Multiple Couriers Planning

Lorenzo Venieri lorenzo.venieri2@studio.unibo.it
Luca Zucchini luca.zucchini6@studio.unibo.it

January 2023

1 Introduction

In this report we will cover our work to solve the Multiple Couriers Planning problem. This problem is a more complicated version of the Vehicle Routing Problem, since it introduces a constraint on the load of the vehicles. Finally, the problem is made even more difficult by diversifying the maximum load for each of the available vehicles. We managed to solve the problem using three different technologies: CP using MiniZinc, SAT using Z3, and MIP using Gurobi. To design and test the models, we used the same data contained in the eleven available instances. The only difference concerns the distance matrix that carries the information about distances between points: for both the MIP and SAT model it is calculated in the same file of the model, while for the CP model it is calculated in a different file and than passed manually to MiniZinc. The objective variable, for all the models, was set to be the overall traveled distance, to be minimized.
The workflow has been the following: Luca started working on the CP model while Lorenzo started working on the SAT one. Then, the first to finish his own work would have started working on the MIP model. After about 10 days we both finished our own model and so we started working on MIP, which we designed in about a week.

2 CP

We now describe the work that led to the implementation of the CP model in MiniZinc

2.1 Decision variables

MiniZinc is a powerful and flexible tool and we decided to take advantage of such features to implement a model slightly different from the one of SAT and MIP. We wanted to develop the system to have an array NEXT, where the value of the element $\text{NEXT}[i]$ was the item delivered after the item i. In such a way we could have had easy access to the whole tour of the couriers just by looking at

the array NEXT, and we could have set the constraints in an efficient way. The problem with this intuition was the fact that we had no way to disambiguate the various steps for the deposit. To better explain this issue, let's consider the case where the tour plan for two couriers is the following (7 = depot):

$$Courier1 : 7 => 2 => 3 => 1 => 7$$

$$Courier2 : 7 => 5 => 4 => 6 => 7$$

If we just consider point '7' as the depot, the NEXT array would become:

$$\text{NEXT} = [7, 3, 1, 6, 4, 7, ?]$$

As we can see there is no way to determine which is the Successor of point 7, because it is both 2 and 5.
For this reason, we "added" $2 * m$ points to let the model handle in a proper way every passage for the depot. In this way the set of POINTS has the range: $1..n + 2 * m$, where the first n points are the ones representing the items to be delivered and the remaining represent the depot points. After explaining this design choice we now define the used decision variables:

- NEXT : Array of domain and dimension $1..n+2*m$, the value of NEXT$[i]$ represents the point reached after the point i, whether it is a new delivery point or the depot

- PREVIOUS: Array of domain and dimension $1..n + 2 * m$, the value of PREVIOUS$[i]$ represents the point reached before the point i, whether it is a new delivery point or the depot.

- COURIER : Array of domain $1..m$ and dimension $1..n + 2 * m$, the value of COURIER$[i]$ represents the courier assigned to the point i

- LOAD: Array of domain $0..max(capacities)$ and dimension $1..n+2*m$, the value of LOAD$[i]$ represents the value of the load of the assigned courier after reaching point i

The declaration of both the arrays of variables NEXT and PREVIOUS is conceptually redundant but it helps with the definitions of some constraints and reduces the search space for some variables.

2.2 Objective function

As stated in the Introduction, the objective function to be minimized was set to be the overall distance traveled by the couriers. For the CP model, in order to calculate the total distance between points, we had to consider that we added some points (in particular $2*m$ points) that didn't have a representation in the

distance matrix. So we had to split the summation to count both the distances between two delivery points reached in succession, and the distances between a delivery point and the depot, in the case of the departure or arrival of the courier to the depot itself. Note that we have considered the depot to be the point $n+1$ given n items to deliver. The summation became:

$$Total\ distance = S_1 + S_2 + S_3$$

where:

$$S_1 = \sum_{i=1}^{n} Distance[previous[i], i] \qquad \text{with} \qquad \text{previous[i]} \leq \text{n}$$

to catch the path between two delivery points

$$S_2 = \sum_{i=1}^{n} Distance[n+1, i] \qquad \text{with} \qquad \text{previous[i]} > \text{n}$$

to catch the departure of a courier

$$S_3 = \sum_{i=1}^{n} Distance[i, n+1] \qquad \text{with} \qquad \text{next[i]} > \text{n}$$

to catch the arrival of a courier.

We also tested a different objective function to investigate whether we could increase the performances of our model. This objective function was modeled to catch for each pair of variables (i,NEXT[i]), how much their distance was greater than the distance from i to the closest point to i. The idea behind this attempt was to drive the system directly to the choice of the closest point for each travel. Unfortunately, this try was not fruitful and did not bring any improvement to the model, but the idea behind it led to the implementation of a constraint that was interesting to test.

2.3 Constraints

We started with an initialization of some variables of the model. So we first constrained the model by stating that every node that preceded a starting point was an arrival point:

$$previous[i] = i + m - 1 \qquad for\ i = n+2, .., n+m$$

and

$$previous[n+1] = n + 2 * m$$

We also needed to state that all points that followed arrival points were starting points:

$$next[i] = i - m + 1 \qquad for\ i = n+m+1, .., n+2*m-1$$

and

$$next[n + 2 * m] = n + 1$$

We then had to associate each start/arrival point with a courier:

$$courier[i] = i - n \qquad for\ i = n + 1, .., n + m$$
$$courier[i] = i - n - m \qquad for\ i = n + m + 1, .., n + 2 * m$$

Then we stated that the load of every courier at the beginning of its tour was equal to zero:

$$load[i] = 0 \qquad for\ i = n + 1, .., n + m$$

At this point, we started defining the constraints regarding the relationships between different points. Starting with predecessors and successors constraints:

$$next[previous[i]] = i \qquad for\ i = 1, .., n + 2 * m$$
$$previous[next[i]] = i \qquad for\ i = 1, .., n + 2 * m$$

To implement both the *allDifferent* constraint and the sub-tour elimination, we used the predicate CIRCUIT provided by MiniZinc:

$$circuit[next]$$
$$circuit[previous]$$

We moved forward constraining that the courier of point i was the same as the courier of the previous and next point:

$$courier[previous[i]] = courier[i] \qquad for\ i = 1, .., n$$
$$courier[next[i]] = courier[i] \qquad for\ i = 1, .., n$$

We finally implemented the constraints about the load of the couriers, stating that for every delivery point i the load *load[i]* was equal to the previous load plus the weight *weight[i]* of the considered item:

$$load[i] = load[previous[i]] + weight[i] \qquad for\ i = 1, .., n$$

For the arrival points at the depot, since there is no item to deliver, we set the load to be equal to the previous load:

$$load[i] = load[previous[i]] \qquad for\ i = n + m + 1, .., n + 2 * m$$

To conclude we constrained all the loads to be less or equal than the capacity of the assigned courier:

$$load[i] \leq Capacities[courier[i]] \qquad for\ i = 1, .., n + 2 * m$$

2.3.1 Implied constraints

Recalling the idea expressed in section 2.2, we have implemented a special constraint to try driving the system in assigning values to variables in a desired manner. Such manner is the following: assigning values at variable NEXT[i] starting from the closest points to i. This attempt was done by implementing the following constrain:
for i = *1,..,n* where *load[i]* $\leq$ *Capacities[courier[i]]*

next[i] in array2set(argsort([Distance[i,j] | j in 1..n where j!=i])) $\vee$ next[i]>n

This constraint is redundant since It just specifies the domain of the variable NEXT[i], which has already been declared, but It sorts the domain in a way that the values of NEXT[i] should be chosen starting from the closest points to i. The performances of such constraint are shown later.

2.4 Symmetry breaking constraints

We have reasoned about the symmetries in our solutions and studied the literature regarding the Vehicle Routing Problem. We came up with the following conclusions:
- The usual symmetry regarding the choice of a certain courier is canceled by the fact that couriers have different capacities. Of course in the case that more couriers have the same capacity, there is symmetry, but the implementation of the constraint to handle it isn't worth the computational cost.
- There is symmetry regarding the routes of each courier, since a route is the same as the "opposite" one, for instance:

$$depot => 2 => 4 => 5 => depot$$

is equivalent to:

$$depot => 5 => 4 => 2 => depot$$

We tried to exploit such symmetry in two different ways. The first way was by imposing that the number corresponding to the second delivered item in each tour had to be greater than the one corresponding to the first one. The second way was by imposing that the number corresponding to the first delivered item had to be greater than the last one delivered by the same courier. We will refer to the first described symmetry breaking constrain as s/b[1] and to the second one as s/b[2]

s/b[1]:
for i = *1,..,n* where *previous[i]*> n

$$next[i] > i$$

s/b[2]:
for i = *1,..,n* where *previous[i]* > n ,

for j = *1,..,n* where *next[j]* > $n \bigwedge$ *courier[i]* = *courier[j]*

$$next[i] > j$$

Unfortunately, performances were not improved by any of these constraints as we will show later.

2.5 Validation

In this section we will cover our process in producing the model and the obtained results, while trying different configurations. Only Gecode 6.3 was used to run the model.

2.5.1 Experimental design

We started with the implementation of a baseline that was able to solve basic problems but failed to find appreciable solutions in reasonable time. This first implementation was built on only one array of variables TOUR and used the predicate BIN_PACKING_CAPA to split the deliveries between couriers according to their capacities. The nesting in the constraints caused the model to perform poorly. We then decided to restart from scratch, using different arrays of variables to define the tour, so to avoid the problem of nesting that caused the failure of the first model and we obtained the model described earlier. After the model was able to solve basic instances of data, we moved to test it with bigger data, and We moved our focus on trying to find the best configuration of constraints and restart strategy. Regarding the search strategy we used the following sequential search strategy:
$-$ $int\_search([next[j] \mid j \; in \; POINTS], dom\_w\_deg, indomain\_random, complete)$
$-$ $int\_search(courier, first\_fail, indomain\_split, complete)$
We used as CPU an Intel core i5-10600K with processor speed of 4.1 GHz as software MiniZinc Version 2.5.3, as solver Gecode 6.3.0 and we set a time limit of 300 seconds.

2.5.2 Experimental results

We now report some results obtained by testing our model on different instances and different configurations. We first tested our model with the data contained in the file Inst01 to study the performances of the symmetry breaking constraints introduced previously. We used the annotation RELAX-AND-RECONSTRUCT(NEXT,97) and tested different restart strategies to look for the best combination. To show the results of such investigation we will refer to the constraint described in section 2.3.1 as "Constr[1]" for simplicity.

Table 1: Results of CP model on Inst01 with different configurations and different restart strategies.

	S/b[1]	S/b[2]	Without Constr[1]	With Constr[1]
Rest_luby(100)	1410	1562	1376	1486
Rest_linear(300)	1678	1438	1456	1472
Rest_geom(1.5,150)	1436	1534	1372	1492
Rest_constant(300)	1760	1658	1642	1324

At this point, we decided to drop the symmetry breaking constraints, and test more the constraint [1] and the two restarts that gave us the best results: restart_geometric(1.5,150) and restart_constant(300).

Table 2: Results of CP model on different instances on different configurations.

Instances	Constr[1] and r_const(300)	No Constr[1] and r_const(300)	Constr[1] and r_geom(1.5,150)	No Constr[1] and r_geom(1.5,150)
Inst01	1324	1642	1496	1372
Inst02	2500	2504	2530	2504
Inst03	4708	5018	4748	5018
Inst04	6860	6864	6792	6618
Inst05	12142	12316	11126	12008
Inst06	N/A	N/A	N/A	N/A
Inst07	2158	1984	2158	2146
Inst08	4570	4780	4580	4754
Inst09	8626	8602	9066	9242
Inst10	N/A	N/A	N/A	N/A
Inst11	1382	1396	1382	1312

It is important to mention the fact that the restart strategy is fundamental in order to get results while handling big-size data, but in the case of small-size data they could cause the model not to find the optimal solution since it could be cut off by the restart.

3 SAT Model

We now describe the work that led to the implementation of the SAT model in Z3, we used the Z3 Python API Z3Py.

3.1 Decision variables

Decision variables are the following, note that we are considering node n as depot and couriers indexed from 0 to $m-1$:

- $x_{i,j}$ with $0 \leq i, j \leq n$ with the semantic: $x_{i,j} = True$ iff any courier goes from node i to node j.

- $v_{i,k}$ with $0 \leq i < n, 0 \leq k < m$ with the semantic: $v_{i,k} = True$ iff courier k is assigned to node i. Note that we are not assigning any courier to node n (depot).

- $u_{i,k}$ with $0 \leq i < n, 0 \leq k < \lceil log_2 n \rceil$. This variable is crucial to tackle the problem of sub-tours. The idea behind the semantic is that u_i should be an integer variable (s.t. $0 \leq u_i \leq n$) that encodes the order in which each node is visited. Of course in our SAT model we can't use integer variables so we had to encode this information in binary notation: $u_{i,k} = True$ iff the binary representation of the integer value of u_i has 1 as the coefficient of 2^k.

3.2 Objective function

For the SAT model we used the same objective function of other models, to be minimized: the overall traveled distance, calculated as the sum of all the distances between the arcs traversed in the solution. So we just summed up over all possible combinations of i, j, the boolean value of $x_{i,j}$ multiplied by the distance between points i and j:

$$Total\ distance = \sum_{i,j=0}^{n} x_{i,j} * D_{i,j}$$

where $D_{i,j}$ is the distance between node i and node j.
Z3 offers an optimization class where it's possible to specify the objective function and ask it to minimize/maximize it given the other constraints. We couldn't use this class because it's not possible to give it a time limit for the search: it doesn't simply try iteratively to find models with lower cost (like MiniZinc) so if we stop the search after 300 seconds it doesn't return the best model found so far, but no model at all.
So we had to implement ourselves the iterative search of models with increasingly lower cost. We added the constraint

$$\sum_{i,j=0}^{n} x_{i,j} * D_{i,j} < upper\_bound$$

and asked Z3 to find a model, then fed the $Total\ distance$ of this solution into the next iteration as the new upper bound.
We started from a safe upper_bound for the objective function: the sum of all the distances between nodes. With this upper bound the solver was able to find many solutions (for the instances it was able to solve) with increasingly lower upper bound before the time limit of 300 seconds, but the final total distance found this way was very high so we tried others upper bound to start the search. After many trials we decided to use as starting upper bound the mean distance between all points multiplied by the number of the nodes: this represents an estimate of the total distance of a random path that passes through every node.

This is not a safe upper bound: depending on the constraints we may need a bigger total distance to serve all customers, but this wasn't the case for the instances we solved.
For smaller instances the solver was able to find many solutions that improved slightly this starting upper bound. To save time and skip these trivial solutions we designed a heuristic to push down the upper bound faster than the basic approach described earlier, we'll see the details in the experimental design section.

3.3 Constraints

Z3 provides the built-in functions $AtLeast$ and $AtMost$ that we have found to be the most efficient encoding for cardinality constraints, we simply combined them to build a function $exactly\_k(List, n)$ that returns a constraint that imposes that in $List$ there are at most k True values and at least k True values.

1. Exactly one arc enters and exactly one leaves each node.
   - Each row has exactly one True value: from each node leaves exactly one arc. For every $0 \leq i < n$ we impose that there is

   $$\text{exactly one True value between } x_{i,j}\text{s with } 0 \leq j \leq n$$

   note that we are not imposing restrictions on arcs leaving node n (depot)

   - Each column has exactly one True value: in each node enters exactly one arc. For every $0 \leq j < n$ we impose that there is

   $$\text{exactly one True value between } x_{i,j}\text{s with } 0 \leq i \leq n$$

   note that we are not imposing restrictions on arcs entering node n (depot)

2. The maximum number of departures from the depot must be lower than the number of available couriers (m):

   $$\text{At most } m \text{ True values between } x_{n,i}\text{s with } 0 \leq i < n$$

3. Each courier starts from depot at most once:
   For every $0 \leq k < m$:

   $$\text{At most one True value between all } v_{j,k} \wedge x_{n,j} \text{ with } 0 \leq j < n$$

4. Each customer has assigned one and only one courier.
   For every $0 \leq i < n$:

   $$\text{Exactly one True value between all } v_{i,k} \text{ with } 0 \leq k < m$$

5. If there's an arc between node i and j they must have the same assigned vehicle.
   For every $0 \leq k < m$, for every $0 \leq i, j < n$:

$$x_{i,j} \implies v_{i,k} = v_{j,k}$$

6. For each courier, the total load over its route must be smaller or equal than its capacity.
   For every $0 \leq k < m$:

$$\sum_{i=0}^{n-1} s_i * v_{i,k} \leq l_k$$

   where s_i is the size of the item i, and l_k is the maximum capacity of courier k.

7. Sub-tour elimination constraints:
   As described in section 3.1, the decision variables $u_{i,k}$ encode the order in which nodes are visited. We impose that if there is an arc between node i and node j then the index order u_j should be equal to $u_i + 1$. To encode this constraint in binary notation we implemented a function $bin\_plus\_1(a, b)$ that returns a constraint imposing that $b = a+1$ in binary notation (a and b are lists of boolean variables of length $\lceil log_2 n \rceil$)
   For every $0 \leq i, j < n$:

$$x_{i,j} \implies bin\_plus\_1(u_i, u_j)$$

   where u_i is the list $u_{i,0}, ..., u_{i,\lfloor log_2 n \rfloor}$.

   The constraints imposed by the function $bin\_plus\_1(a, b)$ are the followings:

$$b_0 = \neg a_0$$

$$b_1 = (a_1 \wedge \neg a_0) \vee (\neg a_1 \wedge a_0)$$

$$c_1 = a_0$$

   For every $i > 1$:

$$c_i = (a_{i-1} \wedge c_{i-1})$$

$$b_i = (a_i \wedge \neg c_i) \vee (\neg a_i \wedge c_i)$$

3.3.1 Implied constraints

- No arcs from a node to itself:
  For every $0 \leq i \leq n$:

$$x_{i,i} = False$$

  Adding this constraint made possible to write other constraints in a more concise way.

- The maximum number of arcs entering the depot must be lower than the number of available couriers (m):

  $$\text{At most } m \text{ True values between } x_{i,n}\text{s with } 0 \leq i < n$$

  This constraint improved the performance of our model.

3.3.2 Symmetry Breaking Constraints

As described in section 2.4, there are two possible symmetries to exploit in CVRP: one regarding the order of the couriers and one regarding the order of the nodes traversed by each courier.

- Order of traversed nodes: A route is the same as the "opposite" one, for instance:

  $$depot => 2 => 4 => 5 => depot$$

  is equivalent to:

  $$depot => 5 => 4 => 2 => depot$$

  In our SAT encoding It is simpler to encode the constraint s/b[1]: we impose that the index corresponding to the second delivered item in each tour has to be greater than the one corresponding to the first one.
  For every $0 \leq i, j < n$:

  $$(x_{n,i} \wedge x_{i,j}) \implies i < j$$

  This constraint slightly improved the performance as we will show later.

- Order of the couriers:
  We tried to impose that vehicles with bigger maximum capacity must be used before: we can't use a courier with capacity smaller than the maximum capacity between unused couriers.
  For every $0 \leq i < n$ and for every $0 \leq k1, k2 < m$:

  $$l_{k1} > l_{k2} \implies (v_{i,k2} \implies v_{i,k1})$$

  Unfortunately, this constraint did not speed up the research of a solution, so we decided to drop it.

3.4 Validation

3.4.1 Experimental Design

We'll briefly describe here different solutions tried and their effect on the performance of our model:

- We began with a model with decision variables different from the one we have in our final model. Instead of the distinct variables $x_{i,j}$ and $v_{i,k}$ we had a set of variables $x_{i,j,k}$ with the semantic: $x_{i,j,k} = True$ iff courier k goes from node i to node j. The problem with this model was that it had too many decision variables. With our current model, to encode this information, we need $n^2 + n * m$ variables while, for the previous one, we had a total of $m * n^2$ variables that is in general (for $2 \leq m < n$) a bigger number of variables. Of course, we had to update our constraints accordingly with this new set of decision variables.

- The sub-tour elimination constraint could be encoded using a more convoluted pseudo-boolean constraint like:
  For every $0 \leq i, j < n$:

$$x_{i,j} \implies \sum_{k=0}^{\lfloor log_2 n \rfloor} 2^k * u_{j,k} - \sum_{k=0}^{\lfloor log_2 n \rfloor} 2^k * u_{i,k} = 1$$

  This complicated constraint had a very bad effect on the performance of our model, so we decided to encode it using a binary adder as described in section 3.3(8). Taking advantage of the fact that we were encoding a simple +1 addition, this constraint turned out to be very simple and the performance of the model improved significantly.

- Z3 offers a way to guide the symbolic reasoning of the solver through "Tactics". We experimented with some of them but we were not able to find a good combination that improved our model's performance.

- With regards to the optimization process, as mentioned in section 3.2, we designed a heuristic to push down the upper bound faster than the basic approach. This function combines:

  - current upper bound: if we have a bigger upper bound we can subtract more.
  - time needed to find the previous solution: if we found the last solution in a short time it means that we can push the upper bound lower, conversely if we needed a lot of time we shouldn't push the upper bound down a lot.
  - time remaining: if we still have a lot of time we can try to push down the upper bound more.

  The upper bound at iteration $t + 1$ will be:

$$upper\_bound_{t+1} = upper\_bound_t - \alpha \frac{upper\_bound_t * time\_remaining}{time\_needed}$$

  α is a constant found empirically to adjust the output to the desired magnitude. We set $\alpha = \frac{1}{1000}$.

We used as CPU an Intel core i5 with a processor speed of 2.5 GHz, z3-solver Version 4.11.2.0 and we set a time limit of 300 seconds. Note that this time limit is calculated differently from the CP and the MIP models. To perform optimization as we described we have to initialize each time a new model with all its constraints. This, multiplied by each iteration of the optimization process, could take a lot of time away from the search of the solution, that is what we wanted to investigate. So we decided to consider a time limit of 300 seconds of pure search, without considering the time needed to initialize the models.

3.4.2 Experimental Results

We will report here the impact on the performance of the different symmetry-breaking constraints tried: no symmetry-breaking constraints, couriers s.b. constraint, node order s.b. constraint and both constraints.
We observed that Z3 has a much more random behaviour than MiniZinc or Gurobi: for the same query, it doesn't find always the same model, nor it does it in the same amount of time. Depending on the occasion, we were able to find solutions for an instance before the time limit or not, and for different runs the best result may vary.
This is relevant in particular for inst03: most of the times we can find a solution in 300 seconds, but not always.
Here we report a table with a single run for each method used.

Table 3: Results of SAT model on different instances with different symmetry breaking constraints

Instances	no sb constraints	couriers sb	node order sb	both
Inst01	3332	N/A	3178	N/A
Inst02	7474	N/A	7430	N/A
Inst03	N/A	N/A	14352	N/A
Inst04	N/A	N/A	N/A	N/A
Inst05	N/A	N/A	N/A	N/A
Inst06	N/A	N/A	N/A	N/A
Inst07	6154	6516	5984	N/A
Inst08	N/A	N/A	N/A	N/A
Inst09	N/A	N/A	N/A	N/A
Inst10	N/A	N/A	N/A	N/A
Inst11	3428	3250	3360	3286

We can see, as mentioned earlier in section 3.3.2, that the constraint on the order of the couriers has overall a very negative effect on the performance of the model. This is not the case for Inst11: here this constraint seems to increase the performance. It may be that the combination of the relatively small number of nodes to visit (47) and the high number of couriers (20) gave too much freedom to the solver, and this constraint was able to guide the search.

4 MIP Model

Now we introduce the MIP model that was built around the Gurobi solver.

4.1 Decision variables

The designed MIP model is in line with the SAT one, and has the following decision variables:

- $x_{i,j}$ binary variables, with $0 \leq i, j \leq n,\ i \neq j$. The semantic is that $x_{i,j} = 1$ iff a courier goes from point i to point j
- l_i integer variables with domain $[0...n]$, for $0 \leq i \leq n$. These variables represent the order in which the points are visited: $l_i < l_j$ if point i is visited before point j.
- $courier\_o\_h\_e_{i,k}$ binary variables with $0 < i \leq n$ and $0 \leq k < m$: $courier\_o\_h\_e_{i,k} = 1$ iff courier k is assigned to point i.
- $aux\_vars_{i,k}$ integer variables, for $0 < i \leq n$ and $0 \leq k < m$. It has been used to set up a necessary constraint that without an auxiliary variable would have been quadratic. In particular we needed one auxiliary variable per item per courier.

4.2 Objective function

For the MIP model, as stated before, we used the same objective function of other models, to be minimized: the overall traveled distance. As described in section 3.2:

$$Total\ distance = \sum_{i,j=0}^{n} x_{i,j} * Dist_{i,j}$$

4.3 Constraints

Regarding the constraints we started from the most generic ones, we constrained each point to have one and only one path leaving from it and one and only one path arriving to it:

$$\sum_{j=0}^{n} x_{i,j} = 1 \quad for\ i\ in\ 1, \ldots, n\ \ with\ \ i \neq j$$

$$\sum_{i=0}^{n} x_{i,j} = 1 \quad for\ j\ in\ 1, \ldots, n\ \ with\ \ j \neq i$$

In addition, we imposed the maximum number of departures from the depot to be lower than the number of available couriers:

$$\sum_{j=1}^{n} x_{0,j} \ \leq m$$

Then we set up the constraint to remove the sub-tours by labeling the order in which each point was visited. The value associated with the order was stored in the variable l and handled in such a way: $x_{i,j} = 1 \implies (l_j = l_i + 1)$. This constraint is obviously not linear so we set up the following constraint:

$$(l_i + x_{i,j}) \ \leq \ (l_j + n * (1 - x_{i,j})) \quad for\ i, j\ in\ 1, \ldots, n \ \ with\ j \neq 0$$

We also had to set the "order" of the depot point so:

$$l_0 = 1$$

The sub-tours were not the only problem regarding the validity of the found tour, we also needed to state the fact that each courier could travel only one iter. To handle this task we just linearized, with the auxiliary variables aux_vars, the constraint:

$$\sum_{i=1}^{n} courier\_o\_h\_e_{i,k} * x_{0,i} \leq 1 \qquad for\ k\ in\ 0, \ldots, m-1$$

And we ended up with:
For every $0 \leq k < m$:

$$aux\_vars_{i,k} \leq courier\_o\_h\_e_{i,k} \quad for\ i\ in\ 1, \ldots, n$$
$$aux\_vars_{i,k} \leq x_{0,i} \quad for\ i\ in\ 1, \ldots, n$$
$$aux\_vars_{i,k} \geq courier\_o\_h\_e_{i,k} + x_{0,i} - 1 \quad for\ i\ in\ 1, \ldots, n$$
$$\sum_{i=1}^{n} aux\_vars_{i,k} \leq 1$$

We set the constraints regarding the couriers' assignment: if two points are visited in sequence, they must share the same courier. This is just the SAT to MIP reduction of constraint (5) of the SAT model.
For every $k < m$, for every $0 < i, j \leq n$ with $i \neq j$:

$$x_{i,j} + courier\_o\_h\_e_{i,k} - courier\_o\_h\_e_{j,k} \leq 1$$
$$x_{i,j} - courier\_o\_h\_e_{i,k} + courier\_o\_h\_e_{j,k} \leq 1$$

After that, we implemented the constraint that imposes that the variables $courier\_o\_h\_e$ represented a one-hot encoding:

$$\sum_{k=0}^{m-1} courier\_o\_h\_e_{i,k} = 1 \quad for\ i\ in\ 1, \ldots, n$$

Finally, we set up the constraint regarding the capacities of the couriers, imposing that the sum of the weights of the items carried by each courier had to be lower or equal to their own capacity:

$$\sum_{i=1}^{n} weight_i * courier\_o\_h\_e_{i,k} \leq capacities_k \quad for\ k\ in\ 0, \ldots, m-1$$

4.3.1 Implied Constraints

We set up the following implied constraint: the number of couriers leaving from the depot had to be equal to the number of couriers arriving to the depot. This constraint is redundant for model construction, and provided improvements in the performances:

$$\sum_{j=1}^{n} x_{0,j} = \sum_{j=1}^{n} x_{j,0}$$

4.4 Validation

In this section we will cover our path in testing the model parameters that control the operations of the Gurobi solver, as well as the results obtained by the model on different instances.

4.4.1 Experimental design

After implementing a working model that was able to solve the asked problem, we moved on to test those parameters that control the Gurobi solver. We so tried the following parameters:
- $ScaleFlag = 1$: This parameter controls the model scaling and can improve performance for particularly numerically difficult models.
- $MIPFocus = 1$: This parameter modifies the high-level solution strategy. Since the numerical complexity of most of the instances, we tried to make the model focus on finding feasible solutions quickly rather than on proving optimality.

We also tried some cuts generator parameters to tackle the problem of the numerical complexity, but none of them produced any significant changes to the outcomes.
We used as CPU an Intel core i5-10600K with processor speed of 4.1 GHz, as MIP solver Gurobi Optimizer Version 10.0.0 and we set a time limit of 300 seconds.

4.4.2 Experimental results

We now report the results obtained by the model on all the available instances, while testing the impact of the previously described parameters on performances. We now show the outcomes of the model when setting: No parameters, all of them or only one of them.

Table 4: Results of MIP model on different instances on different configurations.

Instances	Both $MIPFocus$ and $ScaleFlag$	Only $MIPFocus$	Only $ScaleFlag$	No parameters
Inst01	1148	1160	1180	1180
Inst02	N/A	2016	N/A	N/A
Inst03	N/A	N/A	N/A	N/A
Inst04	N/A	N/A	N/A	N/A
Inst05	N/A	N/A	N/A	N/A
Inst06	N/A	N/A	N/A	N/A
Inst07	1408	1400	1444	1458
Inst08	N/A	N/A	N/A	N/A
Inst09	N/A	N/A	N/A	N/A
Inst10	N/A	N/A	N/A	N/A
Inst11	1144	1142	1176	1172

As we can see, MIPFocus was the parameter that improved performances the most.

5 Conclusions

After having worked on the three models described in this report, MiniZinc was the software that gave us the most satisfaction both for design freedom and overall performance, it was able, in fact, to find solutions for nine out of the eleven available instances. The MIP model, on the other hand, for those instances that was able to solve, found much better solutions with respect to MiniZinc. The SAT model was the one that performed the worst. To conclude we present in one table, the best solutions obtained by each of the models on each one of the instances.

Table 5: The best result obtained for each instance by each model.

Instances	CP	SAT	MIP
Inst01	1324	3178	1148
Inst02	2500	7430	2016
Inst03	4708	14352	N/A
Inst04	6618	N/A	N/A
Inst05	11126	N/A	N/A
Inst06	N/A	N/A	N/A
Inst07	1984	5984	1400
Inst08	4570	N/A	N/A
Inst09	8602	N/A	N/A
Inst10	N/A	N/A	N/A
Inst11	1312	3360	1142