CDMO Report

Mauro Dore (mauro.dore@studio.unibo.it)
Giacomo Gaiani (giacomo.gaiani@studio.unibo.it)
Gian Mario Marongiu (gianmario.marongiu@studio.unibo.it)

1 Introduction

This report examines three different approaches to solve the multiple courier scheduling problem: constraint programming (CP), satisfiability modulo theories (SMT) and mixed integer programming (MIP). Each model includes the following input variables, which do not change along the computation:

- $n\_couriers$: the number of couriers. For sake of readability, this variable will be referred as m.
- n_items: the number of items to deliver. For sake of readability, this variable will be referred as n.
- max_load_z: an array of length m where the cell z contains the maximum load of the courier z.
- $size\_item_i$: an array of length m where the cell i contains the weight item i.
- $all\_distances_{ij}$: a square matrix of size $(n+1) \times (n+1)$ containing the distance between each pair (i, j) of locations. This matrix has size $(n+1)$ and not n because column 0 the row 0 represent the distance from an arbitrary starting point to any other location and vice versa, respectively.

The aim of each model is to minimise the maximum distance covered by any courier.

The lower bound of each objective function is in common to all models and is equal to the round trip distance of the farthest delivery point with respect to the origin:

$$lb = max(all\_distance_{i,n+1} + all\_distance_{n+1,i}) \quad i \in \{1...n\} \tag{1}$$

The CP model was developed and tested by Giacomo Gaiani, the SMT approach was modelled by Mauro Dore, while Gian Mario Marongiu coded the MIP model. However, despite this subdivision of tasks, each member of the group was actively involved in the writing of the other models during the whole process.

Solving and optimising the Multiple Courier Routing problem required a deep knowledge about the framework used, thus, the main difficulties during development involved issues in the understanding of what was possible (and not possible) to do with the different libraries we used and, in particular, what syntax we might use in order to implement our idea of modeling.

2 CP Model

For this project we implemented our model in **Minizinc** and we ran it with two different constraint solvers, **Gecode** and **Chuffed**.

2.1 Variables and Domains

In addition to the variables already defined in chapter 1, we defined two additional decision variables representing the unknown of the problem:

- $delivery\_order_{i,j}$, where $i \in \{0, ..., n+2\}, j \in \{0, ..., m\}$.

  a matrix and representing the items delivered by each courier in chronological order and domain $delivery\_order_{i,j} \in \{0, ..., n\}$ where $delivery\_order_{i,j} = n$ indicates that at time step i, the courier j is at position n.

- $courier\_dist_i$, where $i \in \{1, ..., m\}$. an array containing the total length travelled by each courier and with domain $courier\_dist_i \in \{0, ..., w\}$ where $w = \sum_{i=0}^{n+1} \sum_{j=0}^{n+1} all\_distances_{i,j}$ with meaning of $courier\_dist_i = l$ if the courier i has travelled a total length of l.

2.2 Objective Function

The objective function was modelled trying to minimize the maximum distance travelled by any courier. In order to do so, using the array *courier_dist*, we defined two objective functions to solve the optimization problem. First we try minimizing the maximum value of the array:

$$Minimize(max(courier\_dist)) \tag{2}$$

The main issue with this first approach is the inability to discriminate better solutions with equal maximum distance travelled, especially in the instances with smaller length between delivery points. In order to implement a tie breaking mechanism for those said instances, we developed a second objective function as:

$$Minimize(m + \frac{s}{s+1}) \quad m = max(courier\_dist), s = \sum_{c=1}^{m} courier\_dist_m \tag{3}$$

Given that $s/(s+1)$ is a number less than one, the overall objective minimizes the sum of the distances travelled in the solutions with equal maximum distance.

For both objective the upper bound is equal to the sum of every element of the matrix *all_distances*, while the lower bound is described in 1:

$$ub = \sum_{i=1}^{n+1} \sum_{j=1}^{n+1} all\_distances_{i,j} \tag{4}$$

2.3 Constraints

The constraints used in the model on the decision variable *delivery_order* can be divided in three categories: the constraints necessary to find only the valid solutions given the description of the problem, the symmetry breaking constraints, and lastly the additional constraints used to reduce the search space at the expense of the optimality of the solutions. The first constraints ensures that each courier depart and return to the origin:

$$delivery\_order_{i,j} = 0 \quad \forall j \in \{1, ..., m\}, i = 1 \vee i = n + 2 \tag{5}$$

The second and the third use global constraints to ensure that every item must be delivered exactly once:

$$all\_different\_except\_0(delivery\_order)$$

$$count(delivery\_order, 0, (n + 1) \cdot m))$$

The forth, and last of the necessary constraints ensures that each courier must not exceed its maximum load:

$$\sum_{i=1}^{n+2} size\_item_i \leq max\_load_c \quad \forall c \in \{1, ..., m\}, delivery\_order_{i,c} \neq 0 \tag{6}$$

The only symmetry breaking constraint of the model perform two operation on the decision variable *delivery_order*. First it removes the reload operations of the couriers, so that every courier returns to the origin only after delivering all the items. On top of that, removes the solutions where a courier performs his delivery schedule shifted by a number of time steps:

$$delivery\_order_{i,c} \neq 0 \vee \sum_{i=j}^{n+1} delivery\_order$$

$$\forall c \in \{1, ..., m\}, \forall j \in \{2, ..., n + 1\} \mid delivery\_order_{j,c} \neq 0 \tag{7}$$

The last two constraint belong to the last category. The first ensures that every courier delivers at least one item. Knowing that the number of items is bigger than the number of couriers and that the fairness between couriers is as important as the optimality in terms of distance travelled, this constraints ensures a reduction of the search space without excluding possible better solutions:

$$\forall c \in \{1, ..., m\} \quad delivery\_order_{2,c} \neq 0 \vee \sum_{i=j}^{n+2} delivery\_order_{i,c} = 0$$

$$\forall j \in \{n+1-m, ..., n+2\} \tag{8}$$

The second additional constraint is used to drastically reduce the search space of the problem by forcing a fair distribution of the items among the couriers. Specifically, it ensures that the number of items delivered by a courier is maximum $n+1$, where n is the number of items that the couriers would deliver if the items were distributed equally among all couriers . Because the distribution of the items is relatively homogeneous considering the origin point and that the objective is to maximize the fairness between couriers, this constraint has proven to be effective.

$$j + (i-1) \cdot m > n + (2m) \rightarrow delivery\_order_{i,j} = 0$$

$$\forall i \in \{1, .., m\}, \forall j \in \{1, ..., n+2\} \tag{9}$$

The last constraint of the model is used to define the decision variable *courier_dist* but computing the distance travelled by each courier given their delivery schedule:

$$courier\_dist_i = \sum_{j=1}^{n+1} all\_distances_{d1,d2}$$

$$where \quad d1 = delivery\_order_{j,i} + 1, d2 = delivery\_order_{j+1,i} + 1$$

$$\forall i \in \{1, ..., m\} \tag{10}$$

2.4 Validation

In order to obtain the desired solution, the solver examines the remaining possible values of variables and chooses to constrain some variables further. In the model we used search annotations in order to further guide the search mechanism, always with the goal of minimizing the objective function. We performed an integer search on the variable *delivery_order* using as variable constraining *indomain_split* and as variable choice annotation *dom_w_deg* on Gecode and *input_order* in Chuffed as *dom_w_deg* is not supported.

Those decision where guided comparing the results between instances and solver used. Aside the search heuristics, the result were obtained using Python 3.10.5 and the Minizinc Python API 0.9.0. The testing was performed alternating the presence of additional constraints, **Gecode** and **Chuffed** as solvers and the two objective functions described in2.2. The Table 1 shows the difference using different solvers and using additional constraint to reduce the search space, implying small differences between Gecode and Chuffed and the importance of additional constraints in order to find sub-optimal solution. The Table 2 shows similarities in the trend of the results with the Table 1. In order to appreciate the differences in performance between the two objective functions, one must compare the quality of the results, including the sub-optimal, and the time to compute the results. The implementation of this second objective of the model was able to find betters solutions, omitted by the search mechanism of the first objective, at the expense of longer computational time.

ID	initMod(Ge)	initMod(Ch)	AddConst(Ge)	AddConst(Ch)
1	**14**	**14**	**14**	**14**
11	N/A	N/A	615	692
16	**286**	N/A	**286**	**286**
18	N/A	N/A	819	N/A

Table 1: from left to right the result of the initial default model using Gecode and Chuffed as solver, the model with additional constraints using Gecode and Chuffed, all using maximum distance as objective function

ID	initMod(MD)	initMod(OF)	AddConst(MD)	AddConst(OF)
1	**14**	**14**	**14**	**14**
4	**220**	220	**220**	220
18	N/A	N/A	819	988

Table 2: from left to right the result of the initial default model using respectively the first2 and second 3 objective functions, the model with additional constraints using respectively the first and second objective functions, all using Gecode as solver

3 MIP Model

For this project we implemented our model in Python using the *gurobipy* library.

3.1 Variables and Domains

In addition to the variables already defined in chapter 1 we defined the following decision variables:

- $x_{z,i,j} \in \{0,1\}$, where $z \in \{0,...,m\}, i \in \{0,...,n+1\}, j \in \{0,...,n+1\}$.
  Conceptually, $x_{zij} = 1$ means that the courier z, in its route, will go from the delivery point i to the delivery point j.
- $u_i \in \{0,...,n\}$, where $i \in \{0,...,n+1\}$.
  It is a list of integer decision variables used to avoid couriers sub-tours. It can be seen as the "order" in which a courier visits a location in its route, thus, each value u_i can assume values in the range $\{0,...,n+1\}$.
- $maxTravelled$ and $minTravelled$, two integers which stores the maximum and minimum distance travelled by any courier, respectively.

3.2 Objective Function

In the first version of our model we minimize the maximum distance travelled by any courier (11):

$$Minimize(maxTravelled) \tag{11}$$

We also tried to explicitly minimise the sum between the total distances travelled by all couriers and the difference between the maximum and minimum length of a travelled path (12):

$$f : \sum_{z=0}^{m} \sum_{i=1}^{n+1} \sum_{j=1}^{n+1} all\_distances_{z,i,j} \cdot x_{z,i,j}$$

$$Minimize(f + (maxTravelled - minTravelled)) \tag{12}$$

In these two objective functions the values of *maxTravelled* and *minTravelled* are calculated by updating their values by means of the constraint 13:

$$minTravelled \leq \sum_{i=0}^{n+1} \sum_{j=0}^{n+1} all\_distances_{i,j} \cdot x_{i,j,z} \leq maxTravelled \tag{13}$$

$$\forall z \in \{0, ..., m\}$$

The efficiency and results of the objective functions above are better described in chapter 3.4.

3.3 Constraints

The first constraint in the model ensures that each item is delivered exactly once, namely that every edge which leads to any node (except the starting point) must be travelled by only one courier:

$$\sum_{z=0}^{m} \sum_{i=1}^{n+1} x_{z,i,j} = 1 \qquad \forall j \in \{0, ..., n+1\}, i \neq j, j \neq 0 \tag{14}$$

Also, every node should be reached and left by the same vehicle. We achieve this by stating that the number of times a vehicle enters a node is equal to the number of times it leaves that node:

$$\sum_{j=1}^{n+1} x_{z,i,j} - x_{z,j,i} = 0 \qquad \forall i \in \{0, ..., n+1\}, \forall z \in \{0, ..., m\}, i \neq j \tag{15}$$

Every courier have to enter the starting point once. This is achieved by saying that the number of successors of the depot must be exactly one for each courier:

$$\sum_{i=1}^{n+1} x_{z,i,0} = 1 \qquad \forall z \in \{0, ..., m\}, i \neq 0 \tag{16}$$

Similarly, every courier have to leave the starting point once:

$$\sum_{j=1}^{n+1} x_{z,0,j} = 1 \qquad \forall z \in \{0, ..., m\}, j \neq 0 \tag{17}$$

Each courier does not exceed its maximum load:

$$\sum_{i=0}^{n+1}\sum_{j=0}^{n+1} size\_item_j \cdot x_{z,i,j} \leq max\_load_z \qquad \forall z \in \{0,...,m\} \tag{18}$$

We also need to forbid sub-tours in the route of each courier. We achieve this by implementing a set of constraint based on the MTZ (Miller-Tucker-Zemlin) approach, a well-known method to solve the sub-tour problem. First of all, we assert that the starting point must always be the first node visited ($u_0 = 1$) and that every other point must be visited afterwards ($u_i \geq 2 \forall i \in \{0,...,n+1\}, i \neq 0$). Then we state that if a courier goes from i to j, then j comes after i, thus u_j must be greater than u_i:

$$x_{z,i,j} \cdot u_j \geq x_{z,i,j} \cdot u_i + 1 \qquad \forall i,j \in \{0,...,n+1\}, i \neq j, i \neq 0, j \neq 0 \tag{19}$$

In addition to the constraints above we found two implied constraints, 20 and 21 empower the assumption that each item can be delivered only once and by different couriers, respectively. The symmetry breaking constraint 22 says that couriers with a lower *max_load* will always have a smaller load with respect to couriers with higher ones.

$$\sum_{j=1}^{n+1} x_{z,i,j} = 1 \qquad \forall z \in \{0,...,m\}, \forall i \in \{0,...,n+1\}, i \neq j \tag{20}$$

$$\sum_{z=0}^{m} x_{z,i,j} = 1 \qquad \forall i,j \in \{0,...,n+1\}, i \neq j \tag{21}$$

$$\sum_{i=0}^{n+1}\sum_{j=0}^{n+1} size\_item_j \cdot x_{z_1,i,j} \leq \sum_{i=0}^{n+1}\sum_{j=0}^{n+1} size\_item_j \cdot x_{z_2,i,j} \tag{22}$$

$$\forall z_1, z_2 \in \{0,...,m\} \quad if max\_load_{z1} \leq max\_load_{z2}$$

3.4 Validation

Our experimental study began with the model which uses all the constraints described in 3.3 and the objective 11. We first ran the *tune()* gurobi tool on this model, the function automates the testing of many of the 57 parameters that influence the solving time. Applied to smaller instances, this function was "unable to improve on baseline parameter set". Applying the same procedure to larger instances, the only suggestion from the tool was to set the parameter *focus* of the model in such a way that it first focuses on finding feasible solutions rather than looking for optimal ones from the beginning. Since *gurobi* seems to provide a good initial combination of parameters, we focused on extending the model with implied and symmetry breaking constraints, obtaining the results shown in 3.

ID	default-model	implConst	symBreak	impl-SymBreak	focusParam
4	**220**	**220**	**220**	**220**	**220**
7	**167**	**167**	**167**	**167**	**167**
13	490	514	456	470	490
16	**286**	**286**	N/A	N/A	**286**
19	848	**334**	N/A	N/A	848
21	N/A	N/A	N/A	N/A	N/A

Table 3: from left to right, the result of the initial model, then extended with implied constraints, with symmetry breaking constraints, with both implied and symmetry breaking constraints, with the parameter for focusing on immediate feasible solutions

We also tested this model with a different objective function (12). We did not reported a table with the results of this second model but it is still possible to test the model by code. In general, we observed that this second model was usually able to find a shorter overall path at the expense of a fair distribution of items between couriers, while the initial model sometimes led to a longer overall path but had a higher equity in the distribution of items to be delivered. Even though the second model seems to guarantee a better fairness among the distribution of items, the first one is much faster at computing optimal solutions and better able to find feasible solutions for larger instances. In particular, the results in 3 show that for the first ten instances the model can easily find an optimal solution in less than 1 second, while for larger instances the model finds the optimal one in few cases. In general, the implied and symmetry breaking constraint do not seem to apport a significant contribute to the computation (as observed in chapter5), and the *focus* parameter showed performance similar to the default model.

4 SMT Model

For this project we implemented our model in Python using the **z3** library.

4.1 Variables and Domains

In addition to the variables defined in chapter 1, the model also uses the same decision variables already defined in 3.1.

4.2 Objective Function

Considering the fact that we developed MIP and SMT in parallel, in the initial version of our model, we aimed to minimize the maximum distance traveled by any courier (23). While afterwards, we experimented with minimizing the total distance traveled by all couriers and the difference between the longest and shortest path travelled (24).

$$Minimize(maxTravelled) \tag{23}$$

$$totalDistance : \sum_{k=0}^{m} \sum_{i=1}^{n+1} \sum_{j=1}^{n+1} all\_distances_{i,j} \cdot x_{i,j,k}$$

$$Minimize(totalDistance + (maxTravelled - minTravelled)) \tag{24}$$

In order to minimise our objective function we used the class Optimize() provided by *z3*, which enables the definition of an objective function to be minimized/maximized in compliance with specified constraints. We could not use this function because if we stop the research after the time limit it does not return a solution unless it is optimal. For this reason we had to implement a search that iteratively finds a model with a increasingly lower cost using the following constraint (25).

$$upper\_bound > maxTravelled \tag{25}$$

We used the class Solver(), asked *z3* to find a model, then fed the value *maxTravelled* of the solution into the next iteration as upper bound, until the search has found a minimum solution within the time limit.

4.3 Constraints

1. The first constraint in the model ensures that there are no arcs from a node to itself.

$$\sum_{i=0}^{n+1} x_{i,i,k} = False \qquad \forall k \in \{0, \ldots, m\} \tag{26}$$

2. Every item must be delivered. Each 3-dimensional column must contain only 1 true value, depot not included in this constraint

$$\sum_{k=0}^{m} \sum_{j=0}^{n+1} x_{i,j,k} = 1 \qquad \forall i \in \{0, ..., n+1\}, i \neq j, j \neq 0 \tag{27}$$

3. Every node should be reached and left once and by the same vehicle. In other words, the number of times a vehicle enters a node is equal to the number of times it leaves that node.

$$\sum_{j=1}^{n+1} x_{i,j,k} = \sum_{j=1}^{n+1} x_{j,i,k} \qquad \forall i \in \{0, ..., n+1\}, \forall k \in \{0, ..., m\}, i \neq j \tag{28}$$

4. Each courier leaves and enters exactly once the depot, so the number of predecessors and successors of the depot must be exactly one for each courier.

   Every courier have to enter the starting point once.

$$\sum_{k=0}^{m} x_{i,0,k} = 1 \qquad \forall i \in \{0, ..., n+1\}, i \neq 0 \tag{29}$$

   In the same way, every courier have to leave the starting point once.

$$\sum_{k=0}^{m} x_{0,j,k} = 1 \qquad \forall j \in \{0, ..., n+1\}, j \neq 0 \tag{30}$$

5. For each vehicle, the total load over its route must be smaller than its max load size.

$$\sum_{i=0}^{n+1} \sum_{j=0}^{n+1} size\_item_i \cdot x_{i,j,k} \leq max\_load_k \qquad \forall k \in \{0, ..., m\} \tag{31}$$

6. To prevent couriers from taking unnecessary detours or creating sub-routes we implemented a set of constraint based on the MTZ (Miller-Tucker-Zemlin) approach in both MIP and SMT 19.

4.3.1 Implied Constraint

As described in the chapter 3.3, in the SMT model we implemented Implied constraint and Symmetry Breaking constraint too.

- If a courier goes through the arc (i, j) than all the other couriers must not cross the arc (i, j)

$$x_{i,j,k} \Rightarrow \bigwedge_{k' \in \{0,1,...,m\}, k' \neq k} \neg x_{i,j,k'} \qquad \forall i, j \in \{0, 1, \ldots, n{+}1\}, \forall k \in \{0, 1, \ldots, m\} \tag{32}$$

- For every courier, each row contains only one True

$$x_{i,j,k} \Rightarrow \bigwedge_{j' \in \{0,1,...,n+1\}, j' \neq j} \neg x_{i,j',k} \qquad \forall i, j \in \{0, 1, \ldots, n{+}1\}, \forall k \in \{0, 1, \ldots, m\} \tag{33}$$

ID	default-model	implConst	symBreak	impl-SymBreak
1	**14**	**14**	**14**	**14**
3	**12**	**12**	**12**	**12**
7	280	223	276	295
10	**244**	**244**	274	**244**
13	N/A	N/A	N/A	N/A

Table 4: from left to right, the result of the initial model, then extended with implied constraints, with symmetry breaking constraints, with both implied and symmetry breaking constraints

4.3.2 Symmetry breaking

Given the load of each courier k:

$$load_k = \sum_{i=0}^{n+1} x_{i,j,k} * size\_item_i \tag{34}$$

$$max\_load_{k1} < max\_load_{k2} \Rightarrow load_{k1} \leq load_{k2} \qquad \forall k_1, k_2 \in \{0, \ldots, m\}, k1 \neq k2 \tag{35}$$

4.4 Validation

We started our experimental study with a model which uses all the constraints described in 4.3 and the objective 23. The coding of the MIP and SMT model proceeded simultaneously, hence, as we also described in 3.4, we extended the model with implied and symmetry breaking constraints, and after the test, we obtained the results shown in 4. In a second phase, we tested the model with a different objective function (24). As in paragraph 3.4, we did not reported a table with the results of this second model but it is still possible to test the model by code. As observed in the MIP model, the second objective allows to discover shorter overall routes, at the cost of a balanced distribution of items among couriers. In contrast, the initial model occasionally resulted in longer overall routes but maintained a more equal allocation of items for delivery. While the second model prioritizes fairness in item assignments. Additionally, the extended version of the model, incorporating symmetry-breaking and implied constraints, usually underperform the raw model, except rare cases where it decreases the search time. One problem we have encountered is the long time required by the loading phase of the constraints, which is not taken into account when calculating the final running time of the solver. The configuration that usually takes the longest time to find a solution is the one with implied and symmetry-breaking constraints, while the one with only symmetry-breaking is more alike to no return an optimal solution. Overall, the number of optimal solutions decreases as the complexity of the instances increases.

5 Conclusions

As can be seen from the results, the three models had good performance on the firsts instances, as the CP and MIP models solved instances from one to ten in less than a second. The SMT model presented slightly worst results, due to the search mechanism. The brute-force models of every approach work well on small instances, while for bigger ones it is necessary to add elements which can support the solution searching process. In the optimization process, the lower bound for the objective function has been very helpful for the models performance, while it was very difficult to find valid symmetry breaking constraints capable of reducing the search space in a meaningful way. The difficulty was given by the fact that in the formulation of the problem that we were trying to solve every ordering had a significant meaning, which means that almost every combination of values for the decision variables led to different values for the objective function. We were also able to explicit some implied constraint in the MIP and SMT, which never showed significant results. In fact, there were very few cases where the extended version of a model (with symmetry breaking and implied constraints) performed better than the raw model. We believe that, when it happened it was not due to an actual significant reduction of the search space, but probably due to the different order of reasoning that the solver was forced to follow given these constraints, which led to certain solutions being found earlier in the search process, speeding up the whole optimisation. A different CP model was also developed and tested, with similar structure to the SMT and MIP models, but was later discarded due of the worse performance in comparison to the CP model described in the report.

Moreover, we sought scientific papers that could described possible approaches to improve our model. We found several methods that were not compatible with the formulation of the problem we were working on, such as particular ways to compute a lower bound for the objective function, or the elimination of paths which could never be part of an optimal solution. Such methods were applied to various versions of the multiple courier routing problem, different from ours, hence one of the few possible paths to improve the results in the more complex instances is to find additional constraint to further reduce the search space at the expense of the optimality of the solutions.