Very_cool_project.mzn

Valerio Costa, Luca Domeniconi, Claudia Maiolino, Tian Cheng Xia
{valerio.costa, luca.domeniconi5, claudia.maiolino, tiancheng.xia}@studio.unibo.it

1 Introduction

2 CP model

2.1 Decision variables

The CP model is inspired by the model presented in [1] and relies on the following decision variables:

- For each package p, $A_p \in \{1, \dots, m\}$ (`assignments[p]` in MiniZinc) indicates which courier delivers it. More specifically, $A_p = c$ iff the package p is delivered by the courier c.
- For each courier c, $P_{c,d} \in \{1, \dots, n+1\}$ for $d \in \{1, \dots, n+1\}$ (`path[c, d]` in MiniZinc) models the path taken by the courier c. The path is defined such that $P_{c,d_1} = d_1$ iff the location d_1 is not part of the route of c and $P_{c,d_1} = d_2$ iff d_2 is visited immediately after d_1 in the route of c. In other words, each relevant location indicates which is its successor and the overall route is defined as a Hamiltonian cycle that starts from $n+1$ (i.e., the depot).

2.2 Objective function

Once the route of each courier has been found, the objective function is computed as follows:

$$\max_{c \in \{1,\dots,m\}} \sum_{\substack{d \in \{1,\dots,n+1\}: \\ P_{c,d} \neq d}} \texttt{D[}d\texttt{, } P_{c,d}\texttt{]} \tag{1}$$

The lower-bound of this function corresponds to the longest route that involves a single item and can be computed as follows:

$$\max_{p \in \{1,\dots,n\}} \{\texttt{D[}n+1\texttt{, } p\texttt{]} + \texttt{D[}p\texttt{, } n+1\texttt{]}\} \tag{2}$$

2.3 Constraints

To respect the capacity limits of each courier, the following constraint can be defined:

$$\forall c \in \{1,\dots,m\} : \sum_{p \in \{1,\dots,n\} : A_p = c} s_p \leq l_c \tag{3}$$

In MiniZinc, the global constraint `bin_packing_capa` also models this. In our experiments, we did not notice significant differences between the two formulations and decided to use the latter following the best practice of preferring global constraints.

To model the route, the following constraints have to be imposed:

$$\forall c \in \{1, \dots, m\} : \begin{cases} P_{c,n+1} = n+1 & \text{if } \nexists p \in \{1, \dots, n\} : A_p = c \\ P_{c,n+1} \neq n+1 & \text{if } \exists p \in \{1, \dots, n\} : A_p = c \end{cases} \tag{4}$$

$$\forall c \in \{1, \dots, m\}, \forall p \in \{1, \dots, n\} : \begin{cases} P_{c,p} = p & \text{if } A_p \neq c \\ P_{c,p} \neq p & \text{if } A_p = c \end{cases} \tag{5}$$

The constraint defined in Equation (4) imposes that, for each courier, the depot has a successor only if that courier delivers at least a package. On the same note, Equation (5) imposes that only the packages delivered by a specific courier have a successor in the route. Moreover, it is necessary to impose that the route defined by P_c is a Hamiltonian cycle that passes through the relevant destinations. This can be done by constraining all the elements of P_c to be different and by defining a subtour elimination constraint (e.g., Miller-Tucker-Zemlin [2]). In MiniZinc, this can be easily modelled by using the global constraint `subcircuit`.

2.3.1 Symmetry breaking constraints

The most notable symmetry in this problem is between couriers with the same capacity. In fact, as the assignments between two couriers with the same capacity are interchangeable, it is reasonable to fix an ordering and avoid redundancies during search. We experimented with two approaches:

- By imposing an ordering on the amount of assigned packages:

$$\forall c_1, c_2 \in \{1, \dots, m\} : (c_1 < c_2 \land l_{c_1} = l_{c_2}) \Rightarrow Q_{c_1} \leq Q_{c_2} \tag{6}$$

  where Q_c is the amount of packages delivered by the courier c.

- By imposing an ordering on indexes of the assigned packages:

$$\forall c_1, c_2 \in \{1, \dots, m\} : (c_1 < c_2 \land l_{c_1} = l_{c_2}) \Rightarrow A_{c_1} <_{\texttt{lex}} A_{c_2} \tag{7}$$

  where A_c is a vector containing the packages delivered by the courier c.

Moreover, we experimented with a stronger version of these constraints that is applied between two couriers that satisfy the following condition:

$$\max\{L_{c_1}, L_{c_2}\} \leq \min\{l_{c_1}, l_{c_2}\} \tag{8}$$

where L_c is the actual load carried by the courier c.

2.4 Validation

2.4.1 Experimental design

We experimented variations of our models using as solvers Gecode, Chuffed, and OR-Tools. The search order for all models assigns the packages (`assignments`) first and then searches for the route (`path`) of each courier in decreasing order of capacity. Regarding search, we experimented several combinations of variable selection and assignment strategies on a subset of instances and used only the best ones to obtain the final complete results.

All experiments use the same random seed and were run as workflows on GitHub Actions which provides two cores at 2.45 GHz and 7 GB of memory, which we software limited to 5 GB to guarantee a safe margin for the Docker container to run.

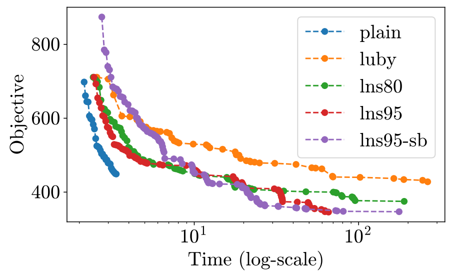
2.4.2 Experimental results

From some preliminary experiments, we observed that first fail and largest domain with weighted degree (`dom_w_deg`) are the best performing search strategies. Therefore, we performed the full experiments using as search strategy `dow_w_deg` for `assignments` and `first_fail` for `path`, respectively. Moreover, between the two symmetry breaking constraints, we observed that the lexicographic ordering defined in Equation (7) has better performances and therefore present only those results. The objective values found by the most significant models are reported in Table 1.

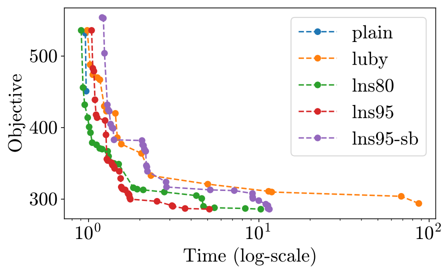
Table 1: Selected subset of CP results. Results in **bold** are solved to optimality. Instances that are all solved to optimality have been omitted.

	Gecode						Chuffed			OR-Tools		
Id	plain	luby	lns80	lns95	lns95 + SB (7)	lns95 + SB (7)(8)	plain	luby	SB (7)	plain	plain + FS	SB (7) + FS
7	201	**167**	**167**	**167**	**167**	**167**	**167**	**167**	**167**	408	**167**	**167**
9	**436**	**436**	**436**	**436**	**436**	**436**	**436**	**436**	**436**	617	**436**	**436**
11	594	597	528	490	503	–	963	–	756	1669	1189	1081
12	449	428	375	**346**	348	–	833	1061	785	1605	706	899
13	648	704	656	616	610	624	1126	914	1126	1734	584	746
14	725	972	794	715	792	–	1449	–	1089	–	–	–
15	659	901	765	738	803	–	1292	–	–	–	–	–
16	451	294	**286**	**286**	**286**	–	487	636	694	998	467	363
17	1324	1468	1119	1076	1155	–	–	–	–	–	–	–
18	691	806	675	662	620	–	1321	–	1779	–	–	–
19	554	412	336	**334**	**334**	–	795	1079	765	1521	623	577
20	1139	1369	1104	1068	1075	–	–	–	–	–	–	–
21	779	687	600	516	529	–	2104	2140	993	2115	1226	–

For Gecode, we experimented different search approaches by testing restart methods, large neighborhood search (LNS), and varying symmetry breaking constraints. We observed that, compared to a depth-first approach, by simply restarting search following the Luby sequence there is a mixed impact on the results with both positive and negative effects. Instead, by also using LNS, results tend to improve globally and we observed that a higher retain probability, with a peak at around 95%, works better. Regarding symmetry breaking constraints, we observed that they tend to worsen the final objective value in the majority of the cases, most reasonably due to the fact that the overhead to impose them is higher than the speed-up in search. For a visual comparison, we show in Figure 1 the evolution of the objective value during search. It can be seen that without restart the objective stops improving in the early stages of search as it most likely gets "stuck" in a branch of the search tree. By introducing some non-determinism, the objective reaches lower values, with LNS being the fastest and best performing.



(a) Instance 12



(b) Instance 16

Figure 1: Intermediate solutions using Gecode

For Chuffed, we only experimented with restarts and symmetry breaking constraints as LNS is not available through MiniZinc. Moreover, as indicated in the documentation[1], we tried enabling free search mode but only obtained worse results. Compared to Gecode, results on bigger instances are generally all worse, while smaller instances tend to be solved faster by Chuffed. As in Gecode, restarts, which we only experimented with random variable selection as it is the only non-deterministic search strategy available, and symmetry breaking constraints both yield mixed effects on the final result.

Finally, for OR-Tools, we conducted fewer experiments as it supports fewer MiniZinc annotations. In this case, we observed that enabling free search mode allows obtaining better results and, similarly to Gecode and Chuffed, symmetry breaking constraints have mixed results. As a side note, we must also observe that, as suggested in the documentation, these results are worse as OR-Tools performs better with multi-threading.

[1] `https://docs.minizinc.dev/en/stable/solvers.html`

3 SAT model

3.1 Decision variables

3.2 Objective function

3.3 Constraints

3.4 Validation

4 SMT model

4.1 Decision variables

4.2 Objective function

4.3 Constraints

4.4 Validation

5 MIP model

5.1 Decision variables

5.2 Objective function

5.3 Constraints

5.4 Validation

References

[1] Bruno De Backer et al. "Solving Vehicle Routing Problems Using Constraint Programming and Metaheuristics". In: *Journal of Heuristics* 6.4 (2000), pp. 501–523. ISSN: 1572-9397. DOI: 10.1023/A:1009621410177.

[2] Martin Desrochers and Gilbert Laporte. "Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints". In: *Operations Research Letters* 10.1 (1991), pp. 27–36. ISSN: 0167-6377. DOI: 10.1016/0167-6377(91)90083-2.