Multiple Couriers Problem

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

1 Introduction

The problem of this project is known in the literature as the capacitated vehicle routing problem and can be easily proven to be NP-hard. We tackle this problem by virtually splitting it into two sub-problems: first we look for an assignment of the items to the couriers and then search for the routes of each of them (i.e., by solving multiple travelling salesman problems). To solve the latter, we follow the approach presented in [**vrp**] where the route of a courier is modelled through the variables $P_d \in [1, n+1]$ with $d \in [1, n+1]$ defined as follows:

$$P_{d_1} = \begin{cases} d_2 & \text{iff the location } d_2 \neq d_1 \text{ is visited immediately after } d_1 \\ d_1 & \text{iff } d_1 \text{ is not part of the route of the courier} \end{cases} \tag{1}$$

With proper assignment and subtour elimination constraints, P_d allows to define a Hamiltonian cycle that passes through the items that the courier delivers and the solution can be extracted by traversing the cycle starting from the depot $n+1$.

The lower-bound of the objective function is common to all models and is defined as the maximum path cost that involves a single package:

$$\max_{p \in [1,n]} \{D_{n+1,p} + D_{p,n+1}\} \tag{2}$$

As upper-bound, also common to all models, we observed that it does not provide any improvement to the results. Nevertheless, we defined it as:

$$\sum_{d_1 \in [1,n+1]} \max_{d_2 \in [1,n+1]} D_{d_1,d_2} \tag{3}$$

Symmetry breaking constraints are also common to all models. By considering couriers with the same capacity, the following constraints can be used to avoid symmetries:

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

$$\forall c_1, c_2 \in [1, m] : (c_1 < c_2 \wedge l_{c_1} = l_{c_2}) \Rightarrow Q_{c_1} \leq Q_{c_2} \tag{4}$$

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

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

$$\forall c_1, c_2 \in [1, m] : (c_1 < c_2 \wedge l_{c_1} = l_{c_2}) \Rightarrow A_{c_1} <_{\texttt{lex}} A_{c_2} \tag{5}$$

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

As the triangle inequality holds, we also identified an implied constraint that consists of imposing that each courier delivers at least a package (a short proof is provided in **??**):

$$\forall c \in [1, m] : Q_c \geq 1 \tag{6}$$

where Q_c is the amount of packages delivered by the courier c. This obviously is applicable only if each courier is able to carry at least an item.

All experiments were done using 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. To guarantee a safe margin for the Docker container to run, the actual usable memory was capped to 5 GB.

The work has been completed in approximately one month and has been roughly split in the following way: Xia did the CP part, Costa worked on SAT, Domeniconi did the SMT models, and Maiolino completed the MIP part. The main difficulties we encountered are the following: (i) lack of proper documentation for many tools we used, (ii) difficulties to find a more efficient way to solve bigger instances, (iii) the need of time to run the experiments on all instances.

2 CP model

2.1 Decision variables

The CP model relies on the following decision variables:

- For each package p, $A_p \in [1, m]$ (`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, $P_{c,d}$ (`path[c, d]` in MiniZinc) is defined following **??**.

2.2 Objective function

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

$$\max_{c \in [1,m]} \sum_{\substack{d \in [1,n+1]: \\ P_{c,d} \neq d}} D_{d,P_{c,d}} \tag{7}$$

2.3 Constraints

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

$$\forall c \in [1, m] : \sum_{p \in [1,n]: A_p = c} s_p \leq l_c \tag{8}$$

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:

$$\forall c \in [1, m] : \begin{cases} P_{c,n+1} = n + 1 & \text{if } \nexists p \in [1, n] : A_p = c \\ P_{c,n+1} \neq n + 1 & \text{if } \exists p \in [1, n] : A_p = c \end{cases} \tag{9}$$

$$\forall c \in [1, m], \forall p \in [1, n] : \begin{cases} P_{c,p} = p & \text{if } A_p \neq c \\ P_{c,p} \neq p & \text{if } A_p = c \end{cases} \tag{10}$$

The constraint defined in **??** imposes that, for each courier, the depot has a successor only if that courier delivers at least a package. On the same note, **??** 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. In MiniZinc, this can be easily modelled by using the global constraint `subcircuit`.

2.3.1 Symmetry breaking constraints

For CP, we experimented with both symmetry breaking constraints defined in **????**. Moreover, we experimented with a stronger version of these constraints that is applied between two couriers whose actual loads are interchangeable. This is defined by the following condition:

$$\max \{L_{c_1}, L_{c_2}\} \leq \min \{l_{c_1}, l_{c_2}\} \tag{11}$$

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 only used the best ones to obtain the final complete results.

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 `dom_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 **??** has better performances and therefore present only those results. The objective values found by the most significant models are reported in **??**.

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 (??)	lns95 + SB (??)(??)	plain	luby	SB (??)	plain	plain + FS	SB (??) + 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 **??** 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.

3 SAT model

3.1 Decision variables

For SAT, we defined two different models:

Unified model based on the definition of the two decision variables A (`assignments` in Z3) and P (`paths` in Z3) as defined in **??**.

Matrix model based on the definition of a single matrix X such that $X[c,\ p,\ k] = 1$ iff courier c delivers item p as its k-th package.

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

Our discussion will be focused on the former to maintain coherence with the models defined in the other methods and obtain more comparable results. Its decision variables are the following:

- A is an $n \times m$ matrix such that $A[p,\ c] = 1$ iff courier c delivers item p.
- P is an $m \times (n+1) \times (n+1)$ matrix such that $P[c,\ loc_1,\ loc_2] = 1$ iff courier c moves from location loc_1 to loc_2.
- U is an $m \times n \times n$ matrix to implement MTZ subtour elimination [**mtz˙subtour**]. $U[c,\ p,\ k] = 1$ iff for the courier c the item p is delivered as the k-th.

3.2 Objective function

The objective function is computed as follows:

$$\max_{c \in [1,m]} \sum_{loc_1=1}^{n+1} \sum_{\substack{loc_2=1, \\ loc_2 \neq loc_1}}^{n+1} \texttt{D}[loc_1,\ loc_2] \cdot P[c,\ loc_1,\ loc_2] \tag{12}$$

3.3 Constraints

Assignment related constraints

- Capacity constraint:
  - The sum of the size of the items delivered by a courier must be within its load limit:

$$\forall c \in [1,m] : \sum_{p=1}^{n} A[p,\ c] \cdot s[p] \leq l[c] \tag{13}$$

  - Each item must be delivered by only one courier:

$$\forall p \in [1,n], \exists! c \in [1,m] : A[p,\ c] = 1 \tag{14}$$

Path related constraints

- General path constraints
  - If a courier delivers at least one package, there must exist a destination from $\texttt{DEPOT} = n+1$ with its value set to true, else it stays in $\texttt{DEPOT}$:

$$\forall c \in [1,m], \forall p \in [1,n] : \begin{cases} P[c,\ \texttt{DEPOT},\ \texttt{DEPOT}] = 0 & \text{if } \sum_{p=1}^{n} A[p,\ c] \geq 1 \\ P[c,\ \texttt{DEPOT},\ \texttt{DEPOT}] = 1 & \text{if } \sum_{p=1}^{n} A[p,\ c] = 0 \end{cases} \tag{15}$$

- If a courier delivers item p, its successor in P must be different from p:

$$\forall c \in [1,m], \forall p \in [1,n] : \begin{cases} P\texttt{[}c\texttt{, }p\texttt{, }p\texttt{]} = 0 & \text{if } A\texttt{[}p\texttt{, }c\texttt{]} = 1 \\ P\texttt{[}c\texttt{, }p\texttt{, }p\texttt{]} = 1 & \text{if } A\texttt{[}p\texttt{, }c\texttt{]} = 0 \end{cases} \tag{16}$$

- For each courier, there is exactly one successor location:

$$\forall c \in [1,m], \forall loc_1 \in [1,n+1] : \quad \sum_{loc_2=1}^{n+1} P\texttt{[}c\texttt{, }loc_1\texttt{, }loc_2\texttt{]} = 1 \tag{17}$$

- For each courier, there is exactly one predecessor location:

$$\forall c \in [1,m], \forall loc_2 \in [1,n+1] : \quad \sum_{loc_1=1}^{n+1} P\texttt{[}c\texttt{, }loc_1\texttt{, }loc_2\texttt{]} = 1 \tag{18}$$

- MTZ subtour elimination constraints
  - U relative to the first item p of the path of a courier c must be 1:

$$\forall c \in [1,m], \forall p \in [1,n] : P\texttt{[}c\texttt{, DEPOT, }p\texttt{]} = 1 \Rightarrow U\texttt{[}c\texttt{, }p\texttt{, 1]} = 1 \tag{19}$$

  - U relative to an item j delivered after i should be greater ($U_j \geq U_i + 1$):

$$\forall c \in [1,m],\ \forall i,j,k \in [1,n] : (P\texttt{[}c\texttt{, }i\texttt{, }j\texttt{]} \wedge U\texttt{[}c\texttt{, }i\texttt{, }k\texttt{]}) \Rightarrow \textstyle\sum_{l=k+1}^{n} U\texttt{[}c\texttt{, }j\texttt{, }l\texttt{]} = 1 \tag{20}$$

  - U relative to an item p of a courier must have exactly one value associated:

$$\forall c \in [1,m], \forall p \in [1,n] : \sum_{k=1}^{n} U\texttt{[}c\texttt{, }p\texttt{, }k\texttt{]} = 1 \tag{21}$$

  - MTZ formulation:

$$\begin{aligned} &\forall c \in [1,m],\ \forall i,j,k_1,k_2 \in [1,n] : \\ &(U\texttt{[}c\texttt{, }i\texttt{, }k_1\texttt{]} \wedge U\texttt{[}c\texttt{, }j\texttt{, }k_2\texttt{]}) \Rightarrow (k_1 - k_2 + 1) \leq (n-1) \cdot (1 - P\texttt{[}c\texttt{, }i\texttt{, }j\texttt{]}) \end{aligned} \tag{22}$$

3.3.1 Symmetry breaking constraint

A possible way to reduce tree exploration is to introduce symmetry breaking constraints. In particular, for SAT, we experimented only with the one defined in **??**.

3.4 Validation

3.4.1 Experimental design

Some modifications were applied to the basic model in order to visualize possible differences in performance. The original Unified Model was modified in three different versions: (i) with Symmetry Breaking Constraints, (ii) with Cumulative Constraint Application, (iii) with the Heule Encoding for `at_most_one()`.

To optimize the objective function, we follow the standard approach of iteratively solving the problem and imposing as new constraint the fact that a new objective must be lower than the one already found.

3.4.2 Experimental results

As we can notice from **??**, the performances were not very much different from the basic model. Only for the first 10 instances it has been possible to reach at least a suboptimal solution, while for the remaining ones the construction of the model required too much time causing to exceed the timeout limit. Symmetry breaking also did not contribute to improve the results.

Table 2: SAT results. Results in **bold** are solved to optimality. Instances that are all without a solution have been omitted.

Id	un-model	un-symm-model	un-cum-constr-model	un-heule-enc-model	matrix-model
1	**14**	**14**	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**	**226**
3	**12**	**12**	**12**	**12**	**12**
4	**220**	**220**	**220**	**220**	**220**
5	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**	**322**
7	232	238	222	296	292
8	**186**	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**	**244**

For all models, some statistics were extracted and are presented in **??**: (i) *time*: there are similar intervals of time for all unified model variants, while the matrix model ensures in some instances a better performance; (ii) *number of conflicts*: it useful for estimating the size of the search space (the lower the better)[2]. We can observe that the matrix model manages to have fewer conflicts, which is in line with its resolution time; (iii) *max memory*: also in line with the previous observations, the matrix model tend to require less memory to solve the problem.

[2] `https://stackoverflow.com/questions/17856574/how-to-interpret-statistics-z3`

4 SMT model

4.1 Decision variables

The SMT model uses the logic of quantifier-free linear integer arithmetic (`QF_LIA`) and relies on the following decision variables:

- For each package j, $A_j \in [1, m]$ (`ASSIGNMENTS[j]` in the code) indicates which courier delivers package j where $A_j = i$ indicates that courier i delivers package j.
- For each courier i, $P_{i,d} \in [1, n+1]$ for $d \in [1, n+1]$ (`PATH[i][d]` in the code) follows the definition of **??**.

4.2 Objective function

The objective function is defined as follows:

$$\max_{i \in [1,m]} \texttt{DISTANCES}_i$$

where $\texttt{DISTANCES}_i$ is equal to the distance traveled by each courier.

4.3 Constraints

- Weight constraint:

$$\forall i \in [1, m] : \sum_{j \in [1,n] : A_j = i} s_j \leq l_i \tag{23}$$

- Assignment and path constraint:

$$\forall i \in [1, m], \forall j \in [1, n] : \quad A_j = i \iff P_{i,j} \neq j \tag{24}$$

$$\forall i \in [1, m], \forall j \in [1, n] : \quad A_j \neq i \iff P_{i,j} = j \tag{25}$$

- All the elements of each row of P should be distinct:

$$\forall i \in [1, m] : \quad P_{i,j_1} \neq P_{i,j_2} \quad \forall j_1, j_2 \in [1, n+1], j_1 \neq j_2 \tag{26}$$

- Subcircuit constraint: each row P_i should define a subcircuit, that is a Hamiltonian path that ignores all the elements $P_{i,j} = j$, namely the packages that the courier i do not deliver. This can be modelled through MTZ subtour elimination as defined in **??**.

4.3.1 Implied constraints

For SMT, we experimented the implied constraint defined in **??**.

4.3.2 Symmetry breaking constraints

For SMT, we experiment both symmetry breaking approaches as presented in ????.

4.4 Validation

4.4.1 Experimental design

The experimental setup consists of two steps: first, we developed a Python package to automate the generation of SMT-LIB code from a high-level interface, allowing us to easily experiment and compare different solvers. More specifically, as solvers we experimented with Z3, cvc5, OpenSMT, SMTInterpol, and Yices 2 using both linear and binary optimization approaches. Then, to improve the performances on larger instances, we experimented with two different search strategies using the Z3 Python library (`z3py`):

Two solvers approach As SMT solvers do not allow to assign a priority to the variables, this approach attempts to guide the exploration of the search space by alternating two solvers: the first one finds A and the second one finds P given A. In other words, the former decides which courier delivers which package and the latter decides the route taken by each courier, basically solving m different travelling salesman problems.

Local search approach Similarly to the previous one, this approach also uses two solvers to first find A and then P. However, instead of letting the second solver find an optimal solution for P on its own, it is manually guided by performing a local search starting from a trivial solution (i.e., a path that delivers the items ordered by index).

4.4.2 Experimental results

The results of our experiments are presented in **??**. As linear optimization always outperformed binary search, we only present the former. Analyzing SMT-LIB results, we can observe that all five solvers perform more or less similarly. The best performing is Yices 2 which solves `QF_LIA` logic based on the simplex algorithm [**yices2**]. On the other hand, the worst performing is SMTInterpol which relies on Craig interpolation [**smtinterpol**]. Furthermore, experiments in `z3py` show that symmetry breaking and implied constraints do not provide significant contribution in improving the results.

By analyzing the two search approaches, we can observe that using two separate solvers have a negligible impact on the final results with mixed effects when using symmetry breaking constraints. Instead, local search enables the model to find a solution in a reasonable amount of time, even for the largest instances.

Table 3: SMT results. Results in **bold** are solved to optimality. Instances that are all solved to optimality have been omitted.

	SMT-LIB (plain)					z3py						
Id	Z3	cvc5	OpenSMT	SMTInterpol	Yices 2	plain	plain + SB	plain + IC	2 solvers	2 solvers + SB	2 solvers + IC	Local search
7	228	210	218	372	**167**	174	168	181	**167**	**167**	**167**	**167**
9	**436**	**436**	**436**	437	**436**	**436**	**436**	**436**	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	381	**244**	**244**	**244**	**244**	**244**	**244**	**244**	**244**
11	–	–	–	–	–	–	–	–	–	–	–	547
12	–	–	–	–	–	–	–	–	–	–	–	435
13	1446	–	–	–	1490	–	–	–	1812	1346	1832	632
14	–	–	–	–	–	–	–	–	–	–	–	1177
15	–	–	–	–	–	–	–	–	–	–	–	1140
16	–	–	–	–	1032	–	–	–	1510	1944	1861	303
17	–	–	–	–	–	–	–	–	–	–	–	1525
18	–	–	–	–	–	–	–	–	–	–	–	917
19	–	–	–	–	–	–	–	–	–	870	–	398
20	–	–	–	–	–	–	–	–	–	–	–	1378
21	–	–	–	–	–	–	–	–	–	–	–	648

5 MIP model

5.1 Decision variables

The MIP models also follow the same idea presented in **??**. The models are based on the following three variables:

- A binary tensor $X \in \{0,1\}^{(n+1)\times(n+1)\times m}$, where $X[i,j,k] = 1$ if and only if the courier k departs from the i-th delivery point and arrives at the j-th one. This formulation is inspired from the AMPL book [**AMPLbook**].
- A binary matrix $A \in \{0,1\}^{n\times m}$, where $A[i,k] = 1$ if and only if the package i is delivered by the courier k. We observe that these variables are not strictly necessary for the model, but they allowed us to write some constraints in an easier way.
- An auxiliary matrix $u \in \{1,\dots,n+1\}^{(n+1)\times m}$ that keeps track of the order in which the nodes are visited by each courier starting from 1. It is necessary, following the MTZ formulation, for subcircuit elimination. The interpretation is that, fixed a courier k, $u[i,k] < u[j,k]$ implies that the node i is visited before the node j by the courier k.

5.2 Objective function

As objective function, defined from the assignments as the maximum distance travelled by any courier, we use the following:

$$\max_{k \in 1 \dots m} \sum_{i=1}^{n+1} X[i,j,k] \cdot D[i,j] \tag{27}$$

5.3 Constraints

We defined the following constraints:

- Constraint to link the variables A and X:

$$\sum_{j=1}^{n+1} X[i,j,k] = A[i,k] \qquad \forall i \in 1 \dots n, \forall k \in 1 \dots m. \tag{28}$$

$$\sum_{i=1}^{n+1} X[i,j,k] = A[j,k] \qquad \forall j \in 1 \dots n, \forall k \in 1 \dots m. \tag{29}$$

- Constraint to guarantee that each package is assigned:

$$\sum_{k=1}^{m} A[i,k] = 1 \qquad \forall i \in 1 \dots n. \tag{30}$$

- Constraint for the capacity of each courier:

$$\sum_{i=1}^{n} A[i,k] s[i] \leq l[k] \qquad \forall k \in 1 \dots m. \tag{31}$$

- Constraint to avoid that each courier departs and arrives at the same point:

$$X[i,i,k] = 0 \qquad \forall i \in 1 \dots n+1, \forall k \in 1 \dots m. \tag{32}$$

- Constraint for the preservation of the flow (if one courier arrives at one node, he departs from the same one):

$$\sum_{i=1}^{n+1} X[i,j,k] = \sum_{i=1}^{n+1} X[j,i,k] \qquad \forall j \in 1 \dots n, \forall k \in 1 \dots m. \tag{33}$$

- Constraints to ensure that each courier starts and ends its route at the depot:

$$\sum_{j=1}^{n} X[n+1,j,k] = 1 \qquad \forall k \in 1 \dots m. \tag{34}$$

$$\sum_{j=1}^{n} X[j,n+1,k] = 1 \qquad \forall k \in 1 \dots m. \tag{35}$$

- Constraints for MTZ subtour elimination:
  - MTZ condition:

$$u[i,k] - u[j,k] + 1 \leq (n-1)(1 - X[i,j,k]) \qquad \forall i \in 1 \dots n, \forall j \in 1 \dots n+1, \forall k \in 1 \dots m. \tag{36}$$

  - $u[i,k] = 1$ iff i is the first item delivered by k:

$$u[i,k] \leq X[n+1,i,k] + (n+1)(1 - X[n+1,i,k]) \qquad \forall i \in 1 \dots n, \forall k \in 1 \dots m. \tag{37}$$

  - $u[j,k] \geq u[i,k] + 1$ iff i is delivered before j:

$$u[j,k] \geq (u[i,k] + 1)X[i,j,k] \qquad \forall i \in 1 \dots n, \forall j \in 1 \dots n+1, \forall k \in 1 \dots m. \tag{38}$$

5.3.1 Implied Model

For the implied model, we added one more constraint with the same meaning of **??**:

$$\sum_{i=1}^{n} A[i,k] \geq 1 \qquad \forall k \in 1 \dots m. \tag{39}$$

5.3.2 Symmetry Model

For the symmetry model, we added the symmetry breaking constraint related to the number of delivered packages as defined in **??**:

$$\sum_{i=1}^{n} A[i,k] \leq \sum_{i=1}^{n} A[i,j], \tag{40}$$

where $k, j \in 1, \dots, m$ with $k < j$ and $l[k] = l[j]$.

5.4 Validation

5.4.1 Experimental design

For the MIP models, we choose to use the solver-independent language AMPL. The workflow is based on the construction of three different models: the initial one, the implied one, and the symmetry breaking one.

For reproducibility, we decided to only use open-source solvers provided by the AMPL framework such as HiGHS, SCIP, and GCG.

5.4.2 Experimental results

Starting from some preliminary experiments, we immediately observed that GCG performs poorly on the first ten instances and therefore decided to discard it from the full experiments.

In **??**, we present the results of the MIP models. We can observe that the SCIP solver with the addition of implied and symmetry breaking constraints improves in performance. On the other hand, this behavior is not the same for HiGHS. Nevertheless, the HiGHS solver performances are in general significantly better than the SCIP ones.

Table 4: MIP results. Results in **bold** are solved to optimality. Instances that are all without a solution have been omitted.

Id	initial-scip	initial-highs	implied-scip	implied-highs	symmetry-scip	symmetry-highs
1	**14**	**14**	**14**	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**	**226**	**226**
3	**12**	**12**	**12**	**12**	**12**	**12**
4	**220**	**220**	**220**	**220**	**220**	**220**
5	**206**	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**	**322**	**322**
7	**167**	**167**	**167**	**167**	**167**	**167**
8	**186**	**186**	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**	**244**	**244**
13	642	726	616	694	526	692
16	–	**286**	–	557	–	320

For the first ten instances, we plotted the graphs about the execution time (in seconds), the number of simplex iterations, and branching nodes explored by these two solvers.

On a theoretical level, SCIP and HiGHS try to initially solve the relaxed-version of the problem (LP) using the revised simplex-method and finding a lower bound for the solution; then, if the solution found is not an integer, they start to solve the MIP part using branch-and-cut (SCIP) or branch-and-bound (HiGHS). For the resolution of the sub-problems generated by these two methods, they proceed recursively by applying the same algorithm. Therefore, from the plots, we can observe that a lower number of simplex iterations and branching nodes corresponds to a faster resolution time. This is in line with our previous observation regarding HiGHS as a better performing solver.

6 Conclusions

We experimented several models using different methods and attempted to implement the same core idea across the whole project to make results comparable.

Overall, all approaches are able to solve the smaller instances while, for bigger instances, only CP and SMT were able to at least produce a suboptimal solution by using proper search heuristics. Moreover, for this problem, we surprisingly noted that symmetry breaking constraints generally tend to, except in a few cases, worsen the results. To conclude, we can observe that, for this formulation of the problem, being able to guide the solver when exploring the search space is one of the most important factors to obtain good results.

A Implied constraint proof

Claim 1. *Assuming that the capacity of each courier allows delivering at least a package, if there exists an optimal solution, then there exists an optimal solution where each courier delivers at least one package.*

Proof. Let us assume that the optimal solution is D_j and there is a courier, say k_1, which do not deliver any package. Let us also suppose that the courier k_j is the one that covers the maximum distance D_j. If we assign one package that k_j brings, say i, to k_1, then, due to the triangle inequality, the two new distances D_1, travelled by the courier k_1 with i, and D_2, travelled by the courier k_j without i, are less or equal to D_j. In fact:

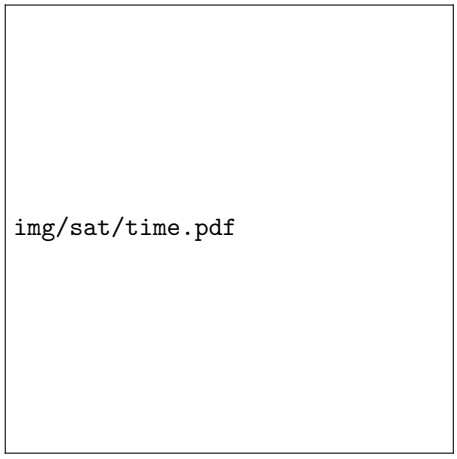
$$\begin{aligned} D_1 =& D[\texttt{depot}, i] + D[i, \texttt{depot}] \leq \\ & D[\texttt{depot}, i_1] + \cdots + D[i_r, i] + D[i, i_s] + \cdots + D[i_t, \texttt{depot}] = D_j. \end{aligned} \tag{41}$$

$$\begin{aligned} D_2 =& D[\texttt{depot}, i_1] + \cdots + D[i_r, i_s] + \cdots + D[i_t, \texttt{depot}] \leq \\ & D[\texttt{depot}, i_1] + \cdots + D[i_r, i] + D[i, i_s] + \cdots + D[i_t, \texttt{depot}] = D_j. \end{aligned} \tag{42}$$

Therefore, there are the following cases:

- If $D_1 = D_2$, either both k_1 and k_j cover the maximum distance or neither of them do, and another courier has a route of cost D_j.
- If $D_1 > D_2$, either k_1 covers the maximum distance or another courier that is not k_1 and k_j does.
- If $D_1 < D_2$, same as above for k_j.

So, an optimal solution still exists and has as objective value D_j. □



(a) Resolution time

(b) Number of conflicts

(c) Maximum amount of memory required

Figure 2: Statistics about the resolution of the problem for the first 10 instances

Figure 3: Compared statistics of the performances of the three models