Multiple Couriers Planning Problem

Bellatreccia Chiara, chiara.bellatreccia@studio.unibo.it
Fusa Edoardo, edoardo.fusa@studio.unibo.it
Quarta Angelo, angelo.quarta@studio.unibo.it

December 10, 2023

1 Introduction

This report is meant to describe the process and the results of solving the Multiple Couriers Planning (MCP) problem. Since this problem is a particular instance of a more general set of problems called Vehicle Routing Problems (VRP), we found that the ILP formulation is the most natural approach for this kind of problem.
In order to achieve solver independence, a solution based on python has been put in place. Said solution is organized in a modular fashion to call all the APIs of the employed solvers and modeling languages. So, we defined a pipeline that makes the user choose how many instances to run and which ones. At the end of the execution, each model returns the JSON file formatted in the required fashion as output.
The modeling phase of the process has been carried out by the group in its entirety. However, the development workload has been split among the members: Edoardo and Angelo worked mainly on CP and MIP models as well as the aspects related to software engineering. Chiara took care of SAT an SMT models.
The development process posed several challenges. One of the major difficulties was the lack of documentation in some open-source tools we employed, such as *pySMT*. Furthermore, that same library does not support a native timeout, therefore, we had to implement from scratch a way to stop the execution. The same issues was reported on SAT as well.
Overall those issues got an heavy impact on the time spent developing during the period right before handing the project. Concerning the developing time, it was stretched out on three months in which we did not consistently work on this particular project.

2 CP Model

In order to exploit the peculiar features of CP, this model has a different formulation with respect to the other ones. So, in order to use global constraints

dealing with graphs, the data in the *instances* directory have been transformed to match the new requirements in terms of model's inputs.
After the pre-processing stage, the parameters fed to the model are: the number of customers N, the amount of couriers available M, the number of edges connecting the nodes where the customers are located E, the capacities of the couriers l arranged in an array of size M and p which is the array taking account of the weights of each item to be carried. Then, for describing the graph we defined $from$, to and w that share the meaning of their indexes and unveil some of the symmetries of the problem.
In order to construct paths staring from the depot and able to do the round-trip, we add a new node that acts as an initial depot denoted by the index $N + 1$ and another one indexed as $N + 2$ that represents the final depot.
Defining this parameters and considering that the order of the arcs does not matter in terms of modeling, the first set of edges starting from the initial depot are placed at the beginning of $from$, to and w, that allows to exploit the symmetry of the problem.

2.1 Decision variables

The model relies on the subsequent decision variables which are:

(I) s and **t**: they are auxiliary variables and serve the purpose of setting the starting and the ending points of the paths produced by our CP model. Such paths are supposed to start from a depot and get back into it at the end of the journey, so they are set to the values $N+1$ and $N+2$ respectively that represent the depot.

(II) ns: this binary variable represents which nodes are visited by the couriers. The model retains this information through a MxN where M is the number of couriers and N is the number of nodes to be visited. Considering $i \in [1..M]$ and j in $[1..N]$, $ns[i][j] == 1$ iff the courier i serves the client j.

(III) es: es is a binary variable and similarly to ns points out the fact that a courier passes through an edge connecting two nodes. The information carried by this variable is stored in a MxE matrix where E is the set containing all the edges. For instance, having $i \in [1..M]$ and $j \in [1..E]$ $es[i][j] == 1$ iff the courier i pass through the edge j.

(IV) K: K represents the distribution of work load among the couriers. So, it is modeled as an array of size M where each element represents the sum of all the distances of the customers locations covered by a single courier.

(V) u: this integer auxiliary variable, which is modeled as an array of size M, accumulates the loads of all items' along the path crossed by a given courier.

(VII) objective: this integer variable stores the information about the objective function.

2.2 Objective function

In order to assure the fairness in terms of work carried out by each courier, the cost function of this optimization process consists of minimizing the maximum value of the vector K. The values in such vector are set by a specific constraints.

2.3 Constraints

In order to serve their purpose, auxiliary variables are set to their expected values using a set of constraints. That's the case for s and t which get the values $N+1$ and $N+2$ respectively. In such a way, each path crossed by the couriers is forced to start and terminate in the depot.
Similarly, u is set such that $u[j] = \sum_{i=1}^{N-2} ns[j][i]p[i]$.
On the other hand, the model sets other constraints that deal with the peculiar aspects of MCP. Since each courier has to perform a round-trip that must not intersect other paths assigned to other couriers, the model constraints the variable ns to follow the subsequent rule: $\sum_{j=1}^{M} ns[j][i] = 1 \ \forall i \in [1..N-2]$. That ensures each node to be visited only once.
To enforce the previous constraint and make sure that the paths associated to each courier are actually a round-trip arranged in a directed fashion, the model ingrains a global constraint called *bounded_path* [min] that works as follows:

$$bounded\_dpath(N, E, from, to, w, s, t, row(ns, j), row(es, j), K[j])$$

This global constraint looks for a directed path from the node s to the node t while taking account of the architecture of the original graph given by the parameters $from$, to and w. Furthermore, this global constraint influences the value of the objective function by assigning to $K[j]$ the sum of the distances that a courier j covers. At the same time, it sets the variables ns and es, therefore it chooses the path assignment to the couriers.
Then, the developed model makes sure that $\forall i \in [1..N] \ u[i] \leq l[i]$ in order to cap the maximum value of the accumulator u with the maximum feasible values given by l. Finally, in order to speed up the solving process, the model reduces the search space by imposing a lower and an upper bound. The first one gives a noticeable contribution to the running time and it consists of computing the longest round-trip starting from the depot and consisting of visiting a single node and using it to cap the objective function on the lower side: $\max_k K \geq lb$ where lb represents the lower bound.
Similarly, the model makes sure that the same value does not exceed a given threshold. The algorithm that computes the upper bound follows a greedy approach based on collecting the first $M-1$ longest round-trips of the same type as the lower bound ones. Then, the courier will greedily build a round-trip taking the shortest edge available at each step until it comes back to the depot and accumulating the total distance during the process. So, the model takes the maximum among all those distances called ub as an upper bound: $\max_k K \leq ub$.

2.3.1 Symmetry breaking constraints

Considering that the MCP presents some symmetries in terms of couriers loads, the model has been developed to exploit them. The basis of our approach is the application of lexicographic order: when two or more couriers can perform the

same job, the model chooses the one which the row in es have the least index that corresponds to the courier. However, this constraint does not apply to all the cases so, the model imposes that $\max(u[i], u[j]) \leq \min(l[i], l[j])$ to make sure that it shoots when needed. Furthermore, we consider just the indexes i such that $i < j$, so we do not take reverted indexes repetitions into account.
All that is feasible because of the pre-processing phase. We made sure that the sequence of the edges mapped into $from$ and to starts with the edges leading out of the depot.

2.4 Validation

2.4.1 Experimental design

Once having the CP model, it got developed using Minizinc exploiting its main feature of being solver independent. Then, we run the experiments on different solvers, which are *Gecode* and *Chuffed*.
The Minizinc model is stored in a *mzn* file imported in a python function that runs it in different setups. In addition to that, all the instance data handed to us have been turned into *dzn* files read by the same python function.
In order to get the routes crossed by all the couriers the script translates the variable es, so they fit the required structure for creating the JSON file.

2.4.2 Experimental results

The table 1 displays the result retrieved by running CP model. Evidently, the model performs the best on the first ten instances where finding an optimal solution is a consistent behaviour no matter the solver or the symmetry breaking constraints applied. On the contrary, finding even a sub-optimal solution for the rest of instances represent a hard challenge for the model. As a matter of fact, the missing entries in the table refer to instances too difficult to be solved by this CP model[1].
Such fact is clear by analysing the results obtained on the instances 12, 13, 16 and 19: even if a couple of solution are available, they are never optimal. Some value insights can be collected by considering the running time of each configurations, so, figure 8 and figure 2 depicts this other point of view on the outcomes that are classified as: *optimal solution*, denoted by the blue and green scatters, is the only class that allows values from 0 to 300. *Timeout*, denoted by the orange scatters and *no solution*, denoted by the red spots. The last two classes, conversely, will be portrayed as zeros for visual reasons. Comparing the plots in figure 8, it emerges that the application of the symmetry breaking constraint degrades the performance of the model in terms of running time. Moreover, the number of instances without a solution among the difficult ones raises.
Although the changing of the solver, watching at figure 2 the same result holds. In this case, the running time reported by the plots are comparable but the

[1]The same holds for all the other tables

ID	Gecode	Gecode+SB	Chuffed	Chuffed+SB
1	**14**	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**
3	**12**	**12**	**12**	**12**
4	**220**	**220**	**220**	**220**
5	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**
7	**167**	**167**	**167**	N/A
8	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**
12	473	N/A	N/A	N/A
13	1218	1218	N/A	1220
16	318	N/A	N/A	N/A
19	N/A	449	N/A	N/A

Table 1: Experimental results of CP model, with different combinations of constraints.

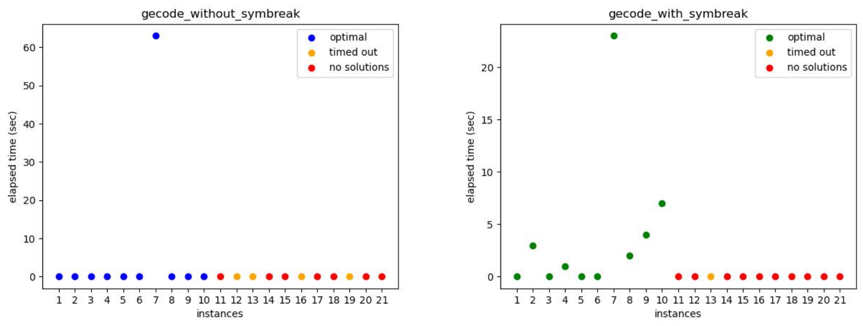

Figure 1: *gecode* solver based comparison between the baseline and the model that exploits the symmetries of the problem

behaviour of the instance number seven stands out. The model ingraining the knowledge the symmetries is not able to find a single solution by the timeout, while the baseline gets the optimal one spending almost 3 minutes.

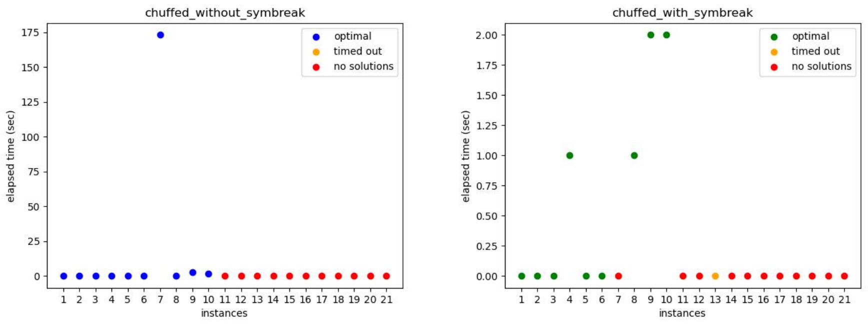

Figure 2: *chuffed* solver based comparison between the baseline and the model that exploits the symmetries of the problem

3 MIP Model

MIP model gets its structure from a model available at the following monograph [TV02] that presents many models with their own peculiarities solving several variations of VRP. Since there were not any for MCP as it is, we picked the closest formulation and modified accordingly to solve it.
MIP does not rely on global constraints as CP does. Then, for obvious reasons, the approach adopted for solving MCP is slightly different.
Differently from CP model, input data have not been heavily modified in order to be fed to the model In this context, we defined the following parameters: N is the number of customers, K stands for the number of couriers, C is an array containing the maximum capacity of each courier, L maps into an array of length N the weights of the items to be carried, finally, D is the $(\mathsf{N}+1)\mathsf{x}(\mathsf{N}+1)$ matrix of distances among the customers and the depot.

3.1 Decision variables

The model utilize the following variables to achieve the required behaviour:

(I) x: this is a binary variable arranged in a $(\mathsf{N}+1)\mathsf{x}(\mathsf{N}+1)\mathsf{x}\mathsf{K}$ tensor depicting which edges a given courier crosses. Being a binary variable, $x[i][j][k] = 1 \iff$ the courier k goes from the node i to the node j.

(II) y: y is a $(\mathsf{N}+1)\mathsf{x}\mathsf{K}$ binary matrix that represents the nodes visited by a certain courier on its columns. Therefore, if the courier k stops in the node denoted by $i \implies y[i][k] = 1$.

(III) u: This variable is defined as an array of size N and each entry i represents the load carried so far by courier that passes through node i.

3.2 Objective function

The objective function associated with this model takes account of the parameter D and the variable x:

$$\max_{k \in K} \sum_{i=0, j=0}^{N+1} D[i][j]x[i][j][k]$$

This is an alternative form for the objective function employed for the CP model and it assures a fair division of the workload in the same way.

3.3 Constraints

To impose the fact that a node is visited only once, the model ensures that $\sum_{k=1}^{K} y[i][k] = 1 \ \forall i \in [1..N]$ where is represent the indexes associated to the nodes. Concurrently, we wish that all couriers start from the depot so, $\sum_{k=1}^{K} y[N+1][k] = K$ holds.
Moreover, MIP model ingrains the fact that each edge is crossed just once through the constraints that connect in a consistent way the variables x and y.

$$\sum_{j=1}^{N+1} x[i][j][k] = \sum_{j=1}^{N+1} x[j][i][k] \ \forall i \in [1..N+1] \ \forall k \in [1..K]$$

$$\sum_{j=1}^{N+1} x[i][j][k] = y[i][k] \ \forall i \in [1..N+1] \ \forall k \in [1..K]$$

As MCP states, there is a maximum load that each courier can carry. So, our model imposes the fact that $\sum_{i=1}^{N} L[i]y[i][k] \leq C[k] \ \forall k \in [1..K]$ which means that a given courier can only choose a path in which the sum of all items to leave along it does not exceed its maximum capacity. A hard to address issue when dealing with multiple couriers is the sub-tour elimination[MTZ60]. In order to solve that, the model imposes that:

$$u[i] - u[j] + Nx[i][j][k] \leq N - 1 \quad \forall i, j \in [1..N] \ \forall k \in [1..K]$$

This particular constraint has been developed distilling the knowledge coming from the monograph [TV02].
Furthermore, another constraint has been modified to accommodate the fact that couriers in MCP formulation can have different capacities while the model suggested in the monograph was more restrictive in this matter: every courier has the same capacity. Therefore, the optimization process is subject to $\sum_{i=1}^{N} L[i]y[i][k] \leq C[k] \ \forall k \in [1..K]$.
As the CP model section describes, we bounded the search space in order to improve the computational performance of our MIP model using the same idea previously explained. So, the constraints stated by the model are: $\max_k \sum_{i,j} D[i][j]x[i][j][k] \geq lb$ that impose the lower bound and $\max_k \sum_{i,j} D[i][j]x[i][j][k] \leq ub$ that caps maximum value that the objective function can reach.

3.3.1 Symmetry breaking constraints

As discussed in previous sections, MCP exhibit some symmetries that can be exploited by imposing an ad-hoc constraint that states: $\max(\sum_i y[i][k]L[i], \sum_i y[i][m]L[i]) \leq \min(C[k], C[m]) \implies ((x[N+1][j][k] = 1) \implies \sum_{l=1}^{j} x[N+1][l][m] == 0))\ \forall k, m \in [1..K]\ s.t.\ k < m,\ j \in [1..N]$. Such constraint sorts the couriers using a lexicographic order on the last row of the tensor x such that if more then one courier can carry out a given job, the one being the least in the lexicographic ranking gets the job.

3.4 Validation

3.4.1 Experimental design

Among the plethora of tools available for building MIP models, we relied on *AMPL*, a modeling language that allows to use multiple solvers without modifying the model itself. Therefore, similarly to the approach put into action for CP model, we considered the following solvers: *gurobi*, *xpress* and *copt*. Then, we create new data files arranging all the information conveyed by the files stored into the *instances* directory.

3.4.2 Experimental results

Looking at table 2, we notice that other than the usual simple instances, the table reports more optimal solutions among the difficult ones. For example, the instance 16 has been solved in an optimal way twice with two solvers out of three. Moreover, our model managed to find a solution for instance 13 for all the different setups, although it is not optimal.
Watching at the results from another perspective, the running time of the most of the instances associated with an optimal solution is close to zero or even less. Overall, the results are consistent among the different solvers employed to carry the experiments.

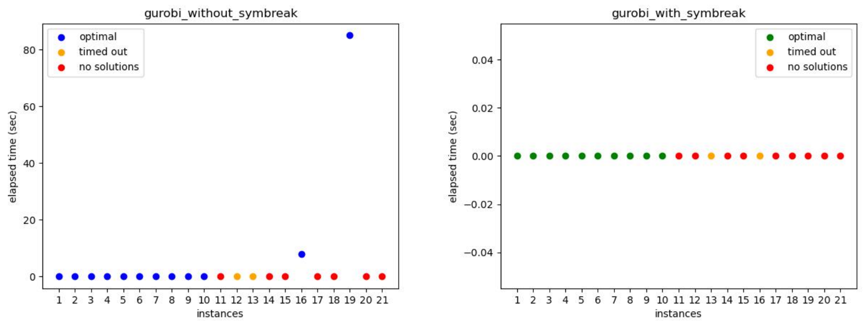

Figure 3: *gurobi* solver based comparison between the baseline and the model that exploits the symmetries of the problem

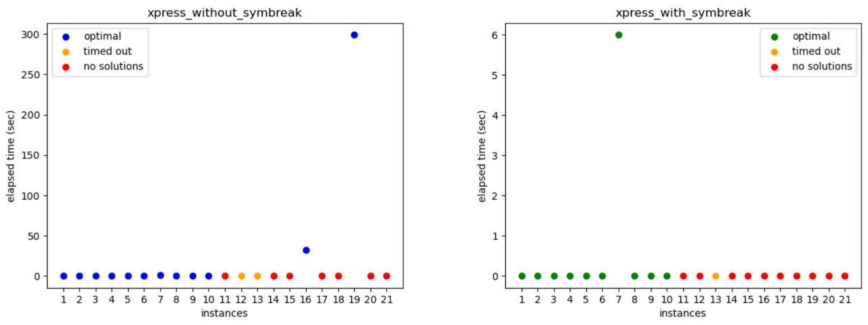

Figure 4: *xpress* solver based comparison between the baseline and the model that exploits the symmetries of the problem

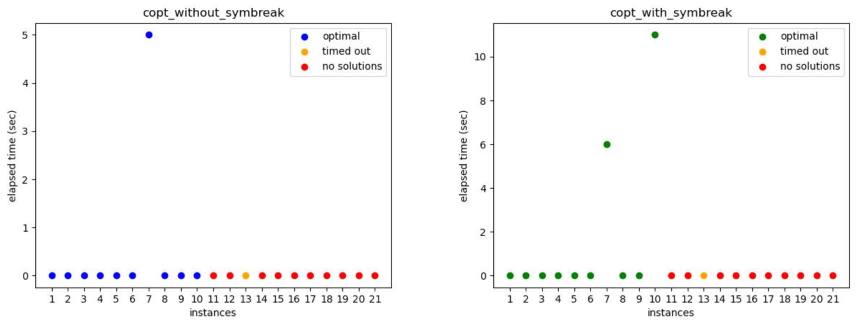

Figure 5: *copt* solver based comparison between the baseline and the model that exploits the symmetries of the problem

4 SAT Model

In the case of SAT, the SMT solver Z3 (available as a Python library) was used. This led to several considerations: first, **the non-binary variables and parameters must be represented as binary numbers.** For the parameters, this was achieved by implementing a Python function toBinary that outputs a list of booleans as the binary representation of the input number. Each variable was directly instantiated as a list of booleans of a well defined length. Second, **the predefined arithmetic operations of Z3 are not usable.** Indeed, they are not based on SAT solving. Thus, customized SAT-based arithmetic operations were developed: the *addition of binary numbers*, implemented in the Python function bin_add, which essentially implements a full adder, and in the function bin_multiadd, which recursively calls bin_add to implement the addition of a list of binary numbers; the *comparison of two (binary) numbers*, implemented in the Python functions greater_than and strictly_greater_than (given two binary numbers a and b, a is strictly greater than b iff the value of the first non-

ID	Gurobi	Gurobi+SB	xpress	xpress+SB	copt	copt+SB
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**
12	510	N/A	454	N/A	N/A	N/A
13	452	462	704	704	772	778
16	**286**	292	**286**	N/A	N/A	N/A
19	**334**	N/A	573	N/A	N/A	N/A

Table 2: Experimental results of MIP model, with different combinations of constraints.

identical bit of a with respect to b is True. a is greater or equal to b iff a is either strictly greater than b or a is equal to b); the *maximum element of a list*, implemented in the Python function max_of (given a (binary) number H_max and a list of (binary) numbers H, H_max is the maximum of H iff it is equal to at least one element of H and it is greater or equal to each element of H). Third, **the predefined AtLeastK, ExactlyK and AtMostK constraints of Z3 are not usable**, being not based on SAT solving. Thus, we implemented the same constraints in SAT with a sequential encoding. Finally, **the predefined minimization method of Z3 is not usable**, being once again not based on SAT solving. More details on our minimization method in SAT are available in Section (4.2).

4.1 Decision variables

We proceed to list all the literals in our SAT model and the respective semantics, recalling that each constant or variable that is not boolean was codified into a boolean list of length length.

(I) **Destination tensor (X matrix)**: refer to Section (3.1).

(II) **Customers matrix (Y matrix)**: refer to section (3.1).

(III) **Cumulative matrix (U matrix)**: an (N+1)xlength matrix which was used as an auxiliary variable in order to formalize the sub-tour constraints.

Indeed, the aim of this matrix is to establish an order between the destinations visited, so that two different destinations cannot be visited at the same time (i.e. so that sub-tours are not allowed). The U matrix contains $\mathsf{N{+}1}$ variables, while the second dimension of length is the bit-length of these variables.

(IV) **Cumulative distance matrix (H matrix)**: an auxiliary $\mathsf{mxlength}$ matrix (where m is the number of couriers), in which each of its m rows is the (binary encoding of) the total distance covered by the respective courier. This matrix was instantiated for optimization purposes (more below).

(V) **Maximum of H matrix (H_max):** the (binary encoding of) the maximum value of the H matrix.

4.2 Objective function

As mentioned before, the objective function to minimize is $\mathsf{H\_max}$:

$$\mathsf{H\_max} = \max_{k \in K} \mathsf{H}[k].$$

In our SAT model, the minimization was done using a while loop: while the constraints are still satisfiable (and, thus, a model is available), at each iteration we store the value of $\mathsf{H\_max}$ provided by the current model in the variable $\mathsf{H\_current}$. Then, we add the constraint that $\mathsf{H\_max}$ must be lower than $\mathsf{H\_current}$ by using the function $\mathsf{strictly\_greater\_than}$. When the model becomes unsatisfiable (because asking that $\mathsf{H\_max}$ is lower than $\mathsf{H\_current}$ becomes incompatible with the other constraints), the iterations stop and the provided $\mathsf{H\_max}$ value is the one of the previous iteration.

4.3 Constraints

(I) Constraints on the tensor X: (1) Each courier cannot go from one location to the same location: $\overline{\mathsf{X}[i][i][k]}$ for $i = 1...N+1$, $k = 1...m$. **(2)** If a courier visits a location, then it enters and exits the corresponding node. This was implemented by defining, given a courier k and a node i, the list $\mathsf{X\_exit_{ik}}$ of the entries of the X matrix that represent the exiting edges ($\mathsf{X}[i][j][k]$, $j = 1...N$) and the list $\mathsf{X\_enter_{ik}}$ of the entries of the X matrix that represent the entering edges ($\mathsf{X}[j][i][k]$, $j = 1...N$), and by adding the constraints: $\mathsf{Y}[i][k] \implies \mathsf{ExactlyOne}(\mathsf{X\_enter_{ik}}) \wedge \mathsf{ExactlyOne}(\mathsf{X\_exit_{ik}})$, $\overline{\mathsf{Y}}[i][k] \implies \overline{\mathsf{X\_enter_{ik}}[j] \wedge \mathsf{X\_exit_{ik}}[j]}, j = 1...N$ for every $i = 1...N$ and $k = 1...m$, where $\mathsf{ExactlyOne}$ refers to our customized sequential encoding of the constraint.

(II) Constraints on the matrix Y: (1) Each location is visited exactly once, meaning that (except for the depot), for every node i, exactly **one** item of the list $\mathsf{Y\_sum_i} = \mathsf{Y}[i][k]$, $k = 1...m$ must be True: $\mathsf{ExactlyOne}(\mathsf{Y\_sum}_i) = 1$, for every $i = 1...N$. **(2)** All the vechicles leave the depot, meaning that $\mathsf{Y[N][k]}$ must be True for every k.

(III) Constraints on the matrix U: (1) every row of the U matrix must be a representation of a binary number between 0 and N. This was assured using the function $\mathsf{greater\_than}$. **(2)** Anti sub-tours constraint: for every node

i, for every node j, if the edge between i and j is visited then it must hold that $\mathsf{U}[j] - \mathsf{U}[i] \geq 1$: $\mathsf{X}[i][j][k] \implies \mathsf{greater\_than}(\mathsf{U[j]}, \mathsf{var})$, where var is forced to be the sum of $\mathsf{U[i]}$ and 1.

(IV) Maximum load constraints for which the implementation was done by first computing, for every courier k, the list $\mathsf{sum\_w\_list_k}$ such that $\mathsf{Y}[i][k] \implies \mathsf{sum\_w\_list_k}[i] == w[i], \overline{\mathsf{Y}[i][k]} \implies \mathsf{sum\_w\_list_k}[i] == 0,\ i = 1...N{+}1$ (where $\mathsf{w}[i]$ is the weight of the node i, given as input, and all the numbers are represented in binary). After that, for every courier k the variable $\mathsf{sum\_w_k}$ was constrained to be the sum of the elements in the list $\mathsf{sum\_w\_list_k}$ by using the function $\mathsf{bin\_multiadd}$. Finally, in order to ensure that the maximum capacity $\mathsf{l[k]}$ of each courier was not exceeded, the constraint $\mathsf{greater\_than}(\mathsf{l[k]}, \mathsf{sum\_w_k})$ was added.

(V) Constraints on the matrix H and H_max: in order to ensure that $\mathsf{H}[k]$ is the total distance traveled by the k-th courier, a list similar to the one for the maximum load constraints was created for each courier k. Then, $\mathsf{H}[k]$ was forced to be the binary addition of the former list. Finally, $\mathsf{H\_max}$ was forced to be the maximum value of H using the function $\mathsf{max\_of}$. The constraints for the lower and upper bound of $\mathsf{H\_max}$ were added using $\mathsf{greater\_than}$.

4.3.1 Symmetry breaking constraints

The symmetry breaking constraint is conceptually equivalent to the one used in CP (2.3.1), formalized in SAT as $\mathsf{greater\_than}(\mathsf{toBinary}(\mathsf{min}(\mathsf{l[i]}, \mathsf{l[j]})), \mathsf{max\_value}) \wedge \mathsf{X[N][t][i]} \implies \bigwedge_{o=1...t} \overline{\mathsf{X[N][o][j]}}$ with $i, j = 1...m, i < j,\ t = 1...N$ where $\mathsf{max\_value}$ is forced to be the maximum between the loads of the two couriers.

4.4 Validation

4.4.1 Experimental design

We used the solver Z3. In order to convert the instances format into the input format necessary to our model, the function converter in the script *converter.py* was used. The experimental results were diversified with respect to the use of the symmetry breaking constraint (SB).

4.4.2 Experimental results

The results show that (in terms of timing) SAT is our less efficient model. This is clearly caused by the large amount of constraints required by this kind of formulation, as a result of which both the process of minimizing and finding a feasible solution are slow. Another result worth observing is that the SB constraint does not really affect the timing (Table 3 and plots in Fig. (6)).

ID	Z3	Z3 + SB	ID	Z3	Z3 + SB	ID	Z3	Z3 + SB
1	**14**	**14**	5	**206**	**206**	9	**436**	**436**
2	**226**	**226**	6	**322**	**322**	10	**244**	**244**
3	**12**	**12**	7	251	279	13	N/A	1208
4	**220**	**220**	8	**186**	**186**			

Table 3: Results of SAT model, with and without the SB constraint.

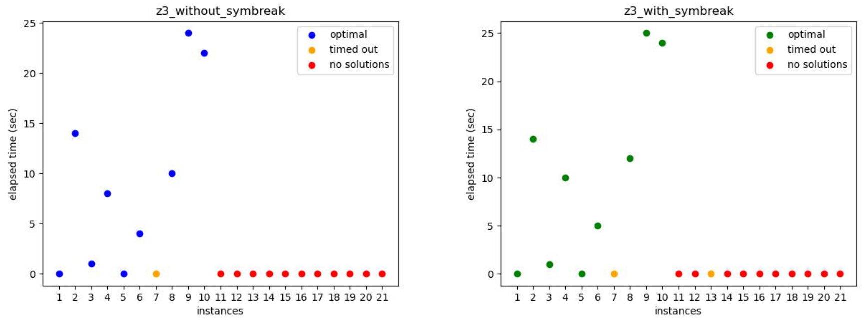

Figure 6: Timing results of SAT model, with and without the SB constraint.

5 SMT Model

5.1 Decision variables

As for the previous SAT model, m is the number of couriers and N is the number of destinations. The tensor X and the matrix Y are exactly identical to the ones used in SAT. The variables U and H are now vectors of length respectively equal to N+1 and m, where each of their entries is an int number. Finally, H_max the maximum value of the H vector.

5.2 Objective function

The objective function to minimize is still H_max, defined in our model as:

$$\mathsf{H\_max} = \max_{k=1\ldots m} \mathsf{H}[k].$$

Because *pySMT* does not provide any built-in minimize function, the minimization method was implemented in the same way as the SAT model.

5.3 Constraints

The constraints on the tensor X and the matrix Y are identical to the SAT model, with the only difference that the function ExactlyOne used in the constraints was the predefined one by *pySMT*. The constraints on the vector U and the *maximum load constraints* are also the same as in SAT, but the functions LE, GE and Minus by *pySMT* were used instead of $\mathsf{greater\_than}$ and $\mathsf{bin\_add}$. Every entry of H ($\mathsf{H}[k]$) was forced to be equal to $\sum_{i,j=1...N} \mathsf{X}[i][j][k] \cdot \mathsf{D}[i][j]$ using the functions Times and Plus by *pySMT*. Then, $\mathsf{H\_max}$ was forced to be the maximum value of H using the function Max by *pySMt*. The constraints for the lower and upper bound of $\mathsf{H\_max}$ were added using GE and LE.

5.3.1 Symmetry breaking constraints

Conceptually equivalent to Section (2.3.1), formalized in SMT as
$\mathsf{Max}(\sum_{i=1...N} \mathsf{Y}[i][k]\mathsf{w}[i], \sum_{i=1...N} \mathsf{Y}[i][s]\mathsf{w}[i]) \leq \mathsf{Min}(\mathsf{l[k]}, \mathsf{l[s]}) \wedge \mathsf{X}[N][t][k] \implies \bigwedge_{o=1...t} \overline{X[N][o][s]}$ with $k, s = 1...m, k < s, t = 1...N$.

5.4 Validation

5.4.1 Experimental design

We used the two solvers Z3 and *mathsat*, provided by the wrapper *pySMT*. The experimental results were diversified with respect to the use of different solvers and the use of the symmetry breaking constraint (SB).

5.4.2 Experimental results

The results show that in terms of timing our SMT model performs better than SAT; however, in terms of optimality of the solutions their performance is comparable. The results are affected by the fact that we wanted to achieve solver-independence, so we had to adapt the constraints to this specific situation. Furthermore, *msat* does not provide any built-in timer, so we were forced to build our own timer, which does not provide reliable timing results for instances from 11 to 21.

ID	Z3	Z3+SB	msat	msat+SB	ID	Z3	Z3+SB	msat	msat+SB
1	**14**	**14**	**14**	**14**	6	**322**	**322**	**322**	**322**
2	**226**	**226**	**226**	**226**	7	168	180	N/A	N/A
3	**12**	**12**	**12**	**12**	8	**186**	**186**	**186**	**186**
4	**220**	**220**	**220**	**220**	9	**436**	**436**	**436**	**436**
5	**206**	**206**	**206**	**206**	10	**244**	**244**	**244**	**244**
					13	N/A	1198	N/A	N/A

Table 4: Results obtained with our SMT model.

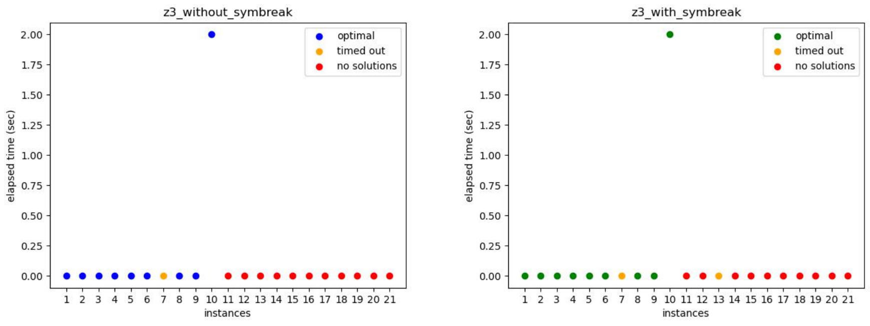

Figure 7: results with and without the SB constraint using Z3.

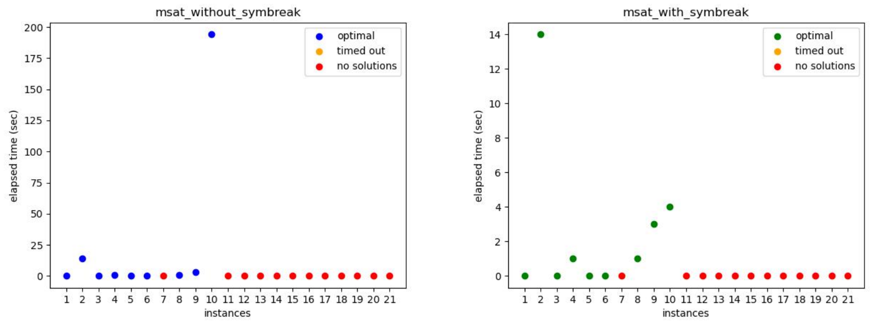

Figure 8: results with and without the SB constraint using *msat*.

6 Conclusions

The experiments revealed that the first ten instances, with the notable exception of the number seven, are relatively easy to solve even with open-source solvers. On the other hand, the rest of the instances require a not negligible amount of time due to the inherent complexity of the problem being NP-hard.
Considering the overall trend of the models' performance, MIP was the best approach in terms of efficiency but that could be due to the solvers employed that are professional tools.
As highlighted by the results, sometimes the symmetry breaking constraint is crucial to solve hard instances, while on other occasions, it degrades the performance of the considered solver. For instance, SAT and SMT models require many more constraints to carry out the same job of other models. As a result, the searching space shrinks but the amount of time needed to retrieve a single solution grows. Therefore, by the results we analysed, we can conclude that the second effect is more prominent than the first one. Finally, we noticed some peculiar behaviour of the MIP model concerning some hard instances for which the solver managed to find an optimal solution: the most probable reason why that happened is the role played by the solvers used for solving the problem.

References

[min] 4.2.2.11. graph constraints.

[MTZ60] Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

[TV02] Paolo Toth and Daniele Vigo. *The vehicle routing problem*. SIAM, 2002.