Automatic Logic-based Benders Decomposition with MiniZinc

Toby O. Davies and Graeme Gange and Peter J. Stuckey

Data61 CSIRO
Department of Computing and Information Systems,
The University of Melbourne, Victoria 3010, Australia

Outline

Outline

Multi-resource Scheduling

$$\begin{aligned}\textbf{minimize} \quad & \sum_{r \in R} \text{cost of schedule for } r \\ \textbf{s.t.} \quad & \forall_{t \in T}.\ \text{task } t \text{ is scheduled on some } r \\ & \forall_{r \in R}.\ \text{schedule for } r \text{ is feasible}\end{aligned}$$

Frequently:

- Objective is a linear combination of 0–1 variables
- Feasibility constraint is something nastily combinatorial
  - Cumulative resource capacities, bin packing, . . .

How do we solve it?

How do we solve it?

Integer Programming?

- Extremely good at optimizing linear terms
- Tends to choke on the feasibility constraints
  - Capacity constraints produce large, weak linearizations

How do we solve it?

Integer Programming?

- Extremely good at optimizing linear terms
- Tends to choke on the feasibility constraints
    - Capacity constraints produce large, weak linearizations

Constraint Programming?

- Specialized reasoning for many combinatorial constraints
- Much weaker bounding than MIP.
    - Only tightens objective bounds when defining variables change.

Logic-Based Benders Decomposition

Decompose the problem into a (MIP) master problem over shared variables, and several independent (CP) subproblems.

Master: Assign tasks to machines

Subproblem: Schedule tasks on a single machine

Logic-Based Benders Decomposition

Decompose the problem into a (MIP) master problem over shared variables, and several independent (CP) subproblems.

Master: Assign tasks to machines

Subproblem: Schedule tasks on a single machine

- Find an optimal solution μ to the master
- Search for a feasible extension of μ to each subproblem
  - If all subproblems are feasible, we have found an optimum.
  - Otherwise, add a cut to the master and restart.

In theory, cuts are derived by solving the inference dual.
In practice, some form of generate-and-test.

Logic-Based Benders Decomposition

Decompose the problem into a (MIP) master problem over shared variables, and several independent (CP) subproblems.

Master: Assign tasks to machines

Subproblem: Schedule tasks on a single machine

$$\boxed{M}$$

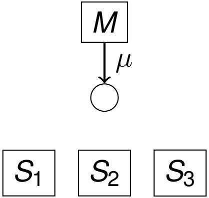
$$\boxed{S_1} \quad \boxed{S_2} \quad \boxed{S_3}$$

Logic-Based Benders Decomposition

Decompose the problem into a (MIP) master problem over shared variables, and several independent (CP) subproblems.

Master: Assign tasks to machines

Subproblem: Schedule tasks on a single machine

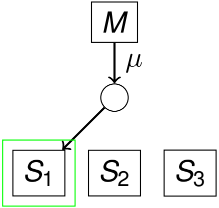


Logic-Based Benders Decomposition

Decompose the problem into a (MIP) master problem over shared variables, and several independent (CP) subproblems.

Master: Assign tasks to machines

Subproblem: Schedule tasks on a single machine

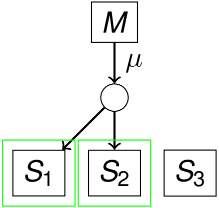


Logic-Based Benders Decomposition

Decompose the problem into a (MIP) master problem over shared variables, and several independent (CP) subproblems.

Master: Assign tasks to machines

Subproblem: Schedule tasks on a single machine

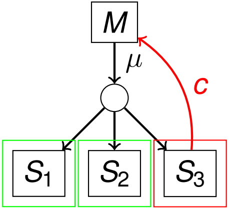


Logic-Based Benders Decomposition

Decompose the problem into a (MIP) master problem over shared variables, and several independent (CP) subproblems.

Master: Assign tasks to machines

Subproblem: Schedule tasks on a single machine

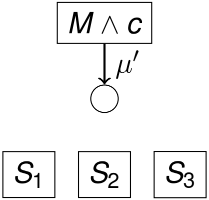


Logic-Based Benders Decomposition

Decompose the problem into a (MIP) master problem over shared variables, and several independent (CP) subproblems.

Master: Assign tasks to machines

Subproblem: Schedule tasks on a single machine

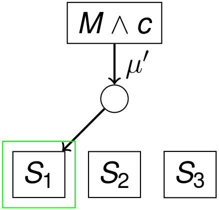


Logic-Based Benders Decomposition

Decompose the problem into a (MIP) master problem over shared variables, and several independent (CP) subproblems.

Master: Assign tasks to machines

Subproblem: Schedule tasks on a single machine

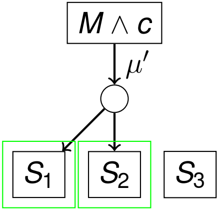


Logic-Based Benders Decomposition

Decompose the problem into a (MIP) master problem over shared variables, and several independent (CP) subproblems.

Master: Assign tasks to machines

Subproblem: Schedule tasks on a single machine

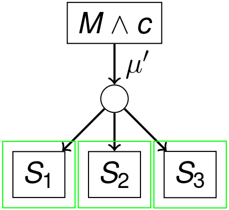


Logic-Based Benders Decomposition

Decompose the problem into a (MIP) master problem over shared variables, and several independent (CP) subproblems.

Master: Assign tasks to machines

Subproblem: Schedule tasks on a single machine



Automating Logic-based Benders Decomposition

An effective strategy, but sees surprisingly little use.

- Specialized implementation per-problem.
- One implementation per PhD

Automating Logic-based Benders Decomposition

An effective strategy, but sees surprisingly little use.

- Specialized implementation per-problem.
- One implementation per PhD

Limitations to be aware of:

- Frequently all-or-nothing (optimal solution or none)
- Subproblems must be fully independent (no coupling)

Automating Logic-based Benders Decomposition

An effective strategy, but sees surprisingly little use.

- Specialized implementation per-problem.
- One implementation per PhD

Limitations to be aware of:

- Frequently all-or-nothing (optimal solution or none)
- Subproblems must be fully independent (no coupling)

What elements do we need for automating LBBD?

1. Automatic partitioning into master/subproblems
2. Systematic extraction of cuts from arbitrary subproblems

Outline

MiniZinc: A solver-independent modelling language

- solver-independent
  - supported by CP, MIP, SAT, SMT, and local search solvers
- high-level
  - encode combinatorial substructures directly as global constraints
- defacto standard for CP modelling

Hands On Session

Learn MiniZinc: Wednesday 28th: 11:00 - 12:30

MiniZinc: How it works

MiniZinc High-level model specification translates to ...

```
constraint forall (m in machines) (
  cumulative(
    [starts[j] | j in jobs],
    [duration[j,m] | j in jobs],
    [resource[j,m]*bool2int(assign[j] = m) | j in
        jobs],
    capacities[m]
  )
);
```

FlatZinc Variable declarations and primitive constraints

```
% ...
constraint cumulative(X_INTRODUCED_233,
    X_INTRODUCED_235,X_INTRODUCED_234,15);
constraint int_lin_le([1,-1],[X_INTRODUCED_27,
    objective],-6);
% ...
```

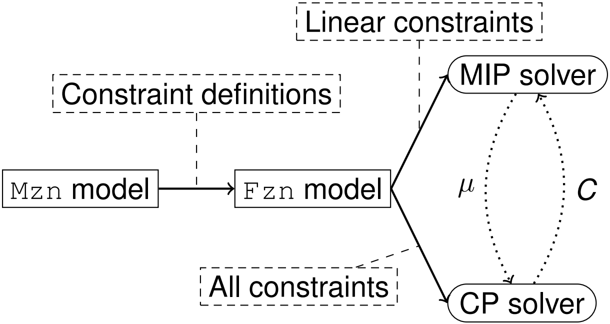
Flattening uses a solver-specific library of transformations.

Outline

Automating 'decomposition'

A simple strategy: MIP master, single CP subproblem.

- Master contains all linear inequalities (and corresponding variables).
- Subproblem contains everything (as if solving directly with CP).



Implicit subproblems

With classical CP, this is a terrible idea.

$$P = \begin{array}{l} p_1 + p_2 + p_3 \leq 2 \\ \wedge\ x_1 + x_2 \leq p_1 \\ \wedge\ y_1 + y_2 \leq p_2 \\ \wedge\ z_1 = z_2 + p_3 \wedge z_1 \neq z_2 \end{array}$$

Assuming $\{p_1 = 1, p_2 = 1, p_3 = 0\}$ set by master:

x_1	x_2	y_1	y_2	z_1	z_2
$[0, 1]$	$[0, 1]$	$[0, 1]$	$[0, 1]$	$[0, 1]$	$[0, 1]$
0					

Implicit subproblems

With classical CP, this is a terrible idea.

$$P = \begin{array}{l} \quad p_1 + p_2 + p_3 \leq 2 \\ \wedge\ x_1 + x_2 \leq p_1 \\ \wedge\ y_1 + y_2 \leq p_2 \\ \wedge\ z_1 = z_2 + p_3 \wedge z_1 \neq z_2 \end{array}$$

Assuming $\{p_1 = 1, p_2 = 1, p_3 = 0\}$ set by master:

x_1	x_2	y_1	y_2	z_1	z_2
$[0,1]$	$[0,1]$	$[0,1]$	$[0,1]$	$[0,1]$	$[0,1]$
0	0				

Implicit subproblems

With classical CP, this is a terrible idea.

$$P = \begin{array}{l} p_1 + p_2 + p_3 \leq 2 \\ \wedge\ x_1 + x_2 \leq p_1 \\ \wedge\ y_1 + y_2 \leq p_2 \\ \wedge\ z_1 = z_2 + p_3 \wedge z_1 \neq z_2 \end{array}$$

Assuming $\{p_1 = 1, p_2 = 1, p_3 = 0\}$ set by master:

x_1	x_2	y_1	y_2	z_1	z_2
[0, 1]	[0, 1]	[0, 1]	[0, 1]	[0, 1]	[0, 1]
0	0	0	0	0	

Implicit subproblems

With classical CP, this is a terrible idea.

$$P = \begin{array}{l} p_1 + p_2 + p_3 \leq 2 \\ \wedge\ x_1 + x_2 \leq p_1 \\ \wedge\ y_1 + y_2 \leq p_2 \\ \wedge\ z_1 = z_2 + p_3 \wedge z_1 \neq z_2 \end{array}$$

Assuming $\{p_1 = 1, p_2 = 1, p_3 = 0\}$ set by master:

x_1	x_2	y_1	y_2	z_1	z_2
[0, 1]	[0, 1]	[0, 1]	[0, 1]	[0, 1]	[0, 1]
0	0	0	0	0	X

Implicit subproblems

With classical CP, this is a terrible idea.

$$P = \begin{array}{l} p_1 + p_2 + p_3 \leq 2 \\ \wedge\ x_1 + x_2 \leq p_1 \\ \wedge\ y_1 + y_2 \leq p_2 \\ \wedge\ z_1 = z_2 + p_3 \wedge z_1 \neq z_2 \end{array}$$

Assuming $\{p_1 = 1, p_2 = 1, p_3 = 0\}$ set by master:

x_1	x_2	y_1	y_2	z_1	z_2
[0, 1]	[0, 1]	[0, 1]	[0, 1]	[0, 1]	[0, 1]
0	0	0	0	0	X
0	0	0	0	1	

Implicit subproblems

With classical CP, this is a terrible idea.

$$P = \begin{array}{l} p_1 + p_2 + p_3 \leq 2 \\ \wedge\ x_1 + x_2 \leq p_1 \\ \wedge\ y_1 + y_2 \leq p_2 \\ \wedge\ z_1 = z_2 + p_3 \wedge z_1 \neq z_2 \end{array}$$

Assuming $\{p_1 = 1, p_2 = 1, p_3 = 0\}$ set by master:

x_1	x_2	y_1	y_2	z_1	z_2
[0, 1]	[0, 1]	[0, 1]	[0, 1]	[0, 1]	[0, 1]
0	0	0	0	0	X
0	0	0	0	1	X

Implicit subproblems

With classical CP, this is a terrible idea.

$$P = \begin{array}{l} p_1 + p_2 + p_3 \leq 2 \\ \wedge\ x_1 + x_2 \leq p_1 \\ \wedge\ y_1 + y_2 \leq p_2 \\ \wedge\ z_1 = z_2 + p_3 \wedge z_1 \neq z_2 \end{array}$$

Assuming $\{p_1 = 1, p_2 = 1, p_3 = 0\}$ set by master:

x_1	x_2	y_1	y_2	z_1	z_2
[0, 1]	[0, 1]	[0, 1]	[0, 1]	[0, 1]	[0, 1]
0	0	0	0	0	X
0	0	0	0	1	X
0	0	0	1		

Implicit subproblems

With classical CP, this is a terrible idea.

$$P = \begin{array}{l} p_1 + p_2 + p_3 \leq 2 \\ \wedge\ x_1 + x_2 \leq p_1 \\ \wedge\ y_1 + y_2 \leq p_2 \\ \wedge\ z_1 = z_2 + p_3 \wedge z_1 \neq z_2 \end{array}$$

Assuming $\{p_1 = 1, p_2 = 1, p_3 = 0\}$ set by master:

x_1	x_2	y_1	y_2	z_1	z_2
[0, 1]	[0, 1]	[0, 1]	[0, 1]	[0, 1]	[0, 1]
0	0	0	0	0	X
0	0	0	0	1	X
0	0	0	1	0	X

...

Lazy Clause Generation (LCG)

Descendant of CP and SAT:

- CP-style propagators
- SAT-style conflict analysis

Operates on 'atomic constraints' $[\![x \geq k]\!]$, $[\![x = k]\!]$.

Key attributes (for our purposes):

- Conflict analysis
  - Cuts to explain failure.
- Activity-driven search
  - Focus on hard-to-satisfy subproblems.
- Phase-saving
  - Save successful partial assignments we find.

Implicit subproblems, with LCG

With $\{p_1 = 1, p_2 = 1, p_3 = 0\}$:

x_1	x_2	y_1	y_2	z_1	z_2
[0, 1]	[0, 1]	[0, 1]	[0, 1]	[0, 1]	[0, 1]
0	0	0	0	0	X

Implicit subproblems, with LCG

With $\{p_1 = 1, p_2 = 1, p_3 = 0\}$:

x_1	x_2	y_1	y_2	z_1	z_2	
$[0, 1]$	$[0, 1]$	$[0, 1]$	$[0, 1]$	$[0, 1]$	$[0, 1]$	
0	0	0	0	0	X	$[\![p_3 \geq 1]\!] \vee [\![z_1 \geq 1]\!]$

Implicit subproblems, with LCG

With $\{p_1 = 1, p_2 = 1, p_3 = 0\}$:

x_1	x_2	y_1	y_2	z_1	z_2	
$[0, 1]$	$[0, 1]$	$[0, 1]$	$[0, 1]$	$[0, 1]$	$[0, 1]$	
0	0	0	0	0	X	$[\![p_3 \geq 1]\!] \vee [\![z_1 \geq 1]\!]$
$[0, 1]$	$[0, 1]$	$[0, 1]$	$[0, 1]$	1		

Implicit subproblems, with LCG

With $\{p_1 = 1, p_2 = 1, p_3 = 0\}$:

x_1	x_2	y_1	y_2	z_1	z_2	
$[0, 1]$	$[0, 1]$	$[0, 1]$	$[0, 1]$	$[0, 1]$	$[0, 1]$	
0	0	0	0	0	X	$[\![p_3 \geq 1]\!] \vee [\![z_1 \geq 1]\!]$
$[0, 1]$	$[0, 1]$	$[0, 1]$	$[0, 1]$	1	X	

Implicit subproblems, with LCG

With $\{p_1 = 1, p_2 = 1, p_3 = 0\}$:

x_1	x_2	y_1	y_2	z_1	z_2	
$[0,1]$	$[0,1]$	$[0,1]$	$[0,1]$	$[0,1]$	$[0,1]$	
0	0	0	0	0	X	$[\![p_3 \geq 1]\!] \vee [\![z_1 \geq 1]\!]$
$[0,1]$	$[0,1]$	$[0,1]$	$[0,1]$	1	X	$[\![p_3 \geq 1]\!]$

Implicit subproblems, with LCG

With $\{p_1 = 1, p_2 = 1, p_3 = 0\}$:

x_1	x_2	y_1	y_2	z_1	z_2	
$[0,1]$	$[0,1]$	$[0,1]$	$[0,1]$	$[0,1]$	$[0,1]$	
0	0	0	0	0	X	$[\![p_3 \geq 1]\!] \vee [\![z_1 \geq 1]\!]$
$[0,1]$	$[0,1]$	$[0,1]$	$[0,1]$	1	X	$[\![p_3 \geq 1]\!]$

Most of the benefits of explicit partitioning, plus:

- Disjointness isn't required
- We get cuts for free

Strengthening cuts

The nogoods we obtain are usually not minimal.

- Choose a strict subset of the current cut, solve again.
  - If $\texttt{UNSAT}(C)$, we have a new, stronger cut.
  - If $\texttt{SAT}(\mu)$, at least one element is needed.
- Repeat until we find a minimal cut (or expend computation budget)

Strengthening cuts

The nogoods we obtain are usually not minimal.

- Choose a strict subset of the current cut, solve again.
  - If $\texttt{UNSAT}(C)$, we have a new, stronger cut.
  - If $\texttt{SAT}(\mu)$, at least one element is needed.
- Repeat until we find a minimal cut (or expend computation budget)

However! The ‘subproblem’ is complete.
Thus μ is a feasible (though not optimal) solution.

We can then tighten bounds on the objective:

- in the master, to get earlier fathoming
- in the subproblem, to derive tighter cuts

A Dual viewpoint of Logic Based Benders

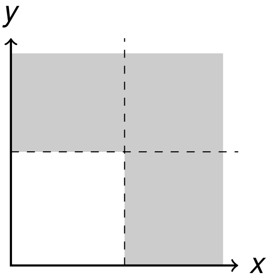
- Master (usual) perspective
  - Master solves relaxed problem
  - Subproblem solver extends master solution or adds cut
- Reversed perspective
  - Master generates a partial solution likely to be “good”
  - CP solver uses this as a basis for Large Neighbourhood Search to find good solutions

Representing cuts

Nogoods from the LCG solver are disjunctions of bounds.

$$[\![x \geq 10]\!] \vee [\![y \geq 10]\!]$$

Problem: Can't be directly expressed as a linear inequality.



Reifying bounds

Lazily introduce 0–1 variables for relevant bounds:

$$x \geq 0 + 10b_{[\![x \geq 10]\!]} + 5b_{[\![x \geq 15]\!]}$$
$$x < 10 + 5b_{[\![x \geq 10]\!]} + 35b_{[\![x \geq 15]\!]}$$

$$b_{[\![x \geq 10]\!]} \geq b_{[\![x \geq 15]\!]}$$

Reifying bounds

Lazily introduce 0–1 variables for relevant bounds:

$$x \geq 0 + 10b_{[\![x \geq 10]\!]} + 5b_{[\![x \geq 15]\!]}$$
$$x < 10 + 5b_{[\![x \geq 10]\!]} + 35b_{[\![x \geq 15]\!]}$$

$$b_{[\![x \geq 10]\!]} \geq b_{[\![x \geq 15]\!]}$$

Which we then use to express cuts:

$$b_{[\![x \geq 10]\!]} + b_{[\![y \geq 10]\!]} \geq 1$$

Outline

Experiments

Several classes of instances:

- Planning and scheduling
  Common LBBD benchmark
- Single-source capacitated plant location
  Pure MIP
- Job shop scheduling w. machine & order-dependent setup times
  TSP subproblem

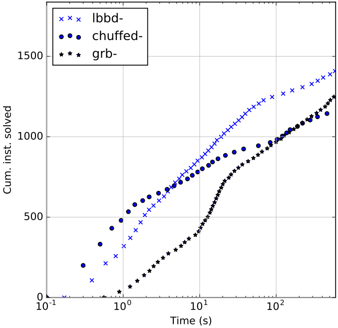
Comparing:

`chuffed` an LCG solver

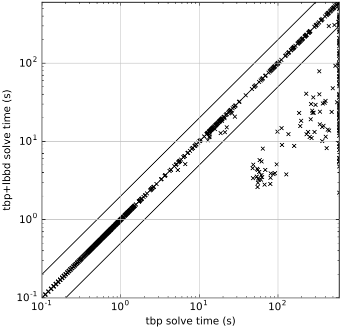
`Gurobi` a MIP solver

`mzn-lbbd` automatic LBBD method (using `Gurobi` and `chuffed`)

Results



instances solved



Theoretical best portfolio, with and without `mzn-lbbd`.

Results: Observations

- Doesn't strictly dominate either CP or MIP
  - but robust, and performs better in aggregate
- Not just best-of-both-worlds
  - Solves 79 instances not solved by either CP or MIP.
- Doesn't compete with Benders' methods with specialized (non-CP) subproblem solvers.
  - TSP subproblems, etc.

Outline

Conclusion

- Automatic Logic Based Benders provides a hybrid of
  - Integer Programming, and
  - Constraint Programming
- Takes advantage of the strengths of both methods
- One PhD worth of implementation is reduced to writing one model!

Further work

Many parameters to tune (globally, or per domain):

- Cut minimization strategy
- Generating multiple cuts
- Resource limits

Master currently includes no relaxation of omitted constraints.