DM877

Discrete Optimization

Introduction to MiniZinc: The Grocery Example

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

The Grocery Problem

Kid goes to store and buys four items

Cashier:	that makes \$7.11
Kid:	pays, about to leave store
Cashier:	hold on, I multiplied! let me add! wow, the sum is also \$7.11
You:	prices of the four items?

Modeling:

In mathematical notation:

- Set of items I indexed by i.
- Parameters: total sum and product $s = 7, 11$
- Decision variables: prices $p_i, i \in I$
- Constraints:

$$\sum_{i \in I} p_i = s$$
$$\prod_{i \in I} p_i = s$$

In programming notation:

- Set of items $\texttt{Items}$ indexed by i.
- Parameters: total sum and product $\texttt{total} = 7, 11$
- Decision variables: prices $\texttt{item}_\texttt{i}, i \in \texttt{Items}$
- Constraints:

$$\texttt{item1} + \texttt{item2} + \texttt{item3} + \texttt{item4} = \texttt{total}$$
$$\texttt{item1} \cdot \texttt{item2} \cdot \texttt{item3} \cdot \texttt{item4} = \texttt{total}$$

The Grocery Problem

We transform the problem into an integer problem:

$$(100)\sum_{i\in I} p_i = 7,11(100)$$

Hence, setting $p'_i = 100p_i$, $i \in I$ we can rewrite:

$$\sum_{i\in I} p'_i = 711$$

and consequently

$$\prod_{i\in I} p'_i = 7,11(10^8)$$

MiniZinc model

Thus the model becomes:

```
var int: item1;
var int: item2;
var int: item3;
var int: item4;

constraint item1 + item2 + item3 + item4 == 711;
constraint item1 * item2 * item3 * item4 == 711 * 100 * 100 * 100;

solve satisfy;

output ["{", show(item1), ",", show(item2), ",", show(item3), ",",
   show(item4),"}\n"];
```

My Minizinc Installation

```
marco@sieglinde:~/Teaching/ay2021/DM877/MiniZinc$ minizincs -solvers
MiniZinc driver.
Available solver configurations:
  Chuffed 0.9 (org.chuffed.chuffed, cp, lcg, int)
  Gecode 6.2.0 (org.gecode.gecode, cp, int, float, set, restart)
  OR Tools 7.0/stable (org.ortools.ortools, cp, int, lcg, or-tools)
Search path for solver configurations:
  /opt/MiniZinc/libminizinc-master/share/minizinc/solvers
```

Gecode:

```
marco@sieglinde:~/Teaching/ay2021/DM877/MiniZinc$ minizincs -solver gecode grocery0.mzn
```

nothing, simply hanging.

Chuffed

```
marco@sieglinde:~/Teaching/ay2021/DM877/MiniZinc$ minizincs -solver chuffed grocery0.mzn
=====UNSATISFIABLE=====
% Top level failure!
```

OR-tools

```
marco@sieglinde:~/Teaching/ay2021/DM877/MiniZinc$ minizincs -solver or-tools grocery0.mzn
Not supported
=====ERROR=====
```

The Flatzinc model

```
array [1..4] of int: X_INTRODUCED_0_ = [1,1,1,1];
var int: item1:: output_var;
var int: item2:: output_var;
var int: item3:: output_var;
var int: item4:: output_var;
var int: X_INTRODUCED_2_ ::var_is_introduced :: is_defined_var;
var int: X_INTRODUCED_3_ ::var_is_introduced :: is_defined_var;
var int: X_INTRODUCED_4_ ::var_is_introduced :: is_defined_var;
constraint int_lin_eq(X_INTRODUCED_0_,[item3,item2,item1,item4],711);
constraint int_eq(X_INTRODUCED_4_,711000000);
constraint int_times(item1,item2,X_INTRODUCED_2_):: defines_var(X_INTRODUCED_2_);
constraint int_times(X_INTRODUCED_2_,item3,X_INTRODUCED_3_):: defines_var(X_INTRODUCED_3_);
constraint int_times(X_INTRODUCED_3_,item4,X_INTRODUCED_4_):: defines_var(X_INTRODUCED_4_);
solve  satisfy;
```

Two points:

- the domain is not finite
- the multiplication is split into parts (because $\prod$ is not available):

$$p'_1 \cdot p'_2 = x \qquad x \cdot p'_3 = y \qquad y \cdot p'_4 = z$$

chuffed might give to x, y, z a default domain, eg, $[-1000000; 1000000]$ which makes the problem unsatisfiable

Let's make the domain finite and let's split in our model the multiplication to control the domain of the auxiliary variables:

```
var 1..711: item1;
var 1..711: item2;
var 1..711: item3;
var 1..711: item4;

var 1..711*10^2: x;
var 1..711*10^4: y;

constraint
    item1 + item2 + item3 + item4 == 711;
constraint
    item1 * item2 == x /\
    x * item3 == y /\
    y * item4 == 711 * 100 * 100 * 100;

solve satisfy;

output ["{", show(item1), ",", show(item2), ",", show(item3), ",",
    show(item4),"}\n"];
```

For some reasons this does not work either with gecode

Gecode:

```
marco@sieglinde:~/Teaching/ay2021/DM877/MiniZinc$ minizincs -s -solver gecode grocery.mzn
Error: invalid integer literal in line no. 8
Error: syntax error, unexpected ':', expecting FZ_INT_LIT in line no. 8
=====ERROR=====
```

Chuffed

```
marco@sieglinde:~/Teaching/ay2021/DM877/MiniZinc$ minizincs -s -solver chuffed grocery.mzn
150,316,120,125
-------
%%%mzn-stat: nodes=76
%%%mzn-stat: failures=52
%%%mzn-stat: variables=6597
%%%mzn-stat: intVars=7
%%%mzn-stat: boolVariables=6588
%%%mzn-stat: propagators=6
%%%mzn-stat: propagations=6490
%%%mzn-stat: peakDepth=16
%%%mzn-stat: solveTime=0.002
```

OR-tools

```
marco@sieglinde:~/Teaching/ay2021/DM877/MiniZinc$ minizincs -s -solver or-tools grocery.mzn
120,125,150,316
-------
%%%mzn-stat: boolVariables=432
%%%mzn-stat: failures=22284
%%%mzn-stat: propagations=7012478
%%%mzn-stat: solveTime=1.27326
```

FlatZinc

```
marco@sieglinde:~/DM877/MiniZinc$ o grocery.fzn
array [1..4] of int: X_INTRODUCED_0_ = [1,1,1,1];
var 1..711: item1:: output_var;
var 1..711: item2:: output_var;
var 1..711: item3:: output_var;
var 1..711: item4:: output_var;
var 1..71100: x:: is_defined_var;
var 1..7110000: y:: is_defined_var;
var 1..5055210000: X_INTRODUCED_4_ ::var_is_introduced :: is_defined_var;
array [1..4] of var int: X_INTRODUCED_5_ ::var_is_introduced  = [item1,item2,item3,item4];
constraint int_lin_eq(X_INTRODUCED_0_,[item3,item2,item1,item4],711);
constraint int_eq(X_INTRODUCED_4_,711000000);
constraint int_times(item1,item2,x):: defines_var(x);
constraint int_times(x,item3,y):: defines_var(y);
constraint int_times(y,item4,X_INTRODUCED_4_):: defines_var(X_INTRODUCED_4_);
solve :: int_search(X_INTRODUCED_5_,input_order,indomain_split,complete) satisfy;
```

The domain of X_INTRODUCED_4_ is wrongly deduced.

Let's make also z (ie, X_INTRODUCED_4_) explicit:

```
var 1..711: item1;
var 1..711: item2;
var 1..711: item3;
var 1..711: item4;

var 1..711*10^2: x;
var 1..711*10^4: y;

var 711000000..711000000: z;

constraint item1 + item2 + item3 + item4 == 711;
constraint item1 * item2 == x /\
           x * item3 == y /\
           y * item4 == z;

solve satisfy;

output ["{", show(item1), ",", show(item2), ",", show(item3), ",",
    show(item4),"}\n"];
```

	gecode	chuffed	or-tools
solution	{316,150,125,120}	{150,316,120,125}	{120,125,150,316}
intVars	7	7	7
boolVariables		6597	432
initTime (sec)	0.00029	0.004	
solveTime (sec)	0.001317	0.003	1.22328
propagators	4	6	
propagations	2790	6490	7012478
nodes	222	76	
restarts	0	0	
nogoods		52	
failures	109	52	22284
peakDepth	9	16	

Observation

- Multiplication decomposed as

```
item1 * item2 = X
X * item3 = Y
y * item4 = Z
```

- What if

```
item1 * item2 = X
item3 * item4 = Y
X * Y = Z
```

propagation changes: from 222 to 814 nodes
propagation is not compositional!
another point to investigate
but the situation improves for or-tools (knowledge of the solver helps modeling)

	gecode	chuffed	or-tools
solution	{316,150,125,120}	{120,125,316,150}	{120,125,150,316}
intVars	7	7	7
boolVariables		19028	124
initTime (sec)	0.000322	0.004	
solveTime (sec)	0.092207	0.037	0.460046
propagators	4	6	
propagations	16409	292043	3982533
nodes	814	408	
restarts	0	0	
nogoods		278	
failures	405	278	177
peakDepth	9	24	

Branching

With large domains:

- Bad idea: try values one by one
- Good idea: split variables
  for variable x
  with $m = (\min(x) + \max(x))/2$
  branch $x < m$ or $x \geq m$
- Typically good for problems involving arithmetic constraints exact reason needs to be explained later

```
solve :: int_search([item1, item2, item3, item4], input_order, indomain_split) satisfy;
```

	gecode	chuffed	or-tools
solution	{120,125,150,316}	{120,125,150,316}	{120,125,150,316}
intVars	7	7	7
boolVariables		82512	553
initTime (sec)	0.000351	0.004	
solveTime (sec)	0.016565	2.591	0.594197
propagators	4	6	
propagations	113432	1079860	2090022
nodes	8535	4828	
restarts	0	0	
nogoods		52	
failures	4261	4209	3781
peakDepth	18	15	

Gecode worsen from 222 to 8535 nodes
Chuffed worsen from 76 to 4828 nodes
OR-tools improves from 1.22 to 0.59 sec.

Better Heuristic?

- Try branches in different order
- split with larger interval first
  try: `indomain_reverse_split`: bisect the variable's domain, excluding the lower half first.

	gecode	chuffed	or-tools
solution	{316,150,125,120}	{316,150,125,120}	{316,150,125,120}
intVars	7	7	7
boolVariables		12592	343
initTime (sec)	0.00042	0.004	
solveTime (sec)	0.002205	0.039	0.805674
propagators	4	6	
propagations	17710	114363	209384
nodes	1479	757	
restarts	0	0	
nogoods		734	
failures	735	734	694
peakDepth	12	10	

Gecode worsen from 222 to 1579 but improves over the other heuristic that had 8535 nodes
Chuffed worsen from 76 to 757 but improves over the other heuristic that had 4828 nodes
OR-tools improves from 1.22 to 0.81 but worsen wrt the other heuristic 0.59 sec.

Symmetries

- Interested in values for `item1, item2, item3, item4`:
- Model admits equivalent solutions
  interchange (shuffle, permute) values for `item1, item2, item3, item4`
- We can add an order for `item1, item2, item3, item4`:
  `item1 <= item2 <= item3 <= item4`
  called symmetry breaking constraint

```
% symmetry breaking
constraint item1 <= item2 /\ item2 <= item3 /\ item3 <= item4;
```

(we keep branching on `indomain_split`)

	gecode	chuffed	or-tools
solution	{120,125,150,316}	{120,125,150,316}	{120,125,150,316}
intVars	7	7	7
boolVariables		13366	250
initTime (sec)	0.000288	0.004	
solveTime (sec)	0.005352	0.056	0.102721
propagators	7	9	
propagations	65157	232133	336087
nodes	2748	1326	
restarts	0	0	
nogoods		1005	
failures	1367	1005	1086
peakDepth	17	12	

Gecode worsen from 222 to 1579 to 2748 nodes
Chuffed worsen from 76 to 757 to 1326 nodes
OR-tools improves from 1.22 to 0.59 to 0.1027 sec.

Effect of Symmetry Breaking

- Let us try `indomain_reverse_split`: again
- interaction between branching and symmetry breaking other possibility: we need to investigate more (later)!

	gecode	chuffed	or-tools
solution	{120,125,150,316}	{120,125,150,316}	{120,125,150,316}
intVars	7	7	7
boolVariables		7368	123
initTime (sec)	0.000321	0.004	
solveTime (sec)	0.023558	0.005	0.0227684
propagators	7	9	
propagations	6837	21456	28017
nodes	364	232	
restarts	0	0	
nogoods		161	
failures	179	161	145
peakDepth	13	9	

Gecode worsen from 222 to 1579 to 2748 but now 364 nodes
Chuffed worsen from 76 to 757 to 1326 but now 232 nodes
OR-tools improves from 1.22 to 0.59 to 0.1027 and now to 0.023 sec.

Any More Symmetries?

- Observe: 711 has prime factor 79
  that is: $711 = 79 \times 9$

- Assume: `item1` can be divided by 79
  add: `item1 = 79 * D`
  for some finite domain `var D`
  remove: `item1 <= item2`
  the remaining `item1, item2, item3` of course can still be ordered

```
var 1..9: D;

% symmetry breaking
constraint item2 <= item3 /\ item3 <= item4;

constraint
        item1 = 79 * D;
```

	gecode	chuffed	or-tools
solution	{316,120,125,150}	{316,120,125,150}	{316,120,125,150}
intVars	8	8	8
boolVariables		6038	29
initTime (sec)	0.000321	0.004	
solveTime (sec)	0.000129	0.001	0.00306466
propagators	7	10	
propagations	706	2374	1938
nodes	28	22	
restarts	0	0	
nogoods		10	
failures	10	10	9
peakDepth	10	9	

Gecode improves a lot: from 222 to 364 (split + reverse + symmetry) but now 28 nodes
Chuffed worsen from 76 to 232 (split + reverse + symmetry) but now 22 nodes
OR-tools improves from 1.22 to 0.023 sec. (split + reverse + symmetry) and now 0.003 sec

Summary: Grocery

- Branching: consider also
  - how to partition domain
  - in which order to try alternatives
- Symmetry breaking
  - can reduce search space
  - might interact with branching
  - typical: order variables in solutions
- Try to really understand problem!

Domination Constraints

- In symmetry breaking, prune solutions without interest
- Similarly for best solution search
  - typically, interested in just one best solution
  - impose constraints to prune some solutions with same "cost"