Multiple Couriers Planning Problem Project Report

Mattia Buzzoni - mattia.buzzoni@studio.unibo.it
Mirko Mornelli - mirko.mornelli@studio.unibo.it
Riccardo Romeo - riccardo.romeo@studio.unibo.it

December, 2024

1 Introduction

This report outlines strategies for addressing the Multiple Couriers Planning Problem (MCPP) through four approaches: CP, SAT, SMT, and MIP. Each approach was developed by an individual team member, with the project taking approximately two and a half months to complete.

1.1 Input Variables

The input variables are the number of couriers m, the number of items n, the array of weights of each item s, the array of maximum load of each courier l and the matrix of distances D.

1.2 Objective Function

The goal of this project is to minimize the maximum distance traveled by any courier, achieved through an objective function defined as

$$\min(Obj)$$

where:

1. **Obj** represents the maximum total distance traveled by any courier.
2. For each courier $i \in \{0, \ldots, m-1\}$, the model constraints **Obj** to be at least as large as the total distance covered by that courier.

By minimizing **Obj**, the model balances the travel distances among couriers, preventing scenarios where one courier has a significantly longer route than others. This approach optimizes operations by evenly distributing the workload across the courier group.

1.3 Symmetry Breaking constraint

Despite of the technology used, we have identified symmetry in the search space. Specifically, when two couriers have identical transportable loads, the solver may produce duplicate solutions by simply swapping their load amounts. To break this symmetry, we introduce a constraint requiring that for every pair of couriers (i_1,i_2) with $i_1 \neq i_2$, one courier must carry a greater total load than the other.

2 CP Model

The CP model was built using the MiniZinc language and then tested with two different solvers.

2.1 Decision variables

The proposed MiniZinc model is based on the following variables:

- *assigned*: An array of integers of dimension N that specifies, for each item, the courier responsible for its delivery.
- *num_of_item*: An integer array of size M that specifies the number of items each courier is assigned to deliver.
- *max_num_of_item*: An integer representing the maximum items each courier can deliver, determined by the number of couriers and items, based on the following formula:

$$max\_num\_of\_item = N/M + 2$$

- *pos*: A matrix of size $N \times (max\_num\_of\_item + 2)$ indicating each courier's position at each time step. The two additional columns ensure that the courier starts and ends at the depot.
- *obj_dist*: This is an integer array of size N that specifies, for each courier, the distance traveled to deliver each assigned item.

2.2 Constraints

- Each route must start and end at the depot:

$$pos[1] = pos[n+2] = n+1$$

- A second constraint associates a specific delivery sequence with each courier, taking into account the courier's load capacity and the weight of each item:

$$bin\_packing\_capa(l, assigned, s)$$

- Total distance traveled by each courier based on the assigned items:

$$obj\_dist(k) = \sum_{i=1}^{max\_num\_of\_item+1} D[pos[k,i], pos[k,i+1]]; \forall k \in 1..m$$

- A constraint to determine the number of items delivered by each courier:

$$count\_eq(assigned, c, num\_of\_item[c]) \ \ \forall c \in 1..m$$

- For each courier specifies the positions of assigned items in the pos array: if the position is less than the total number of items to be delivered, the item's position is assigned; otherwise, it defaults to the depot position:

$$assigned[pos[c,i]] = c \ \ \forall c \in 1..m \wedge c \leq num\_of\_item[c] + 1$$

$$pos[c,i] = n+1 \ \ \forall c \in 1..m \wedge c > num\_of\_item[c] + 1$$

2.3 Validation

Several search strategies were tested, but the most effective proved to be the first fail strategy combined with a domain minimum hierarchy.

Inst	Gecode		Chuffed	
	NO_SB	SB	NO_SB	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	167
8	**186**	**186**	**186**	**186**
9	436	436	436	436
10	244	244	244	244
11	814	698	799	687
12	369	385	352	373
13	1196	1156	1346	1308
14	1218	1445	1151	1228
15	1469	N/A	1207	1207
16	286	382	286	345
17	N/A	N/A	1504	N/A
18	1067	1122	1033	980
19	334	334	335	362
21	912	817	860	784

Table 1: Results using the two solvers (Gecode and Chuffed), both with (SB) and without (NO_SB) symmetry breaking.

3 SAT Model

In this section we show the model used to solve the MCPP implemented in SAT. In particular, we decided to use the Z3-SAT solver, and also to explore two different search methods.

3.1 Decision variables

The SAT model is based on the following decision variables:

- X: this is a 3D matrix of dimension $m \times (n+1) \times (n+2)$, where the element $x_{i,j,k}$ is True if courier i is carrying item j at time k. This indicates that courier i visited the location j at moment k. Thus:

  $x_{i,j,k} \in \{True, False\}\ \forall i \in \{0, ..., m-1\}\ \forall j \in \{0, ..., n\}\ \forall k \in \{0, ..., n+1\}$

  In the case of fair load division among couriers we reduced the number of variables by decreasing the temporal index k.

- *current_loads*: this is a list of dimension $m \times n$ of boolean encoded numbers. Note that $current\_loads_{i,j}$ is a copy of the binary encoding of the size of the item j iff the courier i is carrying it.

- *total_current_load*: this is a list of dimension m of boolean encoded numbers. Consider that $current\_loads_i$ represents the total size carried by the $i-th$ courier.

- *current_distances*: this is a list of dimension $m \times (n+1)$ of boolean encoded numbers. Particularly, $current\_distances_{i,j}$ represents the distance traveled by the courier i in order to reach the $j-th$ position.

- *total_current_distances*: this is a list of dimension m of boolean encoded numbers. Observe that, $total\_current\_distances_i$ represents the total distance traveled by the $i-th$ courier.

3.2 Objective function

In the presented SAT model, the objective function is the Boolean representation of the longest distance traveled by any courier. Due to issues with Z3-SAT's native optimization method, we implemented two optimization strategies:

- **linear search**: The SAT solver is directed to identify solutions with a smaller objective function than the previously found one. This process continues until no better solution can be found, at which point the final solution is deemed the best

- **binary search**: We first establish upper and lower bounds for the objective function. The upper bound is set by the maximum distance a courier would travel if transporting all items alone, while the lower bound is the longest distance to deliver just one item across all combinations. In each iteration, we define a middle bound and instruct the SAT solver to find a solution with an objective function smaller than this middle bound. This effectively halves the search space, allowing us to continue searching in one half until we reach a point of unsatisfiability.

3.3 Constraints

3.3.1 Main Problem Constraints

Due to the decision variables, and the nature of the problem, we decided to implement the following constraints:

- It is obvious that each location must be visited and that it can be visited only once. Thus:

$$\bigwedge_{j=1}^{n} exactly\_one(X_{i,j,k} \mid i \in 0, ..., m-1, k \in 0, ..., n+1).$$

- We ensure that each courier starts its tour in base imposing that the courier is in position $j = 0$ at moment $k = 0$:

$$\bigwedge_{i=0}^{m-1} X_{i,0,0} \Longleftrightarrow True.$$

- Also, we impose that if a courier at moment k is in base, then it will be in base also for each moment $k+1$. This is true for all moments not equal to the initial one:

$$\bigwedge_{i=0}^{m-1} \bigwedge_{k=1}^{n+1} X_{i,0,k} \Longrightarrow X_{i,0,k+1}.$$

- No courier can have the gift of ubiquity and in each moment all couriers have to be in at least one position:

$$\bigwedge_{i=0}^{m-1} \bigwedge_{k=0}^{n+1} exactly\_one(X_{i,j,k} \mid j \in 0, ..., n).$$

- After leaving the base, each courier must return to it at any time:

$$\bigwedge_{i=0}^{m-1} at\_least\_one(X_{i,0,k} \mid k \in 0, ..., n+1).$$

- Given that, in all instances the number of items is greater than or equal to the number of couriers, it is reasonable to require that each courier delivers at least one item in order to reduce the search space:

$$\bigwedge_{i=0}^{m-1} at\_least\_one(X_{i,j,k} \mid j \in 1, ..., n, k \in 0, ..., n+1).$$

- In order to have that **current_loads** represents what said above, we have:

$$\bigwedge_{i=0}^{m-1} \bigwedge_{j=0}^{n} at\_least\_one(X_{x,j+1,k} \mid k \in 1, ..., n{+}1) \Rightarrow isEq(current\_loads_{i,j}, s_j).$$

Where s_j is the boolean representation of the size of the j-th item.

- Obviously, we imposed that each courier cannot carry a load greater than its possible total load l_i:

$$\bigwedge_{i=0}^{m-1} isLessEq(total\_current\_load_i,\ l_i).$$

- To have that **current_distances** represents what said above, we imposed:
  - copying, from D, the traveled distance at the start of the tour:

$$\bigwedge_{i=0}^{m-1} \bigwedge_{j=1}^{n} X_{i,j,1} \Rightarrow isEq(current\_distances_{i,j}, D_{n,j-1}).$$

  - copying, from D, the traveled distance for each pair of visited positions $(j_1,\ j_2)$:

$$\bigwedge_{i=0}^{m-1} \bigwedge_{k=1}^{n} \bigwedge_{j_1=1}^{n} \bigwedge_{j_2=1}^{n} (X_{i,j_1,k} \wedge X_{i,j_2,k+1}) \Rightarrow isEq(current\_distances_{i,j_2}, D_{j_1-1,j_2-1}).$$

  - copying, from D, the traveled distance at the end of the tour:

$$\bigwedge_{i=0}^{m-1} \bigwedge_{k=1}^{n} \bigwedge_{j_1=1}^{n} (X_{i,j_1,k} \wedge X_{i,0,k+1}) \Rightarrow isEq(current\_distances_{i,0}, D_{j_1-1,n}).$$

  - with the following constraint we ensure to have only Falses if the courier $i-th$ does not travel a certain distance:

$$\bigwedge_{i=0}^{m-1} \bigwedge_{j=1}^{n} \neg(at\_least\_one(X_{i,j,k}) \mid k \in 0, ..., n{+}1) \Rightarrow isEq(current\_distances_{i,j}, D_{0,0}).$$

- In order to impose the fact that the objective function must represent the longest distance traveled among couriers, we imposed this constraint:

$$isEq(obj\_function, max(\bigwedge_{i=0}^{m-1} total\_current\_distances_i)).$$

3.3.2 Symmetry Breaking Constraint

For the purpose of breaking the symmetry expressed in the Introduction section, we imposed the following constraint:

$$\bigwedge_{i_1=0}^{m-1} \bigwedge_{i_2=i_1+1}^{m-1} isEq(l_{i_1}, l_{i_2}) \Rightarrow isLessEq(total\_current\_load_{i_2}, total\_current\_load_{i_1}).$$

3.3.3 Fair Load Division Constraint

To reduce the search space, we aimed to distribute the objects among the couriers, which also decreases the number of decision variables to $k \in \{0, ..., n/m+2\}$. Consequently, we require each courier to carry at least n/m items:

$$\bigwedge_{i=0}^{m-1} at\_least\_K(X_{i,j,k} \mid j \in 1, ..., n,\ k \in 0, ..., n/m+2).$$

3.3.4 Encodings

- For the constraints: at_most_one (or K), at_least_one (or K), exactly_one (or K); we implemented five different encodings: naive pairwise, sequential, bitwise, Heule and native Z3 pseudo-boolean.
- In order to implement the equality constraint between two boolean representations (a and b) of numbers we decided to check for each "bit" if the bits are equal:
$$\bigwedge_{i=1}^{bits} a_i \Leftrightarrow b_i$$
- To implement the less or equal constraint we used the following encoding:
$$(\neg a_1 \vee b_1) \wedge (\bigwedge_{i=2}^{bits}((\bigwedge_{k=2}^{i} a_k \Leftrightarrow b_k) \Rightarrow (\neg a_i \vee b_i))$$
- We implemented the boolean summation d between two numbers a and b (with carry c) as follows:
  - $f_1 : \bigwedge_{i=1}^{bits} a_i \Leftrightarrow b_i \Leftrightarrow c_i \Leftrightarrow d_i$
  - $f_2 : \bigwedge_{i=1}^{bits} c_{i+1} \Leftrightarrow (a_i \vee b_i) \wedge (a_i \vee c_i) \wedge (b_i \vee c_i)$
  - $f_3 : \neg c_{bits} \wedge \neg c_0$

  Thus:
$$f_1 \wedge f_2 \wedge f_3 \wedge (\bigwedge_{i=1}^{bits} \neg a_i) \wedge (\bigwedge_{i=1}^{bits} \neg b_i)$$

3.4 Validation

In SAT we implemented different encodings, but in the following we show only the results obtained with the best one.

Inst	Z3-HEULE Encoding							
	LS				BS			
	F		NO_F		F		NO_F	
	SB	NO_SB	SB	NO_SB	SB	NO_SB	SB	NO_SB
1	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
2	**226**	**226**	226	226	**226**	**226**	**226**	**226**
3	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
4	220	220	220	220	**220**	**220**	**220**	**220**
5	**206**	**206**	**206**	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**	**322**	**322**	**322**	**322**
7	167	167	217	268	**167**	**167**	220	232
8	186	**186**	186	186	**186**	**186**	**186**	**186**
9	436	436	436	436	**436**	**436**	**436**	**436**
10	244	244	244	244	**244**	**244**	**244**	**244**
16	N/A	N/A	N/A	N/A	N/A	410	N/A	N/A

Table 2: Results for the SAT model (Heule encoding), with (SB) and without (NO_SB) symmetry breaking constraints, with (F), without (NO_F) fair load division among couriers and for both linear (LS) and binary search (BS).

4 SMT Model

For SMT, we used pySMT to implement a solver independent model. We decided to explore two search methods and we compared different known solvers.

4.1 Decision variables

We decided to use the following decision variables:

- X: This is a matrix of dimension $m \times (n+1)$. In particular, the element $X_{i,k} = j$ iff the courier i is in the position j at the step k. Thus:

$$x_{i,k} \in \{1, ..., n\}; \forall i \in \{0, ..., m-1\} \ \forall k \in \{0, ..., n+1\}$$

- *carried_load*: In this vector of dimension n we put the sum of all the sizes of the objects carried by all the couriers. In particular $carried_load_i$ will be equal to the summation of all the objects carried by the courier i.

$$carried\_load_i \geq 0; \forall i \in \{0, ..., m-1\}$$

- *traveled_distance*: This is a n-dimensional array representing the total distances traveled by all the couriers.

$$traveled\_distance_i \geq 0; \forall i \in \{0, ..., m-1\}$$

- *Obj*: This is an integer variable representing the maximum total distance traveled by any courier.

$$Obj \geq 0.$$

4.2 Objective function

For SMT the objective function is the same expressed in the Introduction section, while the search solution process adhere to the same principles described in the SAT section.

4.3 Constraints

4.3.1 Main Problem Constraints

For the SMT model we decided to implement the following constraints:

- First of all, in order to impose the field of X we imposed that each element $x \in X$ as to be: $1 \leq x \leq n+1$.
- In order to ensure that each courier starts its tour from base we impose to couriers to be in position $j = n+1$ at the moment $k = 0$:

$$\bigwedge_{i=0}^{m-1} X_{i,0} = n+1.$$

- The following constraint is used in order to impose to all couriers to return to the base sooner or later:

$$\bigwedge_{i=0}^{m-1} \bigvee_{k=0}^{n+1} X_{i,k} = n+1.$$

- Also we need to ensure that if a courier returned to base then it will stay on it without starting another tour:

$$\bigwedge_{i=0}^{m-1} \bigwedge_{k=1}^{n} (X_{i,k} = n+1) \implies (X_{i,k+1} = n+1).$$

- In order to have the vector *carried_load* representing what we said above we imposed the following constraints:

$$\bigwedge_{i=0}^{m-1} (carried\_load_i = \sum_{j=1}^{n} \sum_{k=1}^{n+1} ITE(X_{i,k} = j, s_{j-1}, 0)).$$

  In particular, we used if-then-else (ITE), thus we sum in *carried_load* the size s_{j-1} of the carried object, otherwise we sum zero. Note that, s_{j-1} is the size of the item in position $j-1$.

- Obviously, we imposed the bound on the total possible load transportable for each courier:

$$\bigwedge_{i=0}^{m-1} courier\_load_i \leq l_i.$$

Where l_i represents the maximum transportable load of the courier i.

- For the *traveled_distance* vector we used the same logic seen for *carried_load* imposing the two following constraints:

$$\bigwedge_{i=0}^{m-1} (traveled\_distance_i = \sum_{j=1}^{n+1} ITE(X_{i,1} = j, D_{n,j-1}, 0)).$$

$$\bigwedge_{i=0}^{m-1} (traveled\_distance_i = \bigwedge_{j_1=1}^{n+1} \bigwedge_{j_2=1}^{n+1} \sum_{k=1}^{n+1} ITE((X_{i,k} = j_1) \wedge (X_{i,k+1} = j_2), D_{j_1-1,j_2-1}, 0)).$$

The first constraint is used in order to take care of the tour starting point, while the second one is used to map the rest of the tour.

- In order to impose the fact that the objective function must represent the longest traveled distance among couriers we imposed the following constraint:

$$obj\_function = max(\bigwedge_{i=0}^{m-1} traveled\_distance_i).$$

4.3.2 Symmetry Breaking Constraint

With he intention of break this symmetry, we imposed the same constraint seen for SAT:

$$\bigwedge_{i_1=0}^{m-1} \bigwedge_{i_2=i_1+1}^{m-1} l_{i_1} = l_{i_2} \Rightarrow carried\_load_{i_2} < carried\_load_{i_1}.$$

4.3.3 Fair Load Division Constraint

This constraint is conceptually the same explained in SAT. In order to impose this "Fair Load Division" constraint we decided to use a Naive Pairwise encoding of the *at_least_k* constraint as follows:

$$\bigwedge_{i=0}^{m-1} at\_least\_K(X_{i,k} \neq n+1 \mid k \in 0, ..., n/m + 2).$$

The idea is that if a courier has to carry at least k elements it means that a certain number of elements of X must be different to the base $n+1$.

4.4 Validation

We tested the model with four solvers: MatSat, Yices, Z3, and CVC5. All solvers performed better with a fair load division among couriers. Thus, we present only the results obtained with this constraint and the two best-performing solvers.

Inst	CVC5				YICES			
	LS		BS		LS		BS	
	SB	NO_SB	SB	NO_SB	SB	NO_SB	SB	NO_SB
1	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**	**226**	**226**	**226**	**226**
3	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
4	**220**	**220**	**220**	**220**	**220**	**220**	**220**	**220**
5	**206**	**206**	**206**	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**	**322**	**322**	**322**	**322**
7	**167**	171	**167**	**167**	**167**	**167**	**167**	**167**
8	**186**	**186**	**186**	**186**	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**	**244**	**244**	**244**	**244**
12	N/A	N/A	N/A	N/A	N/A	752	N/A	592
13	1274	1458	1498	1498	1260	1178	1632	1392
16	N/A	373	N/A	296	374	286	301	**286**
19	N/A	595	N/A	595	N/A	502	N/A	441

Table 3: Results obtained with CVC5 and Yices solvers. Each solver solved the problem with both linear search (LS) and binary search (BS). We also tried to run the solvers with and without symmetry breaking constraint.

5 MIP Model

In this section, we present the MIP model developed to solve the problem. The model is implemented using solver-independent Google's OR-Tools library. The solvers utilized include Gurobi, CBC, SCIP, and HIGHS.

5.1 Decision variables

The MIP model is based on the following variables:

1. **X**: This is a three-dimensional binary matrix of dimension $m \times (n+1) \times (n+1)$. The element $X_{i,j,k}$ is True iff courier i travels from location j to location k, and False otherwise. The indices i, j, and k represent:
   - $i \in \{0, \dots, m-1\}$: The courier index.
   - $j, k \in \{0, \dots, n\}$: The locations in the network.

$$X_{i,j,k} \in \{0,1\}, \quad \forall i \in \{0,\ldots,m-1\}, \quad \forall j,k \in \{0,\ldots,n\}$$

2. **T**: This is a vector of integer variables of size n, where $T[k]$ represents the order in which a courier visits the location k. Each element of T indicates the position in the sequence of visits for each location, ensuring that all locations are visited in a unique order.

$$T[k] \in \{1,\ldots,n\}, \quad \forall k \in \{0,\ldots,n-1\}$$

3. **tot_dist**: This is a vector of integer variables of size m, where $\text{tot\_dist}[i]$ represents the total distance traveled by courier i. It accounts for the sum of all distances covered by courier i when completing the assigned routes.

$$\text{tot\_dist}[i] \geq 0, \quad \forall i \in \{0,\ldots,m-1\}$$

4. **Obj**: This is an integer variable representing the maximum total distance traveled by any courier.

$$\text{Obj} \geq 0$$

5.2 Objective function

We implemented the objective function exactly according to the Introduction.

5.3 Constraints

For the MIP model we decided to implement this set of constraints:

1. **Objective Bound Constraint**: this constraint ensures that Obj will always be at least as large as the longest individual distance traveled by any single courier:

$$\text{Obj} \geq \text{tot\_dist}_i, \quad \forall i \in \{0,\ldots,m-1\} \tag{1}$$

2. **Assignment Constraint**: each item must be assigned to exactly one courier, ensuring that every location is visited and that no location is duplicated in multiple couriers' routes. We enforce this by requiring that each location k has one incoming and one outgoing edge, indicating that each location is entered and exited precisely once:

$$\sum_{i=0}^{m-1} \sum_{j=0}^{n} X_{i,j,k} = 1, \quad \forall k \in \{0,\ldots,n-1\} \tag{2}$$

$$\sum_{i=0}^{m-1} \sum_{j=0}^{n} X_{i,k,j} = 1, \quad \forall k \in \{0,\ldots,n-1\} \tag{3}$$

3. **Depot Constraints**: each courier begins and ends their route at the depot, designated as location n. The equation (4) enforces that every courier has exactly one outbound trip from the depot, while the (5) ensures that each courier has exactly one inbound trip returning to the depot:

$$\sum_{j=0}^{n-1} X_{i,j,n} = 1, \quad \forall i \in \{0, \dots, m-1\} \tag{4}$$

$$\sum_{j=0}^{n-1} X_{i,n,j} = 1, \quad \forall i \in \{0, \dots, m-1\} \tag{5}$$

4. **No Self-Loop Constraints**: to prevent inefficient or illogical paths in which a courier revisits the same location immediately, we enforce that no courier can travel from a location j directly back to itself. This constraint ensures logical progression in each courier's route:

$$X_{i,j,j} = 0, \quad \forall i \in \{0, \dots, m-1\}, \quad \forall j \in \{0, \dots, n-1\} \tag{6}$$

5. **Flow Conservation Constraints**: these constraints guarantee continuity in each courier's route. Specifically, they ensure that for every location j on a courier's path, the number of inbound trips matches the number of outbound trips, thereby creating a connected route. For each courier, if they arrive at a location, they must also depart from it:

$$\sum_{k=0}^{n} X_{i,k,j} = \sum_{k=0}^{n} X_{i,j,k}, \quad \forall i \in \{0, \dots, m-1\}, \quad \forall j \in \{0, \dots, n-1\} \tag{7}$$

6. **Load Limit Constraints**: this constraint ensures that the total load, represented by the size s_j of each item at location j, does not exceed the courier's maximum load capacity l_i:

$$\sum_{j=0}^{n} \sum_{k=0}^{n-1} X_{i,j,k} \cdot s_j \leq l_i, \quad \forall i \in \{0, \dots, m-1\} \tag{8}$$

7. **Item Assignment Bounds**: to maintain balanced assignments, we aim to keep each courier's item count close to the average, avg_items. By defining bounds with parameters δ (upper) and γ (lower), we set acceptable limits for each courier's assignments:

$$\text{min\_items} = \text{avg\_items} - \gamma, \quad \text{max\_items} = \text{avg\_items} + \delta \tag{9}$$

$$\text{min\_items} \leq \sum_{j=0}^{n-1} \sum_{k=0}^{n} X_{i,j,k} \leq \text{max\_items}, \quad \forall i \in \{0, \dots, m-1\} \tag{10}$$

8. **Visit Sequence Constraints**: this constraint ensures that couriers visit locations in a defined sequential order, starting from the depot and adhering to the delivery route. These constraints use a large constant, bigM, to control when conditions should be active. Here's a breakdown:

   (a) **Sequential Order Constraints**: for every courier i traveling from location k to location j, the visit order $T[j]$ must be exactly one more than $T[k]$. To enforce this condition, two constraints are defined:

   - **Lower Bound Constraint:** ensures that $T[j]$ is at least $T[k]+1$ when $X[i,k,j]=1$:

   $$T[j] \geq T[k] + 1 - \text{bigM} \times (1 - X[i,k,j])$$

   - **Upper Bound Constraint:** ensures that $T[j]$ is at most $T[k]+1$ when $X[i,k,j]=1$:

   $$T[j] \leq T[k] + 1 + \text{bigM} \times (1 - X[i,k,j])$$

   Together, these constraints enforce the proper sequence of visits between any two locations, while the use of bigM disables the constraints if $X[i,k,j]=0$ (i.e., the route is not taken).

   (b) **Visit Order Domain Constraints**: To ensure that all visit orders $T[k]$ remain valid, the following domain constraints are applied:

   $$T[k] \geq 1 \quad \text{and} \quad T[k] \leq n$$

   This ensures that every location is visited exactly once in the sequence.

9. **Distance Calculation Constraints**: for each courier i, the total distance traveled, denoted as tot_dist$[i]$, is calculated by summing the distances $D[j][k]$ for every pair of locations j and k where the courier i travels directly from j to k.

$$\text{tot\_dist}[i] = \sum_{j=0}^{n}\sum_{k=0}^{n} X[i,j,k] \cdot D[j][k], \quad \forall i \in \{0,\ldots,m-1\}$$

10. **Symmetry Breaking Constraints**: in an effort to break the symmetry explained in the Introduction section, we add the following constraint for each courier i up to $m-2$:

$$\forall i \in \{0,\ldots,m-2\}, \quad l_i = l_{i+1} \implies \text{tot\_dist}[i] \leq \text{tot\_dist}[i+1].$$

5.4 Validation

Inst	CBC		SCIP		HIGHS		GUROBI	
	NO_SB	SB	NO_SB	SB	NO_SB	SB	NO_SB	SB
1	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
2	**226**	226	**226**	**226**	**226**	**226**	**226**	**226**
3	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
4	220	220	**220**	**220**	**220**	**220**	**220**	**220**
5	**206**	**206**	**206**	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**	**322**	**322**	**322**	**322**
7	209	186	167	167	**167**	**167**	**167**	**167**
8	**186**	**186**	**186**	**186**	**186**	**186**	**186**	**186**
9	436	436	436	436	**436**	**436**	**436**	**436**
10	244	244	244	244	**244**	**244**	**244**	**244**
11	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1074
12	N/A	N/A	N/A	N/A	N/A	N/A	742	881
13	N/A	N/A	N/A	N/A	N/A	N/A	474	476
16	N/A	N/A	N/A	N/A	N/A	N/A	286	286
19	N/A	N/A	N/A	N/A	N/A	N/A	396	363

Table 4: Results obtained with CBC, SCIP, HIGHS and GUROBI as solvers. Each solver solved the problem with and without symmetry breaking constraint.

6 Conclusions

Regarding the structure of the presented models, we observed that, regardless of the technology used, imposing a load division among couriers helps solvers find solutions more quickly.

The best-performing technology was CP implemented in MiniZinc, which allowed us to find solutions for significantly more instances compared to other technologies.
On the other hand, SMT and MIP performed similarly in terms of the number of optimal solutions found and the number of encoded instances.
SAT, however, proved to be the least effective technology, as it encoded fewer instances.

In CP, Chuffed outperformed Gecode in more complex problems, while Gecode achieved optimal solutions faster in simpler cases.
For SAT, the Heule encoding allowed for solutions in more instances, but it did not consistently find optimal solutions in simpler cases.
In SMT, the Yices solver demonstrated superior performance in terms of optimality, speed, and the number of problems solved.
In MIP, HIGHS and GUROBI found the most optimal solutions, with GUROBI also capable of solving some of the more complex instances.