CDMO - report paper

Babboni Luca (luca.babboni2@studio.unibo.it)
Nanni Gabriele (gabriele.nanni4@studio.unibo.it)
Tedeschini Luca (luca.tedeschini3@studio.unibo.it)

December 8, 2024

1 Introduction

This report outlines our approach to solving the Multiple Couriers Planning Problem. We implemented all four proposed formulations: CP, SAT, SMT, and MIP. By applying each model, we achieved optimal solutions for the first 10 instances (expect instance 7 for SAT), and obtained some certified optimal solutions for instances beyond the tenth.
We collaborated to develop ideas for the structure of the encodings for all models. During implementation, we worked together extensively, with each member specializing in specific areas: Gabriele Nanni focused on CP and SAT encoding, Luca Babboni on CP and MIP encoding, and Luca Tedeschini on SMT and MIP encoding. While Luca Babboni primarily maintained the software structure, all team members contributed to its development.

2 Upper e Lower Bound

Estimating upper and lower bounds is essential in optimization problems as it defines the feasible range for the objective value, guiding the search process. By constraining the objective value, we reduce the solution space, making the problem more manageable and improving computational efficiency. We applied consistent lower and upper bound approximations across all techniques to ensure comparability of results.

2.1 Lower Bound

As a lower bound for the objective function, we consider the minimum distance couriers must travel, calculated by evaluating the distances from the depot to each client and back. The maximum of these distances is taken as the lower bound, representing the minimum distance a vehicle must travel to serve the furthest client. This lower bound is validated by the triangular inequality, which states that the direct distance between two points is always less than or equal to the sum of the distances when traveling through an intermediate point.

Additionally, we have included a constraint for the minimum distance that each courier must travel. This distance is determined by finding the shortest round trip to the client closest to the depot. This constraint is always valid due to the triangular inequality, ensuring that the minimum travel distance is accurately represented.

2.2 Upper Bound

The upper bound is calculated using a Nearest Neighbor greedy heuristic [1] algorithm (pseudocode in Algorithm (1)). The process begins by selecting the depot as the starting node and marking it as visited. The algorithm then iteratively selects the nearest unvisited node, adds the distance to the total cost, and marks the node as visited. This process continues until all nodes have been visited. Finally, the algorithm returns to the starting node to complete the tour, adding this final distance to the total cost. The resulting total cost represents the upper bound, providing an estimate of the maximum distance or cost required to visit all nodes in the problem.

2.3 Implementation

After defining upper and lower bound this way we consider a constraint common to all models:

$$\text{low\_bound} \leq \text{max\_distance} \leq \text{up\_bound} \tag{1}$$

The **Bounds Constraint** (1) ensure that *max_distance* remains within the lower and upper bound limits.

3 MTZ Subtour Elimination

The MTZ (Miller-Tucker-Zemlin) formulation[2] is a classic method for eliminating subtours in optimization problems, particularly in routing and sequencing contexts. It introduces auxiliary variables to represent the order or position of nodes within a tour, ensuring that no disjoint cycles can form. Constraints are added to enforce logical relationships between these variables, preventing subtours while maintaining feasibility.

4 CP Model

In our study, we drew inspiration from the model described in the cited paper [3]. This paper presents two models (Model A and Model B) that address the Vehicle Routing Problem with Time Windows (VRPTW) for a fleet of vehicles with homogeneous capacity.
Due to the heterogeneous capacities of our fleet and the absence of time constraints, we modified the original models. Experimental tests showed that Model

A significantly outperformed Model B. Consequently, we will describe the modified Model A in detail.

4.1 Decision variables

Vehicle \ Destination	1	2	3	4	5	6
1	2	6	3	4	5	1
2	1	2	6	4	5	3
3	1	2	3	6	4	5

Table 1: Successor matrix example (succ_matrix)

- *succ_matrix*$_{i,j}$: For each vehicle, the matrix includes an array of decision variables that represent the assigned route. A vehicle j delivers an item to client i if ($succ\_matrix_{ij} \neq i$). Given n clients in the instance, the matrix contains $n+1$ destinations, with the additional destination representing the depot. The exact route for each vehicle can be determined from the *succ_matrix* by starting at the depot node (last column) and following the assigned next destination until reaching $n+1$. For example looking at Table 1 the assigned path for the vehicle 1 is: $6 \rightarrow 1 \rightarrow 2 \rightarrow 6$. It is interesting to note that each route always start from the depot and ends at the depot.
- *loads* : The loads variable encodes the assignment of clients to vehicles. Specifically, loads[client] indicates which vehicle is responsible for delivering the item to that client. This variable ensures that each client is assigned to exactly one vehicle, and it is used to verify that the total load carried by each vehicle does not exceed its capacity.
- *couriers_loads* : The *couriers_loads* variable encodes the total load carried by each vehicle. Specifically, *couriers_loads*[vehicle] represents the sum of the sizes of all items assigned to that vehicle. This variable works togheter with *loads* to ensure that the load of each vehicle does not exceed its maximum capacity, thereby maintaining the feasibility of the solution with respect to vehicle capacities.

4.2 Objective function

$$\forall v \in \text{vehicles}, \quad \text{v\_dist}[v] = \sum_{i \in \text{nodes}} \text{distances}[i, \text{successor}[v, i]] \tag{2}$$

$$\text{Minimize} \max_{v \in \text{vehicles}} (\text{v\_dist}[v]) \tag{3}$$

The objective function (3) in this model aims to minimize the maximum distance traveled by any vehicle in the fleet. Mathematically, it is expressed as minimizing the maximum sum of distances between consecutive nodes in the route of each vehicle.

4.3 Constraints

$$\forall v \in \text{vehicles}, \quad \text{v\_dist}[v] \geq \text{min\_dist\_bound} \tag{4}$$

To minimize the search space as effectively as possible, we introduced the **Minimum Vehicle Distance Constraint** (4). This constraint serves as a lower bound for the distance each vehicle must travel.

$$\forall v \in \text{vehicles}, \quad \text{subcircuit}(\text{successor}[v, \text{nodes}]) \tag{5}$$

The **Subcircuit Constraint** (5) ensures that each vehicle's route forms a closed loop, meaning the vehicle returns to its starting point after visiting all assigned clients. The nodes variable represents all the locations in the problem, including both the clients and the depot, with the depot being an additional node beyond the number of clients.

$$\forall v \in \text{vehicles}, \quad \text{alldifferent}(\text{successor}[v, \text{nodes}]) \tag{6}$$

The **All-Different Constraint** (6) ensures that each node is visited exactly once by each vehicle, preventing multiple visits to the same node. This constraint is crucial for maintaining the uniqueness of each client's visit within the vehicle routing problem. Although it is implied in the subcircuit constraint, we observed a slight performance improvement by explicitating this redundancy.

$$\text{bin\_packing\_capa}(\text{capacity}, \text{loads}, \text{size}) \tag{7}$$

The **Bin Packing Constraint** (7) ensures that the total size of items assigned to each vehicle does not exceed its capacity

$$\forall v \in \text{vehicles}, \quad \text{capacity}[v] = \sum_{c \in \text{clients}} (\text{size}[c] \cdot (\text{loads}[c] = v)) \tag{8}$$

$$\forall c \in \text{clients}, \quad \sum_{v \in \text{vehicles}} (\text{loads}[c] = v) = 1 \tag{9}$$

The **Load Capacity Constraint** (8) ensures that the total load carried by each vehicle does not exceed its maximum capacity. The **Item Assignment Constraint** (9) ensures that each client is assigned to exactly one vehicle, preventing any client from being served by multiple vehicles. These two constraints are fundamental as they propagate the information derived from the bin packing constraint within the successor matrix.

$$\forall v \in \text{vehicles}, \quad \text{successor}[v, \text{num\_clients}+1] \neq \text{num\_clients}+1 \tag{10}$$

$$\forall v \in \text{vehicles}, \quad \sum_{c \in \text{clients}} (\text{successor}[v, c] = \text{num\_clients}+1) = 1 \tag{11}$$

The **Depot Departure Constraint** (10) ensures each vehicle departs from the depot to begin its route. While theoretically redundant, since the optimal solution requires all vehicles to depart, explicitly declaring this constraint slightly improved solving time. The **Depot Return Constraint** (11) ensures that each vehicle returns to the depot after completing its route.

$$\forall v \in \text{vehicles}, \quad \text{successor}[v][\text{num\_clients}+1] < \text{argmax}(\text{successor}[v][..]) \tag{12}$$

To further reduce the solution space, we implemented the **Path Symmetry Breaking Constraint** (12).This constraint is applied only when the distance matrix is symmetric (distances$[i, j]$ = distances$[j, i]$). It ensures that symmetric routes, such as (depot $\rightarrow 1 \rightarrow 4 \rightarrow 2 \rightarrow$ depot) and (depot $\rightarrow 2 \rightarrow 4 \rightarrow 1 \rightarrow$ depot), are considered only once. Specifically, we prioritize routes where the first package delivered has a smaller index than the last package delivered.

4.4 Validation

In the context of the CP model, we explored various search strategies. The search method optimized for the first 10 instances (s1) does not incorporate any annotations that introduce randomness (with the exception of *in_domain_random*, that helps in the search of the optimal solution for instance 7). This approach is adopted because, in smaller instances, it is feasible to explore the entire search space, and incorporating randomness or restarting the search introduces overhead, thereby preventing the model from achieving optimality.
For the later instances, our objective shifts to finding feasible solutions due to the vast search space resulting from the large instances. Consequently, we introduce randomness through annotations (s2), alternating between exploring the neighborhood of the current solution and restarting the search when outside a valid neighborhood. This strategy enables us to explore more areas of the search space in the quest for solutions.
Furthermore, symmetry-breaking constraints do not significantly influence the search process. However, we observed that adding excessive constraints overly restricts the search space, leading to the model failing to find solutions within the five-minute time-frame we were asked to respect. For this reason the low performing models with multiple symmetry-breakings constraint were discarded.
Chuffed, converting the instance into SAT, is extremely efficient on the smaller instances, but has an explosion in search space dimension reaching the more complex ones.

	Base model		Gecode s1		Gecode s2		Gecode no sb s2		Chuffed	
Inst. n.	obj	sec	obj	sec	obj	sec	obj	sec	obj	sec
1	**14**	0	**14**	0	14	-	14	-	**14**	0
2	**226**	32	**226**	0	**226**	0	**226**	0	**226**	0
3	**12**	0	**12**	0	12	-	12	-	**12**	0
4	220	-	**220**	0	**220**	0	**220**	0	**220**	0
5	**206**	0	**206**	0	**206**	0	**206**	0	**206**	0
6	**322**	77	**322**	0	**322**	0	**322**	0	**322**	0
7	167	-	**167**	0	**167**	0	**167**	0	**167**	1
8	**186**	191	**186**	0	**186**	0	**186**	0	**186**	0
9	436	-	**436**	0	**436**	0	**436**	0	**436**	0
10	244	-	**244**	0	**244**	0	**244**	0	**244**	0
11	704	-	806	-	402	-	402	-	-	-
12	481	-	652	-	**346**	7	**346**	6	-	-
13	-	-	1176	-	546	-	546	-	1226	-
14	1227	-	1358	-	539	-	539	-	-	-
15	-	-	1346	-	616	-	616	-	-	-
16	308	-	-	-	**286**	3	**286**	3	1310	-
17	-	-	1646	-	865	-	865	-	-	-
18	1016	-	1244	-	515	-	515	-	-	-
19	455	-	671	-	**334**	4	**334**	4	-	-
20	-	-	1626	-	775	-	775	-	-	-
21	908	-	985	-	498	-	498	-	-	-

Table 2: The table compares various models and search strategies used for the CP.

5 SAT model

For the SAT method, we approached the problem using a $3D$ matrix to represent the routes and built auxiliary variables to assert constraints. We developed a model with the Z3 solver implementation in Python, utilizing only Boolean variables or Z3 Boolean expressions and references. This method proved to be the least effective due to the high computational cost of encoding all Boolean variables and the extensive space required for multidimensional matrices. Although the model successfully solved the first ten instances, it fails to certify the optimality for the seventh. It also fails to provide solutions for the larger instances (11-21).

5.1 Decision variables

- $routes_{i,j,k}$: A tridimensional matrix that for each courier i determines if it travels from j to k. If it is the case the element $routes_{i,j,k}$ will be set to True. A similar encoding is also being used in SMT ans MIP model and a graphical representation can be seen in Figure 1.

- $assignments_{i,j}$: A bi-dimensional matrix that encodes if the j_{th} item has been assigned to the i_{th} courier.
- $order_{j,k}$ A bi-dimensional matrix that encodes the order of the deliveries: order_{jk} is True if the j_{th} item is delivered as k_{th} by its assigned courier.
- $courier\_loads_i$: the actual load carried by each courier. It is concretely a matrix, but this is only because of the binary nature of the loads. Abstractly it is a list of the loads for each courier.

5.2 Objective function

$$\text{Minimize} \max_{v \in \text{vehicles}} (\text{dist}[v]) \tag{13}$$

Where the distances are computed by the model in binary form and then reconverted to integer numbers to extract the maximum. The constraint for the optimality search converts the *obj_value* obtained to a binary list and then it adds the new constraint to the model each time it finds a solution.

5.3 Constraints

The constraints used in the SAT model impose that each load is less or equal than the courier capacity and that the routes are viable ones.

$$\forall j \in \text{items}, \quad \text{exactly\_one}(assignments[..][j]) \tag{14}$$

The **Unique Courier Assignment Constraint** (14) ensure that each package is properly managed. This verify that exactly one courier is assigned to each item so that all packages are accounted for and not more than once.

$$\begin{aligned} \forall i \in \text{couriers}, \quad & \text{sum\_K\_bin}(assignments[i][..], size\_bin, courier\_loads[i]) \\ & \text{binary\_number\_leq}(courier\_loads[i], capacity[i]) \end{aligned} \tag{15}$$

The **Load Capacity Constraint** (15) ensures that the load assigned to each courier does not exceed their capacity.

$$\forall i \in \text{couriers}, \quad \text{at\_least\_one}(assignments[i][..]) \tag{16}$$

The **Minimum Assignment Constraint** (16) ensures that each courier is assigned to least one package. This is an implied constraint, as the number of couriers is always less than the number of items and, to minimize the maximum distance traveled, it is essential that every courier handles at least one item.

$$\forall j \in \text{items}, \quad \text{exactly\_one}(order[j][..]) \tag{17}$$

The **Unique Delivery Constraint** (17) ensures that each item appears as

”True” only once in the order matrix, guaranteeing that no package is delivered more than once.

$$\forall i \in \text{couriers}, \forall j \in \text{items}, \quad \text{And}(\text{Not}(routes[i][j][j]) \tag{18}$$

The **Self-Loop Avoidance Constraint** (18) ensures that couriers do not travel from their departure point back to the same point. This prevents self-loops and avoids creating ”ghost” routes where travel occurs from the depot back to the depot.

$$\begin{aligned} \forall i \in \text{couriers}, \forall j \in \text{items}, \text{assignments[i][j]} &\implies \text{exactly\_one}(routes[i][j][..]) \\ \forall i \in \text{couriers}, \forall k \in \text{items}, \text{assignments[i][k]} &\implies \text{exactly\_one}(routes[i][..][k]) \end{aligned} \tag{19}$$

The **Route Continuity Constraint** (19) ensures that for each courier, if an item is assigned, there must be a specified origin and destination. This guarantees that every assigned item has a clear route from its starting point to its endpoint. This helps exclude all possible combinations that do not form valid tours.

$$\begin{aligned} \forall i \in \text{couriers} \quad & \text{exactly\_one}(routes[i][n][..]) \\ \forall i \in \text{couriers} \quad & \text{exactly\_one}(routes[i][..][n]) \end{aligned} \tag{20}$$

The **Depot Departure Constraint** (20) ensures that couriers leave the depot and come back. This is an implied constraint, as other constraints already ensure this.

$$\begin{gathered} \forall i \in \text{couriers}, \forall j \in \text{items}, \forall k \in \text{itemsroutes[i][j][k]} \implies \text{is\_next}(order[j][..], order[k][..]) \\ \forall i \in \text{couriers}, \forall j \in \text{itemsroutes[i][n][j]} \implies order[j][0] \end{gathered} \tag{21}$$

The **Order of Delivery Constraint** (21) ensures that for each courier, the sequence of deliveries is respected. Specifically, for each path ($j \to k$), the order of delivery must be maintained. Additionally, if the path is from the depot to j (meaning that j is the first client served by the vehicle), then j must be the first package delivered.

$$\forall i \in \text{couriers}, \quad \text{sum\_K\_bin}(flat\_r[i], flat\_d\_bin, distances[i]) \tag{22}$$

The **Total Distance Constraint** (22) ensures that the total distance traveled by each courier is the sum of the distances of all routes taken.

5.4 Validation

Inst n.	SAT base	SAT binary search
1	**14**	**14**
2	**226**	**226**
3	**12**	**12**
4	**220**	**220**
5	**206**	**206**
6	**322**	**322**
7	342	254
8	**186**	**186**
9	**436**	**436**
10	**244**	**244**

Table 3: Results for the SAT models

In our study using SAT encoding, we found that binary search slightly outperforms linear search by efficiently narrowing down the search space at each iteration preferring solution in the lower half of the search space, unlike linear search which may find solutions nearer to the upper bound. The seventh instance was particularly challenging due to the significant expansion of the search space caused by the quadratic scaling of the variables like the variable *routes* with the number of items, compared to the linear scaling with the number of couriers. This highlights the efficiency of binary search in handling larger and more complex instances. Considering larger instances it is not able to find solutions in the 300 seconds time limit.

6 SMT model

The final SMT model is entirely based on a $3D$ tensor representation, meaning that all the constraints leverage this structure in their formulation. This model is also widely used in literature ([4]). We developed two models using the Python implementation of the Z3 solver. The first model was straightforward in its implementation; we did not identify any symmetry-breaking constraints, and while it worked well for the first 10 instances, it fails to solve anything beyond the tenth.
The second and final model is slightly more complex and introduces an additional decision variable that functions as a *knapsack*. Using this auxiliary variable, we were able to simplify two constraints, thereby improving performance. This improvement is noticeable only in the seventh instance, because the model still fails to solve instances beyond the tenth.

6.1 Constant Conventions

In the following sections, we adopt specific conventions to refer to the parameters of the problem: n represents the number of vehicles, m denotes the number of clients, $package\_size\_i$ is an array containing the size of the i_{th} package, and $vehicle\_capacity\_k$ is an array containing the capacity of the k_{th} vehicle.

6.2 Decision Variables

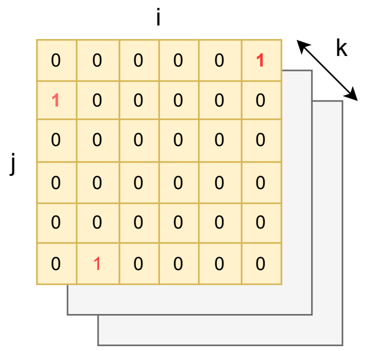

Figure 1: Representation of the $paths_{i,j,k}$ decision variable used. The represented path for the first vehicle is Depot $\rightarrow 1 \rightarrow 2 \rightarrow$ Depot.

- $paths_{i,j,k}$**:** A tensor representation where each matrix $(i \times j)$ describes the route of a single courier. The additional dimension $k = n$ stacks these matrices to form the complete data structure for all couriers. You can see a representation of this matrix in the figure (1).
- $y_{i,k}$**:** A boolean matrix with m rows and n columns representing, for each vehicle, which client is assigned to it.
- $numvisit_i$**:** The variable related to the MTZ subtour([2]) elimination strategy.
- $maxdist$**:** The maximum distance traveled by any courier.

6.3 Objective function

The objective function (23) is defined as follows:

$$\begin{gathered}\sum_{i=1}^{m+1}\sum_{j=1}^{m+1} \text{paths}_{i,j,k} \cdot \text{distances}_{i,j} = d_k \quad \forall k \in 1 \dots VEHICLES \\ \min(\max(d_k)) \forall k \in 1 \dots VEHICLES\end{gathered} \tag{23}$$

6.4 Constraints

The constraints in our SMT model are designed to accurately capture the requirements of the multi-courier routing problem, leveraging various SMT theories for efficient and correct solutions. Specifically, we use the *Linear Integer Arithmetic* (LIA) theory for range-based constraints and arithmetic operations, the *Bit-Vector* (BV) theory for logical manipulations and binary assignments, and the *Array* (A) theory for tensor and matrix representations. This combination ensures the model's flexibility, scalability, and rigor to address the problem's complexity.

$$0 \leq numvisit_i \leq m-1 \quad \forall i \in 1 \ldots n \tag{24}$$

The **Decision Variable Domain Constraint** (24) ensures that the number of visits $numvisit_i$ for each node i is between 0 and $m-1$, adhering to the valid range for visits.

$$\sum_{i}^{m+1} paths_{i,j,k} = \sum_{i}^{m+1} paths_{j,i,k} \quad \forall j \in 1 \ldots m, \forall k \in 1 \ldots n \tag{25}$$

The **Closed Circuit Constraint** (25) guarantees that every visited node has a valid incoming and outgoing connection, ensuring a closed circuit for each vehicle.

$$paths_{i,j,k} \implies numvisit_i < numvisit_j \quad \forall k \in 1 \ldots n, \forall i,j \in 1 \ldots m \tag{26}$$

The **MTZ Constraint** (26) ensures that if an edge connects nodes i and j, node i must be visited before node j. This constraint is applied to eliminate subtours from the possible solutions. For a more detailed explanation, refer to paragraph 3.

$$\sum_{i}^{m+1} paths_{i,i,k} = 0 \quad \forall k \in 1 \ldots n \tag{27}$$

The **Self-Loops Avoidance Constraint** (27) eliminates self-loops, ensuring that a node does not have an edge to itself.

$$\sum_{i}^{m} paths_{i,m,k} = 1 \wedge \sum_{j}^{m} paths_{m,j,k} = 1 \quad \forall k \in 1 \ldots n \tag{28}$$

The **Depot Constraint** (28) ensures that each vehicle starts and ends its route at the depot.

$$\sum_{i}^{m+1} paths_{i,j,k} = y_{j,k} \quad \forall j \in 1 \ldots m, \forall k \in 1 \ldots n \tag{29}$$

The **Channeling Constraint** (29) connects the tensor representation of the solution to the knapsack matrix. This linkage allows us to apply additional constraints directly to the smaller knapsack matrix, resulting in faster verification.

$$\sum_{i}^{m} y_{i,k} * package\_size_i \leq vehicles\_capacity_k \quad \forall k \in 1 \dots n \tag{30}$$

The **Capacity Constraint** (30) checks whether the knapsack assignment to a specific vehicle does not exceed its capacity.

$$\sum_{k}^{n} y_{j,k} = 1 \quad \forall j \in 1 \dots m \tag{31}$$

The **Uniqueness Constraint** (31) imposes that each client is served by only one vehicles.

6.5 Validation

Inst n.	SMT base	SMT auxiliary variable
1	**14**	**14**
2	**226**	**226**
3	**12**	**12**
4	**220**	**220**
5	**206**	**206**
6	**322**	**322**
7	**167**	**167**
8	**186**	**186**
9	**436**	**436**
10	**244**	**244**

Table 4: Results for the SMT models

The table above presents our results, demonstrating that we achieved optimality for all of the first ten instances, indicating strong performance from both models. Notably, the solving time for instance 7 was significantly different: the base model took 19 seconds, while the improved model took only 13 seconds, suggesting the latter's superiority. However, for more challenging instances and within a limited time of 300 seconds, neither models could solve any instance beyond the tenth.

7 MIP Model

This model, similar to the previous one, benefits from the superior performance of the Gurobi solver, allowing us to find optimality in instances beyond the tenth. Additionally, we implemented the same model in MiniZinc and without redundant constraint for comparison, and their performance will be discussed in the validation subsection.

7.1 Decision variables

- $x_{i,j,k}$: A binary variable that equals 1 if vehicle k travels directly from node i to node j, and 0 otherwise. This variable encodes the routes taken by each vehicle. A graphical example is shown in (1).
- $y_{i,k}$: A binary variable that equals 1 if vehicle k visits node i, and 0 otherwise. This variable ensures that each node is visited by exactly one vehicle.
- $d_{i,k}$: A continuous variable representing the distance traveled by vehicle k to reach node i. This variable is used in the subtour elimination constraint to prevent cycles within the routes, using the MTZ formulation 3.
- $maxdist$: A continuous variable representing the maximum distance traveled by any vehicle. The objective is to minimize this variable to ensure that the longest route taken by any vehicle is as short as possible.

7.2 Optimizer Flags

Gurobi offers various optimization flags that can significantly impact performance. After extensive testing, we selected the following flags for our specific use case: setting Threads to 1 to ensure single-thread usage, MIPFocus to 1 to prioritize finding feasible solutions quickly, ImproveStartTime to 200 allows the solver to switch to a "solution improvement strategy" after 200 seconds, Presolve to 1 to activate a conservative presolve strategy, and Cuts to 1 to enable a "moderate" cut generation strategy. These settings collectively enhanced the efficiency and effectiveness of our optimization process. A more complete explanation can be found on the official page.

7.3 Objective function

The objective function is the following:

$$\begin{gathered}\sum_{i}^{NODES}\sum_{j}^{NODES} distances_{i,j} * x_{i,j,k} = d_k \quad \forall k \in 1 \dots VEHICLES \\ \min(\max(d_k)) \forall k \in 1 \dots VEHICLES\end{gathered} \tag{32}$$

7.4 Constraints

$$\sum_{k \in \text{VEHICLES}} y_{i,k} = 1, \quad \forall i \in \text{CLIENT} \tag{33}$$

The **Single Assignment Constraint** (33) ensures that each client is assigned to exactly one vehicle.

$$\sum_{j \in \text{NODES}} \sum_{k \in \text{VEHICLES}} x_{i,j,k} = 1, \quad \forall i \in \text{CLIENT} \tag{34}$$

$$\sum_{i \in \text{NODES}} \sum_{k \in \text{VEHICLES}} x_{i,j,k} = 1, \quad \forall j \in \text{CLIENT} \tag{35}$$

The **Single Arrival Constraint** (34) ensures that each client is visited as an arrival point from exactly one currier. The **Single Departure Constraint** (35) ensures that from each client only one currier continues the path.
These two constraints are redundant as they perform checks that are implicitly covered by Constraint (33). However, we have included them in our model because they significantly reduce the search space, thereby enhancing the overall performance of the optimization process for some instances.

$$\sum_{i \in \text{CLIENT}} y_{i,k} \cdot \text{packages\_size}[i] \leq \text{vehicles\_capacity}[k], \quad \forall k \in \text{VEHICLES} \tag{36}$$

The **Vehicle Capacity Constraint** (36) ensures that the total size of packages assigned to a vehicle must not exceed the vehicle's capacity.

$$\sum_{j \in \text{NODES}} x_{j,j,k} = 0, \quad \forall k \in \text{VEHICLES} \tag{37}$$

The **Self-Loops Avoidance Constraint** (37) prevents a vehicle from traveling from a node back to itself, avoiding loops.

$$\sum_{i \in \text{NODES}} x_{i,n,k} = 1, \quad \forall k \in \text{VEHICLES} \tag{38}$$

The **Return To Depo Constraint** (38) ensures that the last visited node for every vehicle is the depot.

$$\sum_{j \in \text{NODES}} x_{i,j,k} = \sum_{j \in \text{NODES}} x_{j,i,k}, \quad \forall i \in \text{CLIENT}, \forall k \in \text{VEHICLES} \tag{39}$$

The **Route Continuity Constraint** (39) encodes that if a vehicle enters a node, it must also exit the node, ensuring continuity in the route.

$$\sum_{j \in \text{NODES}} x_{j,i,k} = y_{i,k}, \quad \forall i \in \text{CLIENT}, \forall k \in \text{VEHICLES} \tag{40}$$

The **The Channeling Constraint** (40) connects the tensor $x_{i,j,k}$ to the decisional variable $y_{i,k}$. If a client is visited by a vehicle in the tensor $x_{i,j,k}$ then the association is propagated in $y_{i,k}$.

$$d_{i,k} - d_{j,k} + \text{num\_clients} \cdot x_{i,j,k} \leq \text{num\_clients} - 1, \\ \forall k \in \text{VEHICLES}, \forall i, j \in \text{CLIENT}, i \neq j \tag{41}$$

The **Subtour Elimination Constraint** (41) prevents the presence of subtour in the solution using the Miller-Tucker-Zemlin (MTZ) formulation (Paragraph 3), preventing cycles in the routes of vehicles.

7.5 Validation

The Gurobi model demonstrated superior speed compared to the MiniZinc model. Our best model certified optimality for instances 12, 16, and 19, and found solutions for instances 11 and 13. We also investigated the impact of removing the redundant constraints from the original model. While this constraints had minimal effect on the first 10 instances, they significantly impacted larger instances, making it challenging to certify the best solution or even find any solution in some cases.

ID	1	2	3	4	5	6	7	8	9	10	11	12	13	16	19
Gurobi obj	**14**	**226**	**12**	**220**	**206**	**322**	**167**	**186**	**436**	**244**	663	**346**	434	**286**	**334**
time	0	0	0	0	0	0	0	0	0	0	300	58	300	0	31
Gurobi no red obj	**14**	**226**	**12**	**220**	**206**	**322**	**167**	**186**	**436**	**244**	-	397	496	**286**	**334**
time	0	0	0	0	0	0	0	0	0	0	-	300	300	1	39
Minizinc obj	**14**	**226**	**12**	220	**206**	**322**	**-**	186	**-**	**-**	-	**-**	-	**-**	**-**
time	0	25	0	300	0	7	-	300	-	-	-	-	-	-	-

Table 5: Results obtained with the different MIP models

8 Conclusions

In our research, we examined a range of computational strategies—CP, SAT, SMT, and MIP models—to tackle the given problem. The SAT and SMT models performed commendably on the initial ten instances, demonstrating their potential for smaller-scale scenarios. However, the MIP model implemented with Gurobi and the final CP model in Minizinc proved pivotal in scaling up the problem-solving capacity, delivering optimal solutions for larger and more complex instances. This advancement underscores the critical role of leveraging advanced solver capabilities and fine-tuning optimization methodologies to overcome the intricacies of routing problems, showcasing how strategic model selection and configuration can dramatically enhance both efficiency and solution precision. We also explored how different searching strategies can significantly alter the behavior of the same model, making it more suitable for either smaller or larger instances. This exploration highlights the importance of adapting search strategies to the specific characteristics of the problem at hand.

References

[1] N. C, Oct. 2023. [Online]. Available: https://www.savemyexams.com/a-level/further-maths_decision-maths-1/edexcel/17/revision-notes/algorithms-on-graphs/the-travelling-salesman-problem/upper-and-lower-bounds-for-the-travelling-salesman-problem/.

[2] Nov. 2020. [Online]. Available: https://how-to.aimms.com/Articles/332/332-Miller-Tucker-Zemlin-formulation.html.

[3] M. Rousseau Louis-Martin Gendreau, "Using constraint-based operators to solve the vehicle routing problem with time windows," *Journal of Heuristics*, 2002.

[4] A. Zha, R. Gao, Q. Chang, M. Koshimura, and I. Noda, "Cnf encodings for the min-max multiple traveling salesmen problem," in *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, IEEE, 2020.

8.1 Appendix

8.1.1 Hardware Specifications

- CPU: AMD Ryzen 5 7600x
- RAM: 32gb DDR5-6000mhz

8.1.2 Algorithms pseudo code

Algorithm 1 Upper Bound Calculation using a Greedy Heuristic

```
1: Input: Distance matrix $D$
2: $num\_nodes \leftarrow$ number of nodes in $D$
3: $visited \leftarrow$ array of $False$ of length $num\_nodes$
4: $total\_cost \leftarrow 0$
5: $current\_node \leftarrow 0$        ▷ Start at an arbitrary node, e.g., node 0
6: $visited[current\_node] \leftarrow True$
7: for $i \leftarrow 1$ to $num\_nodes - 1$ do
8:     $min\_distance \leftarrow \infty$
9:     $next\_node \leftarrow None$
10:    for $neighbor \leftarrow 0$ to $num\_nodes - 1$ do
11:        if $\neg visited[neighbor]$ and $D[current\_node][neighbor] < min\_distance$ then
12:            $min\_distance \leftarrow D[current\_node][neighbor]$
13:            $next\_node \leftarrow neighbor$
14:        end if
15:    end for
16:    if $next\_node \neq None$ then
17:        $total\_cost \leftarrow total\_cost + min\_distance$
18:        $visited[next\_node] \leftarrow True$
19:        $current\_node \leftarrow next\_node$
20:    end if
21: end for
22: $total\_cost \leftarrow total\_cost + D[current\_node][0]$ ▷ Return to the starting node
23: $upper\_bound \leftarrow total\_cost$
24: Output: $upper\_bound$
```