CDMO Project Report

Mohd Shariq Ansari
mohdshariq.ansari@studio.unibo.it

Michele Carusone
michele.carusone@studio.unibo.it

Davide Cremonini
davide.cremonini8@studio.unibo.it

December 8, 2024

1 Introduction

1.1 Problem Introduction

This report presents 4 alternative methods to tackle and solve the well-known Multiple Couriers Planning problem (MCP). The methods implemented are CP, SAT, SMT and MIP. The problem is defined by some core parameters which are provided by the instances:

- m, the number of couriers
- n, the number of items
- l, the array containing at index i the maximum capacity of the i-th courier
- s, the array containing at index i the size of the i-th item
- D, the distance matrix, containing at $D_{i,j}$ the distance from node i to node j, with node $n+1$ being the depot node.

The main objective of the problem is to minimize the maximum distance traveled by the couriers.

1.2 Common elements between models

Other than the basic parameters, all of our models accept two more which help guide the search process, namely upper and lower bounds for the objective function. These parameters are computed starting from the instance and passed down to the individual solvers.

Upper Bound To compute the upper bound we assume a worst-case scenario in which one courier is assigned with every item. We will focus on the distance traveled by it to find an upper bound. We can consider a wide approximation of this distance as the upper bound, it can be computed by summing all the maximum distances along each row, this is close to assuming the courier always takes the longest possible route between each node.

$$UB = \sum_{i=1}^{n+1} (\max_{j \in \{1,\dots,n+1\}} (D_{i,j})) \tag{1}$$

Lower Bound To compute the lower bound we assume a best-case scenario where $n = m$ so that every courier needs to carry only one item. Under these circumstances, the distance traveled by each courier will be a round-trip between the depot and the node associated with its assigned item. We can assume the maximum of these round trips as a lower bound for the objective function.

$$LB = \sum_{i=1}^{n} (D_{n+1,\,j} + D_{j,\,n+1}) \tag{2}$$

Problem Symmetries Analysing the problem we discovered two main symmetries which can be exploited in order to reduce the solution space of the problem:

- **Load symmetry**
  Assuming two or more couriers have the same capacity, we can say that they are virtually identical and can be swapped without changing the overall solution. In these cases, a good way to reduce the solution space would be to impose an ordering constraint on the items assigned to each courier, in order to avoid similar solutions.

- **Distance matrix symmetry**
  In case the distance matrix is symmetrical (such is the case for instances from 11 to 21) the solution space can be further reduced. Under these assumptions the distance from two nodes is the same in both directions: by propagating this propriety we can say that every possible courier's path through an ordered series of nodes can be traversed in one direction or the other without changing the distance traveled, thus creating two similar solutions. In order to prune the solution space we can impose an ordering constraint between the first node the courier visits and the last one.

1.3 Work division

Initially, the workload was split as follows:

- Mohd Shariq Ansari: SMT

- Michele Carusone: MIP
- Davide Cremonini: CP, SAT

The project was developed between September and early December 2024. In the first month of the project each group member developed their models independently, then in the second month, during a series of in-person meetings, each model was collectively improved and tested. The last month saw the group members coming together to integrate the different models into a single solving program which was then loaded and further tested on docker. The last couple of weeks were dedicated to writing the report.

2 CP Model

Constraint Programming represents one of the most renowned methods to perform optimization, and the Minizinc language is definitely one of the best CP modeling languages available. During the development process, three models were implemented but only the most effective and compact one will be presented below. Discarded encodings for routes included a $m \times (n+1) \times (n+1)$ binary tensor and one using a $(n+1) \times (n+1)$ matrix with values in $\{0, \ldots, m\}$. The chosen model tries to fully utilize global constraints and their efficient propagation.

2.1 Decision Variables

- $\boldsymbol{Z} \in \{1, \ldots, n+1\}^{m \times (n+1)}$ encodes both assignment and routing for the couriers. Each row is structured to represent a linked list, so that:
  - $Z_{i,j} = k,\ j \neq k$ means that the courier i will move from node j to node k
  - $Z_{i,j} = k,\ j = k$ means that the courier i won't reach node j
- $\boldsymbol{Carry} \in \{1,\ldots,\mathrm{m}\}^n$ encodes the assignment for each courier: if $Carry_i = j$ it means courier i is carrying item j
- $\boldsymbol{Dist} \in \{\mathrm{LLB},\ldots,\mathrm{UB}\}^m$ represents the distances traveled by each courier. Each distance is limited between the upper bound UB discussed before and a minimum value LLB computed as the minimum round trip time possible for all couriers. The maximum of these values will be considered the objective function to minimize.

2.2 Objective Function

The objective function for this model is the maximum of the distances traveled by each courier, in this case, $\max_{i=\{1,\ldots,m\}} Dist_i$. Each one of the distances is

computed by summing values in the D matrix for the arcs traveled by each courier i. This relationship is enforced by a constraint:

$$\forall i \in \{1, \ldots, m\} \;\; Dist_i \;\; = \sum_{\substack{j=1 \\ Z_{i,j} \neq j}}^{n+1} D_{j,Z_{i,j}} \tag{3}$$

The objective function is bound by LB and UB, these boundaries are enforced by a straight-froward constraint.

2.3 Constraints

We now describe the constraints that bind the model.

- Each item can only be carried by one courier, this is enforced by using the *count* global constraint so that along each column of Z there is only one occurrence of $Z_{i,j} \neq j$, this way there will be only one courier reaching node j.

$$\forall j \in \{1, \ldots, n\} \;\; count_{i \in \{1,\ldots,m\}}(Z_{i,j} \; if \; Z_{i,j} \neq j) \;\; = 1 \tag{4}$$

- Load capacities must be respected, this is enforced by constraining the sum of all sizes of the items carried by each courier i to be lesser or equal to the capacity of said courier.

$$\forall i \in \{1, \ldots, m\} \quad \sum_{\substack{j=1 \\ Z_{i,j} \neq j}}^{n} s_j \leq l_i \tag{5}$$

- Each courier must leave the depot, this is enforced by constraining the last element of each row of Z (which represents the node reached from the depot) to be different from the depot itself.

$$\forall i \in \{1, \ldots, m\} \;\; Z_{i,n+1} \neq n+1 \tag{6}$$

- Each route must be a circuit, by using the *subcircuit* Minizinc constraint we enforce each row i of Z to be a circuit where each node only has one ingoing and one outgoing arc and also ensures that the circuit is closed by starting and ending on the same node and crossing every node once.

$$\forall i \in \{1, \ldots, m\} \;\; subcircuit(Z_{i,1..n+1}) \tag{7}$$

Implied Constraints
One *all_different* implied constraint was added as its inclusion has proven useful to improve performance. It constrains every row of Z to be a permutation of numbers from 1 to $n+1$. However, by definition the *subcurcuit* constraint needs to make sure that the list it constrains is constituted by a permutation of all elements present.

$$\forall i \in \{1, \ldots, m\} \;\; all\_different(Z_{i,1..n+1}) \tag{8}$$

Redundant Constraints
Despite the load capacities were already dealt with using 5 to improve performance a second formulation of this constraint was added by leveraging the global constraint *bin_packing_capa*. This constraint takes three inputs *capacities*, *assignments* and *weights* and makes sure that the sum of weights defined by assignments does not exceed the specified capacity. To utilize *bin_packing_capa* it was necessary to add the $Carry$ decision variables and a channeling constraint for the assignments between Z and $Carry$.

$$\forall\, i \in \{1, \ldots, m\}\, \forall\, j \in \{1, \ldots, n\}\;\; Z_{i,j} \neq j \implies Carry_j = i \tag{9}$$

$$bin\_packing\_capa(l,\;\; Carry,\;\; s) \tag{10}$$

Symmetry Breaking Constraints
Both symmetries described above were implemented for the CP model.

- Distance matrix symmetry was tackled by imposing an ordering between the first and last nodes visited by each courier so that the last was greater than the first. On each row the last node can be identified by using the Minizinc *arg_max* function as the node pointing to the depot will have the highest value, while the first is the node pointed by the last element of the row.
$$\forall\, i \in \{1, \ldots, m\}\;\; arg\_max(Z_{i,1..n+1}) > Z_{i,n+1} \tag{11}$$
- Load symmetry was instead broken by imposing a lexicographic ordering constraint between the paths traveled by two couriers with the same capacity, to do so, the *lex_lesseq* Minizinc constraint was used.
$$\begin{aligned} &\forall\, i \in \{1, \ldots, m\} \qquad \forall\, j \in \{i+1, \ldots, m\} \\ &l_i = l_j \implies lex\_lesseq(Z_{i,1..n+1}, Z_{j,1..n+1}) \end{aligned} \tag{12}$$

2.4 Validation

Experimental Design
As the Minizinc model supports different solvers, our experimental setup followed two main steps: first, we tried experimenting with different solvers (Gecode, Chuffed and Highs) to find the one that worked best. All of these attempts were conducted by experimenting with different search strategies such as First-Fail, Dom_W_Deg with and without random assignments. Once we determined the effectiveness of Gecode above the others, further configurations were implemented for it. Using the Gecode solver, further attempts were made with the following configurations:

1. Symmetry Breaking on both Loads and Distance matrix
2. Solver restart with different strategies (Exponential, Luby)

3. Restart with sequential search, relaxation and reconstruction to explore solution neighborhoods.

Different combinations of these strategies were attempted, below we report those which lead to the most interesting results.

Experimental Results
Using sequential search led to certifying the optimality for instances 12, 16

Instance	Gecode	Chuffed	Highs	Gecode_sb	Gecode_ss	Gecode_ss_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**	186	168	**167**	**167**	**167**
8	**186**	**186**	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**	**244**	**244**
11	762	N/A	N/A	1056	422	402
12	487	N/A	N/A	544	**346**	**346**
13	1074	N/A	N/A	1192	444	444
14	1098	N/A	N/A	1098	699	730
15	999	N/A	N/A	999	716	673
16	433	N/A	N/A	433	**286**	**286**
17	1414	N/A	N/A	1445	1101	1112
18	937	N/A	N/A	1011	574	587
19	461	N/A	N/A	534	**344**	**344**
20	1461	N/A	N/A	1380	1073	1246
21	895	N/A	N/A	865	444	442

Table 1: Table showing results of the CP Model

and 19. Another thing to note was that the usage of restarts made it difficult for the solver to certify optimality for very small instances such as 1 and 3, we hypothesized that frequent restarts, which are useful for solving large instances lead to incomplete exploration in smaller ones.

3 SAT Model

The main challenge with this approach was managing integer values and handling conversions and constraint propagation across all boolean variables. In order to enforce all constraints many ad-hoc functions were defined to be used throughout the model, such as *compare*, to impose comparison constraints, *sum_cond_on_array_constraint*, to propagate sum of multiple variables, *consecutive*

which ensures that the 1s occurring in two arrays are shifted by one position. In order to operate only on boolean values, all integer instance variables were converted into lists of booleans.

3.1 Decision Variables

- $\boldsymbol{A} \in \mathbb{B}^{m \times n}$ is a matrix representing item assignments, if $A_{i,j}$ is $True$, courier i is carrying item j.
- $\boldsymbol{G} \in \mathbb{B}^{m \times (n+1) \times (n+1)}$ represents the route graph, if $G_{i,j,k}$ is $True$, courier i follows the arc from node j to node k.
- $\boldsymbol{V} \in \mathbb{B}^{n \times n}$ represents delivery order, if $V_{i,j}$ is $True$, item i is carried as the j-th element of the route it belongs to.

Support Variables
To define each constraint in a SAT encoding, other variables were added to support constraint propagation.

- **internal sum variables**, in order to perform boolean sums inside the dedicated functions, arrays of boolean variables where necessary both to take care of carries and to memorize partial sums when summing multiple elements.
- **Distances and Loads** variables were defined end encoded in order to memorize value produced by the sums and allow constraint enforcing on said results.

All of these variables were encoded with a total number of bits sufficient for encoding the upper bound of both distances and loads.

3.2 Objective Function

The objective function is represented with an array of variables encoded as the other distance variables and named Max_Dist. The array itself is constrained in the interval $[LB..UB]$. This is achieved by using the *compare* function. To ensure the Max_Dist is the actual maximum distance, two more constraints are added:
Max_Dist is one of the distances and is greater or equal than any other:

$$\bigwedge_{i=1}^{m} (Max\_Dist \geq Dist_i) \qquad \bigvee_{i=1}^{m} (Max\_Dist = Dist_i) \tag{13}$$

3.3 Constraints

Below we describe the constraints used by the sat model, as discussed before some of them were implemented on a binary level using the functions *compare*, *sum_cond_on_array_constraint* and *consecutive* to define each constraint more easily.

- Every item is carried

$$\bigwedge_{j=1}^{n}(Exactly\_One(A_{1..m,j})) \tag{14}$$

- Every courier load is the sum of the items it carries and must respect the courier's capacity

$$\bigwedge_{i=1}^{m}(C_i = \sum_{\substack{j=1 \\ A_{i,j}=True}}^{n} s_j) \qquad \bigwedge_{i=1}^{m}(C_i \leq l_i) \tag{15}$$

- Every node is visited exactly once

$$\bigwedge_{j=1}^{n}(Exactly\_One(V_j)) \tag{16}$$

- There are no self-loops in each courier's route

$$\bigwedge_{i=1}^{m}\bigwedge_{j=1}^{n}(\neg G_{i,j,j})) \tag{17}$$

- Channeling constraints between assignments and routes: if item is assigned to a courier, its node must belong to its path.

$$\bigwedge_{i=1}^{m}\bigwedge_{j=1}^{n+1}((A_{i,j} \implies Exactly\_One(G_{i,j,1..n+1})) \wedge (\neg A_{i,j} \implies \neg \bigvee_{k=1}^{n+1}(G_{i,j,k}))) \tag{18}$$

$$\bigwedge_{i=1}^{m}\bigwedge_{k=1}^{n+1}((A_{i,k} \implies Exactly\_One(G_{i,1..n+1,k})) \wedge (\neg A_{i,k} \implies \neg \bigvee_{j=1}^{n+1}(G_{i,j,k}))) \tag{19}$$

- Every courier leaves from and return to the depot

$$\bigwedge_{i=1}^{m}(Exactly\_One(G_{i,n,1..n+1})) \qquad \bigwedge_{i=1}^{m}(Exactly\_One(G_{i,1..n+1,n})) \tag{20}$$

- Sub-tour Elimination Constraint

$$\bigwedge_{i=1}^{m}\bigwedge_{j=1}^{n}((\bigwedge_{k=1}^{n}(G_{i,j,k} \implies consecutive(V_j, V_k))) \wedge (G_{i,n,j} \implies V_{j,1})) \tag{21}$$

- The distance traveled by a courier is the sum of the arcs it follows

$$\bigwedge_{i=1}^{m}(Dist_i = \sum_{j=1}^{n+1}\sum_{\substack{k=1 \\ G_{i,j,k}=True}}^{n+1} D_{j,k}) \tag{22}$$

Implied Constraints
Every courier delivers at least one item

$$\bigwedge_{i=1}^{m}(\bigvee_{j=1}^{n} A_{i,j}) \tag{23}$$

Symmetry Breaking Constraints
If two couriers have the same capacity, the first's assignments encodes a larger integer than the second's.

$$\bigwedge_{i=1}^{m}(\bigwedge_{j=i}^{m}((l_i = l_j) \implies (A_{i,1..n} \geq A_{j,1..n}))) \tag{24}$$

3.4 Validation

Experimental Design
Given the difficulties encountered in the definition of the SAT model, the experiment focused more on using a z3 solver and refining the search algorithm that could lead to the best results. This was achieved by combining the symmetry-breaking techniques with three types of search:

- z3 native optimization
- linear search
- binary search

Experimental Results
Comparing the different solution strategies, we observe no particular differences

Instance	z3	z3_sb	z3_ls	z3_bs
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	248	N/A	N/A	317
8	**186**	**186**	**186**	**186**
9	436	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**

Table 2: Table showing results of the SAT Model

between them: the binary search is the only one capable of both producing a solution for the 7-th instance and certifying all the others.

4 SMT Model

The Satisfiability Modulo Theory (SMT) refers to the problem of determining whether a first-order formula is satisfiable with respect to some logical theory[1]. We decided to approach the problem first by using z3 solver and then move toward solver-independent models (SMT-LIB). The solution is designed to reduce the use of non-binary logic, unless required, employing the integer logic only for load capacities and expressing the objective function.

4.1 Decision Variables

The following decision variables are defined in our SMT model:

- $\boldsymbol{load} \in \mathbb{N}^m$: at index k represents the total load carried by courier k.
- $\boldsymbol{assignment} \in \mathbb{B}^{m \times n}$: This is a variable that represents whether item j has been assigned to courier k.
- $\boldsymbol{route} \in \mathbb{B}^{(n+1)\times(n+1)\times m}$: `true` if courier k travels from location i to location j, representing a valid route between i and j.
- $\boldsymbol{visit\_order} \in \{0, \ldots, n\}^{(n+1)\times m}$: represents the order in which courier k visits location j. This variable helps ensure that trips start and end at the depot and aids in sub-tour elimination.
- $\boldsymbol{total\_distance} \in \mathbb{N}^m$: stores the total distance traveled by courier i. The array is populated as follows:

$$\forall\, k \in \{1, \ldots, m\} \quad total\_distance_k = \sum_{i=0}^{n} \sum_{\substack{j=0 \\ route_{i,j,k}=True}}^{n} D_{i,j} \tag{25}$$

- $\boldsymbol{max\_distance} \in \mathbb{N}$: represents the maximum distance traveled by any courier.

4.2 Objective Function

The max_distance variable represents the objective function to minimize and it's also bounded by Lower Bound(LB) and Upper Bound(UB). It is constrained to be greater or equal than any courier total distance as follows:

$$total\_distance_k \leq max\_distance \quad \forall k \in \{1, \ldots, m\} \tag{26}$$

4.3 Constraints

- **Each item should be assigned to exactly one courier**

$$\bigwedge_{k=1}^{m} Exactly\_One(assignment_{k,1..n}) \tag{27}$$

- **Load and weight constraint**

$$load_i = \sum_{\substack{j=1 \\ assignment_{i,j}=True}}^{n} weight_j \quad \forall i \in \{1, \ldots, m\} \tag{28}$$

$$load_i \leq capacity_i \quad \forall i \in \{1, \ldots, m\} \tag{29}$$

- **Channelling constraint** Chanelling constraint ensure that if there is an assignment of item j to courier k, courier k must have a route with j and vice-versa.

$$assignment_{k,j} \implies \bigvee_{i=1,\, i\neq j}^{n+1} route_{i,j,k}, \quad \forall k \in \{1, \ldots, m\} \quad \forall j \in \{1, \ldots, n\} \tag{30}$$

$$\bigvee_{i=1,\, i\neq j}^{n+1} route_{i,j,k} \implies assignment_{k,j}, \quad \forall k \in \{1, \ldots, m\} \quad \forall j \in \{1, \ldots, n\} \tag{31}$$

- **Constraint to ensure start and end of trip at depot**

$$\sum_{j=1}^{n} route_{n+1,j,k} = 1, \quad \forall k \in \{1, \ldots, m\} \tag{32}$$

$$\sum_{j=1}^{n} route_{j,n+1,k} = 1, \quad \forall k \in \{1, \ldots, m\} \tag{33}$$

- **Constraint to ensure continuity in route and sub-tour elimination** This constraint ensures that each courier starts at the depot and the incoming and outgoing routes are exactly one and there are no self-loops.

$$visit\_order_{k,n+1} = 0, \quad \forall k \in \{1, \ldots, m\} \tag{34}$$

$$\sum_{\substack{i=1 \\ i\neq j}}^{n+1} route_{i,j,k} = \begin{cases} 1, & \text{if assignment}_{k,j} = True, \\ 0, & \text{otherwise.} \end{cases}, \quad \forall k \in \{1, \ldots, m\}, \forall j \in \{1, \ldots, n\} \tag{35}$$

$$\sum_{\substack{i=1 \\ i\neq j}}^{n+1} route_{j,i,k} = \begin{cases} 1, & \text{if assignment}_{k,j} = True, \\ 0, & \text{otherwise.} \end{cases}, \quad \forall k \in \{1, \ldots, m\}, \forall j \in \{1, \ldots, n\} \tag{36}$$

This clause ensures that if there is a route from i to j by courier k, the visit order of j must be greater than the visit order of i by exactly 1, hence ensuring the continuous routes without any breaks.

$$\begin{aligned} &\forall k \in \{1, \ldots, m\}, \forall i \in \{1, \ldots, n+1\}, \forall j \in \{1, \ldots, n\},\, i \neq j \\ &route_{i,j,k} \implies visit\_order_{k,j} = visit\_order_{k,i} + 1 \end{aligned} \tag{37}$$

Symmetry Breaking Constraints

- **Symmetry Breaking by Distance**

  Symmetry breaking by distance is only applied when the distance matrix is symmetric, else the solver reverts to vanilla z3 configuration.

$$route_{n+1,1..n,k} > route_{1..n,n+1,k} \quad \forall k \in \{1, \ldots, m\} \tag{38}$$

  where the two slices of the route variable can be interpreted as boolean arrays encoding two integers.

- **Symmetry Breaking by Load**

  To eliminate duplicate solutions we impose an order between the assignments arrays. The order is numerical interpreting each row of assignment as an integer.

$$\begin{aligned} &\forall i, j \in \{1, \ldots, m\}, \quad \text{if} \quad capacities_i = capacities_j \\ &assignments_i \geq assignments_j \end{aligned} \tag{39}$$

4.4 Validation

Experimental Design The experiments are designed using z3 solver and SMT-LIB, which was integrated with python using the pysmt module [3]. For each evaluation we have created different configurations to evaluate instances, the different configurations are as follows:

1. Z3: Vanilla Z3 without symmetry breaking.
2. Z3 with Load SB: Z3 with symmetry breaking by Load.
3. Z3 with Distance SB: Z3 with symmetry breaking by distance.
4. Z3 with Distance and Load SB: Z3 with symmetry breaking by distance and Load.
5. SMT LIB using z3 solver: SMT LIB implementation for solver independent approach.
6. SMT LIB using CVC5: CVC5 implementation for solver independent approach.

We have performed several other experiments like limiting the number of objects carried per courier and defining heuristics to reduce the search space, but they were not reported as they never produced better results than the vanilla z3 configuration.

Instance	z3_direct	z3_load_sb	z3_dist_sb	z3_full_sb	z3_SMTLIB	cvc5_SMTLIB
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**	243	171
8	**186**	**186**	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**	**244**	**244**
13	1660	N/A	N/A	N/A	1622	N/A
16	1343	994	375	N/A	N/A	N/A

Table 3: Table showing results of the SMT Model

Experimental Results
Based on our experiments, we noticed that the vanilla Z3 solver is giving the best performance. It was also observed that solver-independent implementation reaches solutions more slowly.

5 MIP Model

The Mixed-Integer Linear Programming (MIP) model has been implemented using AMPL, a solver-independent language that allows using several commercial and non-commercial solvers. It has been integrated with Python through the amplpy library.
We experimented with 3 different encodings. The key difference lies in the size needed to store the routing variables. The first encoding uses a 3-dimensional binary array for the routing variables, the second uses a 2-dimensional integer array, and the last uses two 2-dimensional binary arrays to manage the routing plan [4].
Despite being the least efficient encoding in terms of encoding size, the first one outperformed both the alternative models across various solvers regardless of whether symmetry-breaking constraints were adopted or not, and was therefore selected to perform the final experiments that are shown at the end of this section.

5.1 Decision Variables

The model has the following decision variables:

- $\boldsymbol{x} \in \{0,1\}^{(n+1)\times(n+1)\times m}$ where $x_{i,j,k}$ equals 1 if courier k travels from node i to node j, 0 otherwise.

- $\boldsymbol{u} \in \mathbb{N}^n$ used for the Miller-Tucker-Zemlin formulation of the sub-tour elimination constraint.

- $\boldsymbol{maxDist} \in \mathbb{N} \cap [LB, UB]$ which represents the objective function, i.e. the maximum distance traveled among the m couriers.

5.2 Objective Function

The objective function of the problem can be formally defined as:

$$\max_{k \in \{1, \dots, m\}} (\sum_{i=1}^{n+1} \sum_{j=1}^{n+1} x_{i,j,k} \, D_{i,j}) \tag{40}$$

which in AMPL is obtained by constraining the $maxDist$ variable as follows:

$$\sum_{i=1}^{n+1} \sum_{j=1}^{n+1} x_{i,j,k} \, D_{i,j} \leq maxDist \qquad \forall \, k \in \{1, \dots, m\} \tag{41}$$

5.3 Constraints

- **Each customer is visited exactly once**

$$\sum_{k=1}^{m} \sum_{i=1}^{n+1} x_{i,j,k} = 1 \qquad \forall j \in \{1, \dots, n\} \tag{42}$$

- **All couriers start from and end at the depot**

$$\sum_{j=1}^{n} x_{n+1,j,k} = 1 \qquad \sum_{i=1}^{n} x_{i,n+1,k} = 1 \qquad \forall k \in \{1, \dots, m\} \tag{43}$$

- **Couriers must avoid self-loops**

$$x_{i,i,k} = 0 \qquad \forall k \in \{1, \dots, m\} \quad \forall i \in \{1, \dots, n+1\} \tag{44}$$

- **Flow conservation: couriers must leave nodes they enter**

$$\sum_{j=1}^{n+1} x_{i,j,k} = \sum_{j=1}^{n+1} x_{j,i,k} \qquad \forall k \in \{1, \dots, m\} \quad \forall i \in \{1, \dots, n+1\} \tag{45}$$

- **Capacity constraint: couriers must not exceed their load capacities**

$$\sum_{i=1}^{n+1} \sum_{j=1}^{n} s_j \, x_{i,j,k} \leq l_k \qquad \forall k \in \{1, \dots, m\} \tag{46}$$

- **Sub-tour elimination (MTZ formulation)**

$$\begin{aligned} \forall k \in \{1,\dots,m\} \quad \forall i \in \{1,\dots,n\} \quad \forall j \in \{1,\dots,n\} \qquad s.t. \quad i \neq j \\ u_j - u_i \geq s_j - Q(1 - x_{i,j,k}) \end{aligned} \tag{47}$$

where Q is the maximum load capacity. The u variables are bounded as follows:

$$s_j \leq u_j \leq Q \qquad \forall j \in \{1,\dots,n\} \tag{48}$$

Implied Constraints We discovered that enforcing that no courier remains idle improves the model's performance. This constraint is implied by the 'no self-loops' constraint and is formalized as follows:

Each courier delivers at least one item

$$x_{n+1,\,n+1,\,k} = 0 \qquad \forall\, k \in \{1,\dots,m\} \tag{49}$$

Symmetry Breaking Constraints For the MIP model, we considered the distance matrix symmetry, which applies only to instances from 11 to 21, and the hierarchical constraint type 1 taken from [2].
The latter imposes a hierarchical order on the assignment of customers to couriers having same load capacity. Specifically, it ensures that if customer i is visited by courier k, then at least one other customer with an index smaller than i is visited by courier k^*, namely the courier with maximum index among the ones with index smaller than k and same load capacity than k (if such courier exists). The constraints are formalized as follows.

Distance symmetry breaking

$$\begin{aligned} & i_k^* \leq j_k^* \qquad \forall\, k \in \{1,\dots,m\} \\ where \quad & i_k^* = i \in \{1,\dots,n\} \mid x[i, n+1, k] = 1 \\ & j_k^* = j \in \{1,\dots,n\} \mid x[n+1, j, k] = 1 \end{aligned} \tag{50}$$

Hierrachical constraint type 1 (HC1)

$$\begin{aligned} \sum_{i=1}^{n+1} x_{i,j,k} \leq \sum_{j'=1}^{j-1} \sum_{i=1}^{n+1} x_{i,j',k^*} \qquad \forall\, k \in \{2,\dots,m\} \quad \forall\, j \in \{2,\dots,n\} \\ where \quad k^* = max\{\hat{k} \mid l_{\hat{k}} = l_k,\ \hat{k} \in \{1,\dots,k-1\}\} \end{aligned} \tag{51}$$

5.4 Validation

Experimental Design We have experimented with all the available commercial solvers and some non-commercial ones, specifically CBC, HiGHS, Gurobi, Xpress, Cplex, Mosek and COPT. The models are run with and without symmetry breaking constraints to evaluate their contribution to the model performance. All the experiments are run including the implied constraint as it

proved to slightly improve the performance. Since some solvers performed very poorly, their results are not shown for conciseness. We show only CBC as non-commercial solver, and Gurobi and COPT as commercial ones.

Experimental Results
The CBC solver found the incorrect solution 435 for the 9th instance despite

Instance	CBC	CBC_sb	Gurobi	Gurobi_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	N/A	N/A	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**	**244**	**244**
12	N/A	N/A	N/A	N/A	712	925
13	656	768	516	530	1046	1336
16	N/A	N/A	**286**	**286**	494	445
19	N/A	N/A	**334**	**334**	694	698

Table 4: Table showing results of the MIP Model

the lower bound being 436, so it was discarded.

6 Conclusions

- The **CP** model is the only one capable of finding a solution for every instance, thus making it the best approach to the problem.
- The **MIP** model showed enough flexibility to certify some of the large instances, despite its heavy encoding.
- The **SMT** model was dragged down by its heavy encodings, which try to capture high-level logic at the cost of performance.
- The **SAT** model proved to be the weakest, in some cases not even finishing the encoding within the time limit.

In every approach despite our attempts at implementing effective symmetry-breaking constraints, no significant improvement was obtained. In some cases, it even worsened performance, probably due to slow propagation caused by the heavy encoding. In some cases. they make the solution space too sparse to actually help finding a solution within the time limit.

References

[1] Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. *Springer International Publishing*, 2018.

[2] Maryam Darvish, Leandro C. Coelho, and Raf Jans. Comparison of symmetry breaking and input ordering techniques for routing problems. Technical report, CIRRELT, Québec, QC, Canada, 2020.

[3] Marco Gario and Andrea Micheli. Pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms. In *SMT Workshop 2015*, 2015.

[4] S. Madankumar and C. Rajendran. A mixed integer linear programming model for the vehicle routing problem with simultaneous delivery and pickup by heterogeneous vehicles, and constrained by time windows. *Sādhanā*, 44:39, 2019.