP). A major drawback of LBBD is that it is a far more involved process to implement an LBBD solution to a problem than the "model-and-run" approach provided by both CP and MIP. We propose an automated approach that accepts an arbitrary MiniZinc model and solves it using LBBD with no additional intervention on the part of the modeller. The design of this approach also reveals an interesting duality between LBBD and large neighborhood search (LNS). We compare our implementation of this approach to CP and MIP solvers on 4 different problem classes where LBBD has been applied before.

1 Introduction

Logic-based Benders decomposition is among the most effective approaches for finding optimal solutions to complex configuration and scheduling tasks, frequently two or three orders of magnitude faster than pure MIP or CP approaches (Hooker and Ottosson 2003).

The essence of logic-based benders decomposition is to take the problem P, and derive a *relaxed master* P_M (typically a MIP) which relaxes or omits some constraints in P, and a set of independent subproblems $P_1, \ldots, P_k$ such that $P \leftrightarrow P_M \wedge P_1 \wedge \cdots \wedge P_k$. The master solves P_M and the solution μ is checked by each subproblem solver for P_i. If μ does not satisfy P_i a cut is added to the master to eliminate μ and othe solutions which will not satisfy P_i for the same reason.

However, designing a concrete instantiation of this framework is typically nontrivial. The first difficulty is in choosing the relaxation P_M. If the master omits important constraints entirely, the candidate solutions are too optimistic, and the method converges slowly. Conversely, if the constraint is not substantially relaxed, solving the master becomes unmanageable (effectively regressing to a pure MIP approach).

A second issue is the extraction of feasibility cuts from infeasible subproblems. Logic-based Benders cuts are typically couched in terms of the *inference dual* (Hooker and Ottosson 2003). However typical subproblem solvers (in particular, classical CP solvers) do not provide sufficient information to reconstruct a compact justification of failure, so in practice cuts are derived by exploiting the independence of the subproblems and knowledge of their structure (Ciré, Coban, and Hooker 2013).

In this paper, we present a fully automatic approach for solving constrained optimization problems via logic-based Benders decomposition. The automated decomposition diverges from typical instances of logic-based Benders decomposition in a key respect: it constructs only a single complete 'subproblem'. Rather than explicitly decomposing our problem into independent subproblems (either manually or heuristically), we instead rely on the conflict analysis capabilities of lazy clause generation solvers to identify relevant subsystems.

Though this approach makes the subproblem considerably more difficult, it offers several advantages. Unlike classical LBBD, it can cope with subsystems which are not fully disjoint. There is no need to design problem- or objective-specific cuts; cut derivation is entirely generic. And while generating cuts, the subproblem solver also acts as a primal heuristic.

This formulation reveals an interesting duality. From the perspective of the MIP solver, this is an instance of logic based Benders decomposition where the subproblem solver doubles as a primal heuristic. But from the perspective of the CP solver, our framework is an instance of large neighborhood search (LNS) (Pisinger and Ropke 2010) – the MIP proposes promising candidate regions, which the CP solver progressively explores and expands – in which the neighborhood selection heuristic also supplies valid lower bounds. We will refer to the search space implied by a set of assumptions as a neighborhood throughout this paper.

The contributions of this paper are as follows:

- A variant of logic-based Benders decomposition which reveals a duality between LBBD and large-neighborhood search (LNS)
- An automated approach for applying LBBD techniques to arbitrary constraint models
- A simple way of integrating traditional CP optimisation-as-repeated-satisfaction into the LBBD framework.

2 Preliminaries

2.1 Constraint Programming and Lazy Clause Generation

Constraint programming (CP) systems solve a problem of the form $\exists V.D \wedge C$, D is a conjunction of unary constraints (*a domain*) constraining each integer variable[1] $v \in V$ to take a finite set of values, and C is an arbitrary collection of constraints on variables V. A domain D that entails a single value for each variable in V is a *valuation* which we denote by μ. We use notation $\mu(v)$ to return the value of variable v given by valuation μ. Each constraint $c \in C$ is implemented using a propagator which given a domain D infers new unary constraints d which must hold, i.e. $D \wedge c \rightarrow d$. The domain is then updated to $D' = D \wedge d$. Failure is detected if $d = false$. Success is detected if D' is a valuation (under reasonable assumptions about the strength of propagator inferences). Otherwise when no further inference is possible, the system guesses a new unary constraint l, and recursively considers the two systems $\exists V.(D \wedge l) \wedge C$ and $\exists V.(D \wedge \neg l) \wedge C$.

A *lazy clause generation (LCG)* solver in addition tracks the reasons for its inferences. For each inference $D \wedge c \rightarrow d$ it stores a *reason* clause $d_1 \wedge \cdots \wedge d_n \rightarrow d$ which is a consequence of c, that explains the inference. When failure is detected it uses the reasons to construct a *nogood* by resolution which explains the failure. The nogood is then added to the solver to prevent the same failure re-occuring.

An LCG solver can be extended to support an *assumption interface*. Given a set of unary constraint assumptions μ the solver solves $\exists V.(D \wedge \mu) \wedge C$. If the solver determines the problem has no solution it can return a clause of the form $\mu_1 \wedge \cdots \wedge \mu_n \rightarrow false$ where $\mu_i \in \mu$, that explains which assumptions were responsible for the failure, i.e. such that $\exists V.(D \wedge \mu_1 \wedge \cdots \wedge \mu_n) \wedge C$ is unsatisfiable.

2.2 The MINIZINC solver pipeline

MINIZINC (Nethercote et al. 2007) is a high-level declarative modeling language for constrained optimization problems. Underlying solvers typically do not support MINIZINC directly. Instead, an instance of the high-level MINIZINC model is compiled down to a simpler FLATZINC. During this *flattening* step, existential quantification and complex Boolean structure are eliminated. Each solver provides a library of predicate definitions to control this flattening; global constraints are handled in one of three ways:

- If the solver provides a declaration but no definition, the constraint call is passed directly to the solver.
- If a definition for the predicate is provided, the definition is expanded (and recursively flattened).
- Otherwise, the default flattening is expanded.

This flattened model is then fed to the solver, paired with directives for formatting output. This architecture provides a uniform framework for allowing high-level model specifications while exploiting the heterogeneous capabilities of different solvers.

[1]Here we treat Boolean variables as 01 integers.

In Section 3, we shall demonstrate how to (mis-)use this framework to support a logic-based Benders approach.

2.3 Classical and Logic-based Benders Decompositions

Logic-based Benders decomposition (Hooker and Ottosson 2003) replaces the linear programming dual used in classical benders decomposition with the more general *inference dual* – the problem of inferring the tightest objective bound from a set of constraints, its solution being a proof of the optimal bound. This proof is then translated into a sound bounding function suitable for addition to the master – in the case of a MIP master problem, the optimality proof must be translated into one or more linear inequalities over variables occurring in the master. Unfortunately, most subproblem solvers cannot provide information to reconstruct the inference dual, supplying only the primal solution. In this case, it is common to instead design specialized cuts based on problem structure, identifying some subset E of assignments to shared variables such that $f(E) \leq z$. In contrast, we use a clausal cut of the form: $E \rightarrow false$ (or equivalently $E \rightarrow [\![z > k]\!]$ in place of optimality cuts), these cuts are explained in more detail in section 3.1.

2.4 Cut strengthening and MUS construction

Without explicit dual information, the generated Benders cuts are quite coarse. These may then be *strengthened*, using the subproblem solver as a feasibility oracle – progressively discarding assumptions, and checking that the (now relaxed) subproblem remains infeasible.

When the master assignment is viewed as a conjunction of propositions, this "cut strengthening" process corresponds exactly to minimal unsatisfiable subset (MUS) construction: given an unsatisfiable conjunction C of constraints, identify one or more minimal subsets $C' \subseteq C$ which preserve infeasibility.

MUS construction arises in various contexts, and has received particular attention in SAT and CP (Dershowitz, Hanna, and Nadel 2006a; Liffiton and Malik 2013; Junker 2004; Hemery et al. 2006). As in the LBBD case, SAT MUS construction algorithms use a decision procedure (here a SAT solver) as an oracle, repeatedly choosing $P \subset C$ and testing satisfiability of $C \setminus P$ until an MUS has been identified. These algorithms differ in their strategy for choosing subsets to test – sequential (Bakker et al. 1993), dichotomic (Felfernig, Schubert, and Zehentner 2012) and geometric progression (Marques-Silva, Janota, and Belov 2013) strategies have been proposed. One general refinement to these approaches, *clause-set refinement* (Dershowitz, Hanna, and Nadel 2006b), is directly applicable to the CP case.

Clause-set refinement exploits the capability of SAT solvers to return an unsatisfiable core. When a query $C \setminus P$ returns `UNSAT(`C'`)` (showing there is some MUS excluding P), MUS construction may simply replace $C \setminus P$ with C' before continuing.

Figure 1 illustrates a sequential approximate MUS algorithm, with clause-set refinement and resource limits.

```
MUS-seq(C, lim)
  M := ∅; U := ∅
  while(∃c ∈ C)
    C := C \ c
    case(is-sat(M ∪ U ∪ C, lim))
      [SAT]:
        M := M ∪ {c};
      [UNSAT(C')]:
        C := C ∩ C'
        U := U ∩ C'
      [UNKNOWN]:
        U := U ∪ {c}
  return M ∪ U
```

Figure 1: Pseudo-code for sequential MUS construction with conflict-set refinement and resource limits. M contains those propositions definitely in the MUS, and U those which could not be eliminated. We may terminate this procedure at any time, returning $M \cup U \cup C$ as an unsatisfiable core.

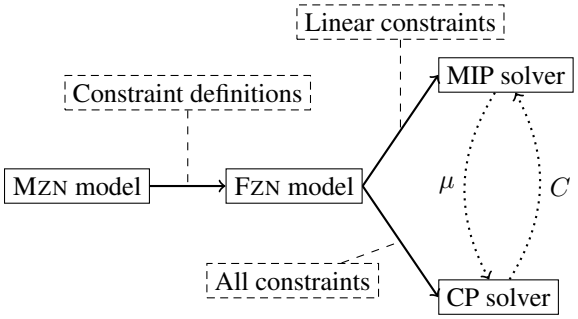

Figure 2: Architecture of the automatic Logic-based Benders Decomposition.

3 Automating Logic-based Benders Decomposition

The key idea behind this approach is quite simple, and is shown in Figure 2. During flattening, any constraints not supported by a solver are replaced by an equivalent (modulo introduced variables) solver-specific decomposition. If we discard any subset of these constraints, what remains is necessarily a valid relaxation.

The master problem M is constructed directly from the existing flattened model, retaining those constraints for which compact encodings exist – primarily ELEMENT and (reified) LINEAR constraints – and discarding the rest of the model.

Similarly, we do not attempt to identify independent subproblems, or partition variables between master and subproblem. The single "subproblem" S consists of the full flattened model, and the master and subproblem are solved over all variables appearing in any master constraint; and all variables respectively.

The high-level LBBD procedure is shown in Figure 3: we first solve the master M to optimality, then try to extend the partial solution μ to a full solution μ^* of the subproblem S. We then extract and minimize one or more cuts C' from the

```
solve-lbbd(obj, M, S):
  μ* := ⊥
  while(solve-master(M) = SAT(μ))
    S := S ∧ [[obj ≥ μ(obj)]]
    case(solve-assume(S, μ, μ*))
      [UNSAT(∅, μ*)]
        return(μ*)
      [UNSAT(C, μ*)]
        X := ∅
        while(solve-lim(S, μ \ X, μ*) = UNSAT(C, μ*))
          (C', μ*) := minimize-cut(S, C, μ*)
          M := M ∧ (C' → false)
          if(C' = ∅) return(μ*)
          M := M ∧ [[obj < μ*(obj)]]
          X := X ∪ C'
```

Figure 3: Pseudo-code for Logic-based Benders decomposition with a complete subproblem.

subproblem and re-solve the (now tighter) master. We use X to track literals that have appeared in any clause so far, and omit these from subsequent solves to guarantee cuts are independent. Note: μ, μ^*, C and C' are all sets of bounds literals, and μ and μ^* are valuations.

The subproblem solve *solve-assume*(S, μ, μ^*) takes the constraints S and assumptions μ and the current best solution μ^* and optimizes the objective. This means the subproblem never returns SAT, but instead always returns a (potentially new) μ^*, as whenever a new model μ^* is found, it is immediately invalidated by adding a new constraint to S: $[\![obj < \mu^*(obj)]\!]$. Consequently the termination condition is different to traditional LBBD: search terminates when the master fails (typically after being returned an empty cut).

This exploits CP optimisation: If a new incumbent is found, we may now tighten the objective bound in both the master and subproblem in the hope that the subproblem can utilize this bounds information to generate more succinct cuts. In particular, the subproblem can generate an empty cut as soon as it finds any model with an objective equal to the lower bound.

In classical LBBD, the master can be seen as incrementally building a MIP model representing a projection of the problem onto the master variables. By adding these objective-based cuts, and allowing the sub-problem to use them to simplify its cuts, we can instead view the master as building a projection of the subset of solutions strictly better than the incumbent. The **while** loop represents an approximation of looping over subproblems, described more fully in Section 3.3.

3.1 Deriving and encoding cuts

In the above formulation, we have not explicitly decomposed the CP model into disjoint subproblems; indeed, the subproblem is exactly the model that would be used to solve the problem with a direct CP approach.

Using a classical CP solver, this is not an ideal approach; the solver cannot readily pinpoint the unsatisfiable subproblem. However, conflict directed clause learning allows

the solver to identify potentially infeasible subsystems, and activity-driven search heuristics direct the solver to explore those subsystems.

The LCG subproblem detects infeasibility when it is forced to backtrack past the most recent assumption. From the final conflict, we can traverse the implication graph (in the same manner as conflict analysis) to derive a cut C of the form $[\![x_1 \geq k_1]\!] \wedge \cdots \wedge [\![x_n < k_n]\!] \rightarrow \mathit{false}$ consisting only of (possibly relaxed) assumptions. This can be added to the master as: $(1 - [\![x_i \geq k_i]\!]) + \cdots + (1 - [\![x_n < k_n]\!]) \geq 1$. If these cuts happen to be over Boolean variables, then no further action is needed, but for more general cuts such (e.g. $[\![x > 30]\!] \wedge [\![y \leq 10]\!] \rightarrow \mathit{false}$), we must first introduce fresh Boolean variables encoding these atoms.

In the case that all bounds literals from the lower to upper bound have been instantiated, these new literals can be encoded in the MIP by the following constraints:

$$[\![x \geq k+1]\!] \geq [\![x \geq k]\!] \qquad \forall k \geq lb \qquad (1)$$

$$x = lb(x) + \sum_{i=lb(x)+1}^{ub(x)} [\![x \geq i]\!] \qquad (2)$$

However it is desirable to be able to lazily generate these bounds literals only when they appear in some cut. Consequently we define upper and lower bounds for equation 2 which are correct for any subset B of bounds literals belonging to variable x. Assume $B = \{[\![x \geq k_1]\!], \cdots, [\![x \geq k_n]\!]\}$, and $k_i < k_{i+1}$, and let $k_0 = lb(x)$ and $k_{n+1} = ub(x)$.

$$x \geq k_0 + \sum_{i=1}^{n} (k_i - k_{i-1}) [\![x \geq k_i]\!] \qquad (3)$$

$$x \leq k_1 - 1 + \sum_{i=1}^{n} (k_{i+1} - k_i) [\![x \geq k_i]\!] \qquad (4)$$

When B is the full set of bounds literals, we can see that the RHS of both equation 3 and equation 4 converge to the RHS of equation 2. The encoding is revised as fresh bounds are introduced as adding additional bounds literals can only tighten the inequalities. Observe that this encoding permits x to take values *between* introduced bounds.

3.2 Cut minimization

The nogoods $(C \rightarrow \mathit{false})$ derived by the LCG solver will typically involve only a small subset of problem variables – in the case of independent subproblems, the nogoods will refer to only one subproblem. These form valid cuts, but are not necessarily minimal. These cuts can be reduced by applying any of the MUS construction approaches outlined in Section 2.4, we use MUS-seq (Figure 1). This achieves much the same effect as the cut strengthening outlined by (Hooker 2007).

Here we see some side-effects of subproblem completeness. During cut minimization, the subproblem solver may find a model μ^*. As we have only a single complete subproblem containing all the constraints, this model is a feasible solution to the overall problem.

We can then add a constraint to the subproblem constraining the objective to be strictly better than this new solution, potentially allowing the cut to be further simplified. An interesting side effect is that during cut minimization the subproblem solver will find new incumbent solutions, tighten the objective bound and continue searching until it either proves no solutions better than the new incumbent exist in this neighborhood or it exceeds its resource budget. Consequently, if the subproblem solver were executed with an unbounded resource limit, it would always return an empty cut (essentially just running the LCG solver to completion). In the algorithm of Figure 1, this means we never positively identify a bound as being in the MUS; instead, constraints may always be eliminated by a later conflict.

3.3 Deriving multiple independent cuts

Typically in LBBD, each subproblem can be used to learn a cut per iteration. We approximate this using the observation that a minimal cut often contains variables exclusively from one subproblem. Thus, after obtaining a cut $C \rightarrow \mathit{false}$, we attempt to generate additional independent cuts by removing all assumptions which occur in C, and asking the CP solver for a new cut over the remaining assumptions. It is possible that this process will learn multiple independent cuts from the same subsystem before moving on to the next, however it is not obvious that this is a bad thing.

Similar to cut minimization, it is necessary to limit the resource budget of the subproblem solver (*solve-lim*), as the search space implied by the reduced set of assumptions can become arbitrarily large, and the **while** loop only stops generating cuts when the subproblem solver returns UNKNOWN or an empty cut.

3.4 Cut generation as large neighborhood search

As we noted in Section 1, removing assumptions corresponds to growing the neighborhood explored by the subproblem. In both cut-minimisation, and multiple cut generation, we remove assumptions that caused failure from the initial neighbourhood generated by the MIP. This leads to an exploration strategy similar to explanation-based LNS (Prudhomme, Lorca, and Jussien 2014).

The number of assumptions provided to the subproblem solver will vary hugely during the solving process, so we rely on the resource limits to prevent long-tailed solve times. So long as one cut is generated then the MIP is guaranteed to generate a different solution, and the search space will be fully explored eventually. Since the initial solve given the full MIP assignment is run without any limit, this is guaranteed to terminate eventually.

4 Experimental Evaluation

We have implemented the described approach, modifying the assumption interface of CHUFFED, a lazy clause generation solver, to report feasibility cuts.

We evaluated the approach on several sets of scheduling problems, described below. For each class, we tested Chuffed (CP) (Chu 2011), Gurobi 6.5 (MIP) (Gurobi 2016) and our logic based Benders decomposition approach (LBBD). For LBBD, cut minimization was run with a budget of 512 conflicts, after which the current cut was returned.

The problem classes we consider are described in the next four paragraphs. In each case identical MiniZinc models were provided to each solver.

Planning and Scheduling This problem requires scheduling independent tasks on a set of machines with capacity limits, and machine-dependent task durations. These instances have been used to evaluate the effectiveness of logic-based Benders decomposition in a number of prior works (Hooker 2007; Heinz, Ku, and Beck 2013; Ciré, Coban, and Hooker 2013), minimizing cost, makespan or tardiness. We report on models minimizing cost (PS-cost) and makespan (PS-makespan) in our results tables.

Alternative Resource Scheduling with Sequence Dependent Setups This problem, like the planning and scheduling problem, requires scheduling independent tasks on a set of machines. However machines cannot run tasks in parallel, and tasks require setup time which varies with machine and preceding task. The LBBD approach of Tran and Beck (2012) separates the machine allocation from scheduling, resulting in a per-machine TSP which is solved with a dedicated TSP solver. There are 3 subclasses of this benchmark: production-dominated (ARS-p-dom), setup-dominated (ARS-s-dom), and balanced (ARS-balanced).

Single-source capacitated plant location problem The SSCPLP (Barcelo, Fernandez, and Jörnsten 1991) is a discrete plant location problem, where each customer is assigned to a single facility, such that the combined cost of open facilities and customer service is minimized. This is not an ideal candidate for LBBD, having a natural MIP encoding, but an LBBD approach in which the master decides which plants to open and the subproblem assigns customers to plants is similar to the classical benders approach which has been used for variants of this problem (Geoffrion and Graves 1974). The SSCPLP is also interesting as it is the basis for the following problem.

Capacity- and distance-constrained plant location problem The CDCPLP (Albareda-Sambola, Fernández, and Laporte 2009) extends the SSCPLP, adding a fleet of distance-limited vehicles required to service customers. An LBBD approach was presented in (Fazel-Zarandi and Beck 2011) which allocates customers to facilities in the master, leaving a bin-packing subproblem per facility to be solved with a CP solver (after first trying a greedy heuristic).

Results Table 1 compares our LBBD approach (using *Gurobi 6.5* and *Chuffed*) to MIP (*Gurobi 6.5*) and CP (*Chuffed*) on 3 metrics: average solution quality; average time to prove optimality; and the number of instances proved optimal, for each of the benchmark sets described above.

We can see in Figure 4 that *Chuffed* performs very well on some small instances, but for a time-budget of over 13 seconds, LBBD is the fastest to prove optimality of the 3 techniques tested. This is supported by Table 1, where we see

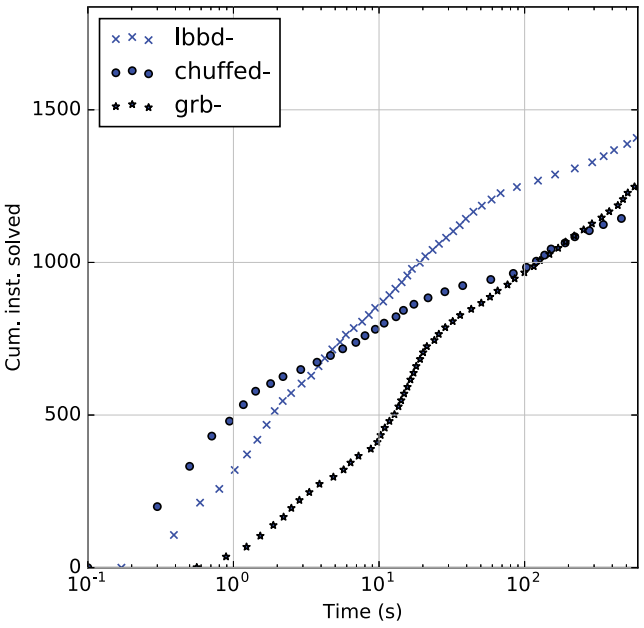

Figure 4: Solutions proved optimal vs time

that LBBD is both fastest on average to prove optimality, and proves the largest number of instances optimal.

In addition to LBBD's expected strength at proving optimality, Table 1 shows that our implementation (including the LNS primal heuristic which naturally occurs from our single-subproblem formulation) makes for an excellent primal solver, finding higher quality solutions on average than both other techniques.

Solution quality obtained by a solver s for each instance i is defined to be $c_{tbp}(i)/c_s(i)$ where $c_s(i)$ is the objective of the best solution to instance i found by s in the time limit, and *tbp* is the theoretical best portfolio of all 3 solvers. We report this as a percentage, which can be seen as a ratio of the performance of the theoretical best portfolio of the 3 solvers. We see that our approach achieves 95% of the performance of the theoretical best portfolio, giving us the best of both of these contrasting optimisation technologies.

However our approach is more than just a portfolio, in Figure 5 we can see that many optimality proofs are faster using our LBBD approach than MIP or CP. In particular note the 79 instances which were only proved optimal by LBBD.

Table 2 examines the hard "c" instances from the planning and scheduling benchmarks in detail. These instances are some of the most studied in the LBBD literature (Hooker 2007; Heinz, Ku, and Beck 2013; Ciré, Coban, and Hooker 2013; 2015), and this table is designed to be reasonably comparable to tables in previous work (Ciré, Coban, and Hooker 2013; 2015). Our experiments use a much shorter time limit than those papers, however we can still make broad comparisons: most notably *Chuffed* performs much better than the traditional CP approaches considered in previous work. To our knowledge, this is the first time a LBBD approach has been directly compared with an LCG CP solver. This strong performance may suggest that the clausal learning of

	Count	Quality %			Time			Num. Optimal			
		LBBD	MIP	CP	LBBD	MIP	CP	LBBD		MIP	CP
PS-makespan	335	**100.0**	57.4	89.0	**61.7**	369.4	101.3	**311**	+25	150	286
PS-cost	335	**100.0**	80.7	88.8	**95.7**	400.1	122.3	**301**	+29	131	281
SSCPLP	57	89.9	**100.0**	72.6	271.9	**95.8**	589.6	34	+0	**50**	1
CDCPLP	300	73.2	**99.8**	57.0	**424.4**	426.7	597.3	99	+23	**140**	3
ARS-balanced	270	**100.0**	**100.0**	83.6	144.3	**47.0**	180.1	224	+0	**269**	211
ARS-p-dom	270	**100.0**	**100.0**	83.6	106.8	**73.9**	179.9	255	+2	**261**	206
ARS-s-dom	270	**100.0**	**100.0**	72.9	207.0	**100.7**	265.8	189	+0	**253**	171
Total	1837	**95.3**	88.7	79.3	**173.8**	245.6	248.6	**1413**	+79	1254	1159

Table 1: Quality score, mean runtime, and number of instances proved optimal in 600s.
+N indicates LBBD solved N instances unsolved by either MIP or CP.

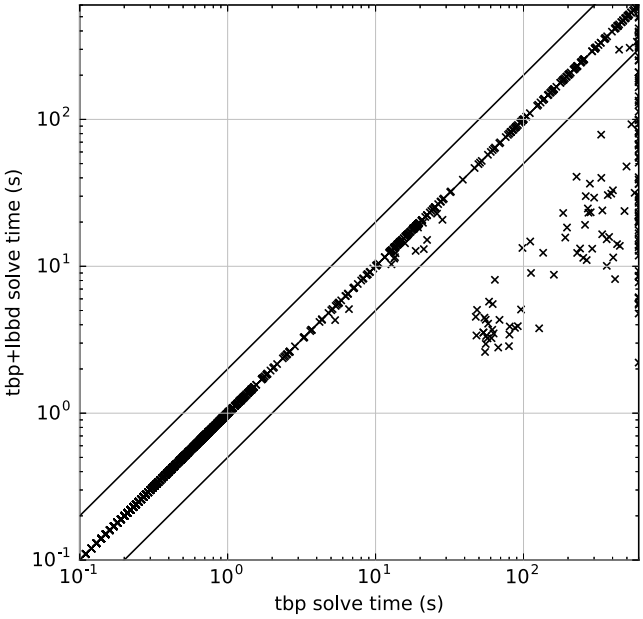

Figure 5: Performance of the the theoretical best portfolio with and without LBBD. 79 instances were solved by LBBD but neither MIP nor CP.

LCG, and cut-generation in LBBD have similar strengths. It is also interesting to note that the custom LBBD solver of Ciré, Coban, and Hooker (2015) was only able to solve 6 more instances in 2 hours than our LBBD approach solved in 10 minutes.

5 Conclusion and Further Work

We have introduced the first "model-and-run" LBBD solver, which additionally uses a natural LNS-like primal heuristic. This solver may not be able to outperform all custom LBBD implementations, e.g. ARS with a dedicated TSP solver, but can tackle any problem with no additional implementation cost. The resulting hybrid combines the strengths of MIP and CP and can be superior to both of them on appropriate problems.

One important feature of many LBBD solvers that we have not addressed is relaxations of the subproblem encoded in the master, especially using continuous variables. We expect this can be achieved by defining relaxations for global constraints using fresh variables (which cannot then cause conflict in the subproblem). We expect these to be important for evaluating this approach against a broader array of benchmarks.

Acknowledgments

This work was inspired by a talk by Fahiem Bacchus on the MaxHS MAXSAT solver (Davies and Bacchus 2011), given at the University of Melbourne. The authors acknowledge the support of Data61; the Australian Research Council through grant DE160100568; and the Asian Office of Aerospace Research and Development grant 15-4016.

References

Albareda-Sambola, M.; Fernández, E.; and Laporte, G. 2009. The capacity and distance constrained plant location problem. *Computers & Operations Research* 36(2):597–611.

Bakker, R. R.; Dikker, F.; Tempelman, F.; and Wognum, P. M. 1993. Diagnosing and Solving Over-Determined Constraint Satisfaction Problems. In *Proceedings of IJCAI-93*, 276–281. Morgan Kaufmann.

Barcelo, J.; Fernandez, E.; and Jörnsten, K. O. 1991. Computational results from a new Lagrangean relaxation algorithm for the capacitated plant location problem. *European Journal of Operational Research* 53(1):38–45.

Chu, G. 2011. *Improving Combinatorial Optimization*. Ph.D. Dissertation, Department of Computing and Information Systems, University of Melbourne.

Ciré, A.; Coban, E.; and Hooker, J. N. 2013. Mixed Integer Programming vs. Logic-Based Benders Decomposition for Planning and Scheduling. In Gomes, C., and Sellmann, M., eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, number 7874 in Lecture Notes in Computer Science. Springer Berlin Heidelberg. 325–331. DOI: 10.1007/978-3-642-38171-3_22.

Ciré, A. A.; Coban, E.; and Hooker, J. 2015. Logic-based benders decomposition for planning and scheduling: a computational analysis.

Davies, J., and Bacchus, F. 2011. Solving maxsat by solving a sequence of simpler sat instances. In *International confer-*

	Num. Optimal			Mean Runtime		
	LBBD	MIP	CP	LBBD	MIP	CP
2m-10j	**5**	**5**	**5**	0.218	17.288	**0.148**
2m-12j	**5**	**5**	**5**	0.304	51.444	**0.162**
2m-14j	**5**	**5**	**5**	0.412	85.742	**0.288**
2m-16j	**5**	**5**	**5**	1.13	211.964	**0.508**
2m-18j	**5**	2	**5**	1.876	406.58+	**0.782**
2m-20j	**5**	2	**5**	1.95	402.704+	**0.814**
2m-22j	**5**	0	**5**	143.898	—	**103.226**
2m-24j	**4**	0	**4**	197.684+	—	**194.526+**
2m-26j	2	0	**3**	**385.768+**	—	447.648+
2m-28j	**4**	0	1	**134.132+**	—	546.024+
2m-30j	**3**	0	2	**418.734+**	—	451.346+
2m-32j	**3**	0	0	**255.462+**	—	—
3m-10j	**5**	**5**	**5**	0.26	17.626	**0.154**
3m-12j	**5**	**5**	**5**	0.414	49.222	**0.24**
3m-14j	**5**	**5**	**5**	0.614	111.21	**0.298**
3m-16j	**5**	1	**5**	1.156	488.596+	**0.474**
3m-18j	**5**	0	**5**	2.82	—	**2.328**
3m-20j	**5**	0	**5**	**2.686**	—	4.392
3m-22j	**5**	0	4	**4.476**	—	125.788+
3m-24j	**5**	0	**5**	**17.948**	—	70.838
3m-26j	**5**	0	3	**44.138**	—	251.972+
3m-28j	**5**	0	2	**67.198**	—	404.374+
3m-30j	**2**	0	**2**	**405.072+**	—	476.176+
3m-32j	**3**	0	1	**421.492+**	—	597.066+
4m-10j	**5**	**5**	**5**	0.294	15.592	**0.19**
4m-12j	**5**	**5**	**5**	0.392	35.982	**0.246**
4m-14j	**5**	4	**5**	0.608	188.058+	**0.294**
4m-16j	**5**	**5**	**5**	0.94	213.0	**0.618**
4m-18j	**5**	1	**5**	1.452	560.298	**0.62**
4m-20j	**5**	0	**5**	2.534	—	**1.272**
4m-22j	**5**	0	**5**	**5.622**	—	13.858
4m-24j	**5**	0	4	**34.734**	—	127.848+
4m-26j	**5**	0	4	**119.762**	—	158.276+
4m-28j	**5**	0	**5**	168.614	—	**29.424**
4m-30j	**3**	0	**3**	411.584+	—	**380.55+**
4m-32j	3	0	**5**	361.884+	—	**326.076**

Table 2: Planning and Scheduling "c" instances: "Xm-Yj" schedules Y jobs over X machines.

ence on principles and practice of constraint programming, 225–239. Springer.

Dershowitz, N.; Hanna, Z.; and Nadel, A. 2006a. A Scalable Algorithm for Minimal Unsatisfiable Core Extraction. In Biere, A., and Gomes, C. P., eds., *Theory and Applications of Satisfiability Testing - SAT 2006*, number 4121 in Lecture Notes in Computer Science. Springer Berlin Heidelberg. 36–41. DOI: 10.1007/11814948_5.

Dershowitz, N.; Hanna, Z.; and Nadel, A. 2006b. A Scalable Algorithm for Minimal Unsatisfiable Core Extraction. In Biere, A., and Gomes, C. P., eds., *Theory and Applications of Satisfiability Testing - SAT 2006*, number 4121 in Lecture Notes in Computer Science. Springer Berlin Heidelberg. 36–41. DOI: 10.1007/11814948_5.

Fazel-Zarandi, M. M., and Beck, J. C. 2011. Using Logic-Based Benders Decomposition to Solve the Capacity- and Distance-Constrained Plant Location Problem. *INFORMS Journal on Computing* 24(3):387–398.

Felfernig, A.; Schubert, M.; and Zehentner, C. 2012. An efficient diagnosis algorithm for inconsistent constraint sets. *AI EDAM* 26(01):53–62.

Geoffrion, A. M., and Graves, G. W. 1974. Multicommodity distribution system design by benders decomposition. *Management science* 20(5):822–844.

2016. Gurobi optimization solver. www.gurobi.com.

Heinz, S.; Ku, W.-Y.; and Beck, J. C. 2013. Recent Improvements Using Constraint Integer Programming for Resource Allocation and Scheduling. In Gomes, C., and Sellmann, M., eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, number 7874 in Lecture Notes in Computer Science. Springer Berlin Heidelberg. 12–27. DOI: 10.1007/978-3-642-38171-3_2.

Hemery, F.; Lecoutre, C.; Sais, L.; and Boussemart, F. 2006. Extracting MUCs from constraint networks. In *In Proceedings of ECAI'06*, 113–117.

Hooker, J. N., and Ottosson, G. 2003. Logic-based Benders decomposition. *Mathematical Programming* 96(1):33–60.

Hooker, J. N. 2007. Planning and Scheduling by Logic-Based Benders Decomposition. *Operations Research* 55(3):588–602.

Junker, U. 2004. Quickxplain: Preferred explanations and relaxations for over-constrained problems. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence*, 167–172. AAAI Press / The MIT Press.

Liffiton, M. H., and Malik, A. 2013. Enumerating Infeasibility: Finding Multiple MUSes Quickly. In Gomes, C., and Sellmann, M., eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, number 7874 in Lecture Notes in Computer Science. Springer Berlin Heidelberg. 160–175. DOI: 10.1007/978-3-642-38171-3_11.

Marques-Silva, J.; Janota, M.; and Belov, A. 2013. Minimal Sets over Monotone Predicates in Boolean Formulae. In Sharygina, N., and Veith, H., eds., *Computer Aided Verification*, number 8044 in Lecture Notes in Computer Science. Springer Berlin Heidelberg. 592–607.

Nethercote, N.; Stuckey, P.; Becket, R.; Brand, S.; Duck, G.; and Tack, G. 2007. Minizinc: Towards a standard CP modelling language. In Bessiere, C., ed., *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, volume 4741 of *LNCS*, 529–543. Springer-Verlag.

Pisinger, D., and Ropke, S. 2010. Large neighborhood search. In *Handbook of metaheuristics*. Springer. 399–419.

Prudhomme, C.; Lorca, X.; and Jussien, N. 2014. Explanation-based large neighborhood search. *Constraints* 19(4):339–379.

Tran, T., T., and Beck, J. C. 2012. Logic-based Benders Decomposition for Alternative Resource Scheduling with Sequence Dependent Setups. volume 242 of *Frontiers in Artificial Intelligence and Applications*, 774–779. IOS Press.