Multiple Couriers Planning Problem

Francesco Cavaleri, francesco.cavaleri@studio.unibo.it
Giacomo Piergentili, giacomo.piergentili2@studio.unibo.it

March 2025

Both group members contributed to all the solutions. To improve productivity, tasks were divided: Piergentili Giacomo primarily developed the MIP model, Cavaleri Francesco focused on the CP model, and the SMT approach was developed collaboratively. The project took approximately six months to complete.

1 Introduction

All models, except the MiniZinc models which requires .mzn files, were implemented in Python and executed in a Docker environment (Ubuntu 22.04 image). Solver independence was achieved for the MIP and CP approaches but not for SMT. During the development the main difficulty was handling large instances due to hardware limitations (mobile Core i5, 16 GB RAM). Some instances exceeded available memory, causing solver crashes. Eventually we added 16 GB SWAP that solved the problem at the expenses of performances. A common preprocessing step was applied to all methods and it consist in parsed the raw instance and transformed it into a suitable format per model. Additionally, for each instance, a lower bound on the objective was computed using the following formula (where D is the distance matrix):

$$\text{lowerbound} = \max_i(\mathrm{D}_{i,N} + \mathrm{D}_{N,i}) \tag{1}$$

2 CP Model

We wanted to exploit bounded_dpath global constraint which, as explained in the documentation, takes in input a fixed graph as arguments represented using two arrays, from and to, representing the graph edges (from[i],to[i]). That requires a preprocessing step of the raw instances, managed by the python class CP_file_instance. The preprocessing add a new node N+1, in that way N represent the start of the trip and N+1 the end (this is important for constructing the exact order in which a courier visit the nodes). For creating the from[i], to[i] arrays, it has bees used the Graph object returned by the python class networkx.

Input Parameter	**Description**
`int: M`	Number of couriers
`int: N`	Number of nodes

Input Parameter	Description
`int: E`	Number of edges in the graph
`int: LOWER`	Lower bound for max_distance
`array [1..M] of int: L`	Maximum load for each courier
`array [1..N-2] of int: S`	Item size or destination node
`array [1..E] of int: FROM`	Source node of each edge
`array [1..E] of int: TO`	Target node of each edge
`array [1..E] of int: W`	Weight of the edge from `FROM[i]` to `TO[i]`

2.1 Decision variables

The variable `courier_distances[i]` (domain: `var int`) represents the total distance traveled by courier i. The variable `node_subset[i, j]` (domain: `var bool`) indicates whether courier i is responsible for visiting node j. Similarly, `edge_subset[i, e]` (domain: `var bool`) is true if courier i uses edge e in their route.The variable `current_load[i]` (domain: `var int`) computes the total weight of items assigned to courier i, based on the sizes of the items and the `node_subset` configuration. To reconstruct the sequence of visits in each courier's path, the matrix `path[i, j]` (domain: `0..N`) is used. A value of `path[i, j] = k` means that node j is the k-th stop in courier i's delivery sequence, while 0 indicates that the node is not visited by that courier.The variable `evaluated_courier_distances[i]` (domain: `var int`) captures the actual path cost based on the ordering in `path`. The variable `max_index` (domain: $1..M$) identifies the index of which courier travels the longest route, and `max_distance` stores the maximum distance.

2.2 Objective function

The objective of this model is to minimize the variable `max_distance` that contains the maximum distance traveled by any courier.

2.3 Constraints

Each courier must travel along a path from a common starting node (`start = N-1`) to a common endpoint (`endpoint = N`) using only the nodes and edges assigned to them. The `bounded_dpath` global constraint ensures that a valid directed path exists for each courier, within the specified subset of nodes and edges, while also computing the path's distance.

```
constraint forall(i in 1..M)(
  bounded_dpath( N, E, FROM, TO, W, start, endpoint,
    row(node_subset, i), row(edge_subset, i), courier_distances[i])
);
```

Each item must be visited by exactly one courier:

$$\sum_{i=1}^{M} \text{node\_subset}_{i,j} = 1 \quad \forall j \in \{1, \ldots, N-2\} \tag{2}$$

Each courier has a maximum load capacity (`L[i]`), which must not be exceeded. The load is calculated as the sum of the item sizes assigned to that courier:

$$\text{current\_load}_i = \sum_{j=1}^{N-2} \text{node\_subset}_{i,j} \cdot S_j \quad \forall i \in \{1, \ldots, M\} \tag{3}$$

$$\text{current\_load}_i \leq L_i \quad \forall i \in \{1, \ldots, M\} \tag{4}$$

A lower bound is imposed on `max_distance` to avoid exploring trivially suboptimal solutions:

$$\text{max\_distance} \geq \text{LOWER} \tag{5}$$

Each courier's route is ordered using the `path` array. The link (channelling constraints) between this model and the previous one are represented by equations (eq. 7, 8, 9, 10). In this model the start node is assigned position 1 (eq. 6), the endpoint corresponds to the number of visited nodes (eq. 7), and set the domain of other variables accordingly (eq. 7, 8, 9) taking in consideration the values of `node_subset`:

$$\text{path}_{i,N-1} = 1 \quad \forall i \in \{1, \ldots, M\} \tag{6}$$

$$\text{path}_{i,N} = \sum_{j=1}^{N-2} \text{node\_subset}_{i,j} \quad \forall i \in \{1, \ldots, M\} \tag{7}$$

$$1 < \text{path}_{i,j} < \text{path}_{i,N} \quad \forall i, j \in \{(i,j) | \text{node\_subset}_{i,j} = 1\} \tag{8}$$

$$\text{path}_{i,j} = 0 \quad \forall i, j \in \{(i,j) | \text{node\_subset}_{i,j} = 0\} \tag{9}$$

$$\text{path}_{i,j} \neq \text{path}_{i,k} \quad \forall i, j, k \in \{(i,j,k) | j \neq k, \text{node\_subset}_{i,j} = 1, \text{node\_subset}_{i,k} = 1\} \tag{10}$$

This part aim to reconstruct the order of the trip for each courier. it works by creating a circuit where it search for the possible next items to be delivered. Consider, for each courier i, the following sets:

$$\xi_i = \{j \in \mathbb{N}^+ | j \leq \text{path}_{i,N} - 1\} \tag{11}$$

$$\Omega^-_{i,j} = \{k \in \{1, \ldots, E\} | \text{path}_{i,FROM_k} = j\} \tag{12}$$

$$\Omega^+_{i,j} = \{k \in \{1, \ldots, E\} | \text{path}_{i,TO_k} = j + 1\} \tag{13}$$

Eq. 11 represent, for each courier i, the sequence of visited nodes. Eq. 12, 13 represent, respectively, for each courier i the index of the edges that deliver the j-th item and the next item (j+1).

$$\Phi_i = \{k \in \{1...E\} | \forall j \in \xi_i, \Omega^-_{i,j} \cap \Omega^+_{i,j}\} \quad \forall i \in \{1, \ldots, M\} \tag{14}$$

Eq. 14 represent the index of the edges used by the courier i. This ordered structure reconstruct each courier's route accordingly to this model:

$$\text{evaluated\_courier\_distances}_i = \sum_{k \in \Phi_i} \text{W}_k \quad \forall i \in \{1, \ldots, M\} \tag{15}$$

At the end, we add other channelling constraints that allows, if a better solution if found in this other model, to apply it

$$\text{evaluated\_courier\_distances}_i \leq \textit{max\_distance} \quad \forall i \in \{1, \ldots, M\} \tag{16}$$

And given the index max_index of the longest trip:

$$\begin{aligned} &\text{evaluated\_courier\_distances}_{\text{max\_index}} < \text{max\_distance} \\ &\quad \Rightarrow \text{courier\_distances}_{\text{max\_index}} = \text{evaluated\_courier\_distances}_{\text{max\_index}} \end{aligned} \tag{17}$$

2.3.1 Symmetry breaking

Symmetry in the problem arises due to the interchangeability of couriers with similar or equal capacities. To address this, the model includes a symmetry-breaking constraint that lexicographically orders the node assignments of any two couriers whose maximum loads are within compatible bounds. Whenever two couriers have comparable capacity and workload, it enforces a consistent ordering of node assignments.

$$\begin{aligned} \max(\text{current\_load}_i, \text{current\_load}_j) \leq \min(L_i, L_j) \\ \Rightarrow \text{lex\_lesseq}(\text{row}(\text{node\_subset}, i), \text{row}(\text{node\_subset}, j)) \\ \forall i, j \in \{1, \ldots, M\},\ i < j \end{aligned} \tag{18}$$

This eliminates symmetric solutions that are equivalent under courier index permutations, thus improving solver performance by avoiding redundant branches in the search tree.

2.4 Validation

Both the standard and symmetry-breaking models are solver-independent, allowing us to conduct experiments using both the Gecode and Chuffed solvers. The search process follows a two-phase strategy: an initial boolean search over the node_subset variables using the dom_w_deg heuristic combined with indomain_min, followed by an integer search over the path variables using input_order. These two phases are composed using the seq_search construct. Additionally, a Luby restart policy is applied to enhance exploration and avoid early stagnation in the search.

2.4.1 Experimental results

The table below summarizes the results obtained across all CP model configurations. For instances not listed, no solution was found by any of the tested models within the time limit. We observe that, in many cases, when a solution is found, it is also optimal. This suggests that the model benefits from strong constraint propagation, effectively guiding the search toward optimal solutions (text in bold). The positive impact of the symmetry-breaking constraint is particularly evident in instance 07, which was only solved to optimality when symmetry breaking was applied. Additionally, under the current model and set of instances, the Gecode solver demonstrated a slight advantage over Chuffed, successfully finding solutions for instances 07 and 13 where Chuffed did not. Execution times are shown in Figure 1. All instances before the 10th, except for instance 7, were solved relatively quickly across all solvers. Instance 7, however, reached an optimal solution within a reasonable timeframe only when using Gecode with the symmetry-breaking constraint. For instance 13, we observe a clear cutoff at the 300-second time limit, with the solver returning a suboptimal solution before timing out.

INST	GECODE	SB-GECODE	CHUFFED	SB-CHUFFED
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	176	**167**	N/A	N/A
8	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**
13	954	1210	N/A	N/A

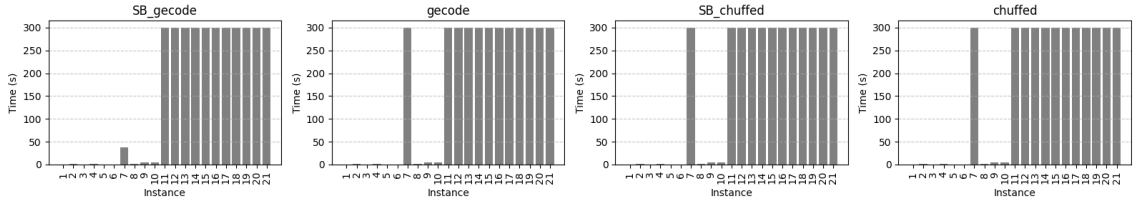

Figure 1: CP solvers execution times

3 SMT Model

SMT model leverages the power of SAT solvers combined with theory solvers to handle both boolean logic and integer arithmetic constraints simultaneously. This approach allows for a more direct encoding of the problem logic, where routing decisions and delivery ordering can be expressed through uninterpreted functions and their relationships.

Input Parameter	**Description**
`int: M`	Number of couriers
`int: N`	Number of nodes
`int: LOWER`	Lower bound for max_distance
`array [0..M] of int: L`	Maximum load for each courier
`array [0..N] of int: S`	Item size or destination node
`array [0..N][0..N] of int: D`	distance matrix

3.1 Decision Variables

The depot is represented as node n, while items are numbered from 0 to $n-1$. To enable a more expressive and less constrained model, we adopt uninterpreted functions, which avoid issues such as the ArithRef indexing problem by allowing symbolic variables to handle indexing directly.

The assignment of items to couriers and the sequencing of deliveries are encoded through the function **item_order**(i, j) with signature IntSort() $\times$ IntSort() $\rightarrow$ IntSort(). It returns an integer k, interpreted as the delivery position of item j in courier i's route. A value $k \geq 1$ indicates the order in which the item is delivered. Routing behavior is captured by the Boolean function **travels**(i, j, k) with signature IntSort() $\times$ IntSort() $\times$ IntSort() $\rightarrow$ BoolSort(). It evaluates to true if and only if courier i travels directly from node j to node k, where nodes include both items and the depot (the latter being indexed as n). To finalize the courier route, each courier is also associated with an uninterpreted function **courier_last_item** indicating the last item they deliver before returning to the depot. The model maintains, for each courier, in the variable **courier_load** representing the total load they carry. The domains of these functions are implicitly constrained by the problem structure: courier indices range from 0 to $m-1$, item indices from 0 to $n-1$, and location indices from 0 to n (including the depot). The delivery order values are constrained to be non-negative integers, with zero indicating non-assignment.

3.2 Objective Function

The objective of this model is to minimize the variable `max_dist` that contains the maximum distance traveled by any courier.

3.3 Constraints

We categorize the constraints into main problem constraints, implied constraints, and symmetry breaking constraints.

3.3.1 Main Problem Constraints

The core constraints that are strictly necessary for modeling the MCP problem are:

Objective Bound Constraints: The maximum distance variable bounds each courier's total distance.

$$\text{max\_dist} = \max_i \left\{ \sum_{j=0}^{n} \sum_{k=0}^{n} \text{If}(\text{travels}(i, j, k), d[j][k], 0) \right\} \tag{19}$$

Courier load and Capacity Constraint: represent the total load carried by each courier. For courier i, the load is computed as:

$$\text{courier\_load}[i] = \sum_{j=0}^{n-1} \text{If}(\text{item\_order}(i, j) \geq 1, s[j], 0)$$

Each courier cannot exceed their maximum load capacity.

$$\text{courier\_load}[i] \leq l[i] \quad \forall i \in \{0, \ldots, m-1\} \tag{20}$$

Item Assignment Constraint: Each item must be assigned to exactly one courier.

$$\sum_{i=0}^{m-1} \text{If}(\text{item\_order}(i, j) \geq 1, 1, 0) = 1 \quad \forall j \in \{0, \ldots, n-1\} \tag{21}$$

Depot Start/End Constraints: The number of times each courier leave/arrive at the depot is once.

$$\sum_{j=0}^{n-1} \text{If}(\text{travels}(i, n, j), 1, 0) = 1 \quad \forall i \in \{0, \ldots, m-1\} \tag{22}$$

$$\sum_{j=0}^{n-1} \text{If}(\text{travels}(i, j, n), 1, 0) = 1 \quad \forall i \in \{0, \ldots, m-1\} \tag{23}$$

Flow Conservation Constraint: For each assigned item, the courier must visit it exactly once.

$$\sum_{k=0}^{n} \text{If}(\text{travels}(i, k, j), 1, 0) = \text{If}(\text{item\_order}(i, j) \geq 1, 1, 0) \quad \forall i, j \tag{24}$$

Ordering Consistency Constraints: The delivery order must be consistent with the travel sequence.

$$\forall i \in \{0, \ldots, m-1\},\ \forall k \in \{0, \ldots, n-1\}, \quad \text{travels}(i, n, k) \Rightarrow \text{item\_order}(i, k) = 1 \tag{25}$$

$$\forall i \in \{0, \ldots, m-1\}, \quad \text{travels}(i, \text{courier\_last\_item}(i), n) = \text{True} \tag{26}$$

$$\forall i \in \{0, \ldots, m-1\},\ \forall j, k \in \{0, \ldots, n\},\ j \neq k, \quad \text{travels}(i, j, k) \Rightarrow \text{item\_order}(i, k) = \text{item\_order}(i, j) + 1 \tag{27}$$

where courier_last_item is the index of the last delivered item by courier i.

Lower Bound Constraint: A precomputed lower bound strengthens the formulation.

$$\text{max\_dist} \geq \text{lower} \tag{28}$$

3.3.2 Implied Constraints

Several additional constraints strengthen the formulation and improve solving performance:

Domain Constraints: Explicit bounds on the delivery order variables.

$$0 \leq \text{item\_order}(i, j) \leq n \quad \forall i \in \{0, \ldots, m-1\}, j \in \{0, \ldots, n-1\} \tag{29}$$

Assignment-Travel Consistency: Non-assigned items have no associated travel.

$$\text{item\_order}(i, j) = 0 \Leftrightarrow \left(\sum_{k=0}^{n} \text{travels}(i, j, k) = 0 \wedge \sum_{k=0}^{n} \text{travels}(i, k, j) = 0 \right) \tag{30}$$

No Self-Loops: Prevent degenerate travel decisions. Combined with formula 24 it ensures the courier can't ever comeback to the same item.

$$\neg\text{travels}(i, j, j) \quad \forall i, j \tag{31}$$

Distinct Order Values: Items delivered by the same courier have different order values.

$$(j_1 \neq j_2 \wedge \text{item\_order}(i, j_1) \geq 1 \wedge \text{item\_order}(i, j_2) \geq 1) \Rightarrow \text{item\_order}(i, j_1) \neq \text{item\_order}(i, j_2) \tag{32}$$

3.3.3 Symmetry Breaking Constraints

The model exhibits symmetry when multiple couriers have identical or compatible capacities, as their assignments can be interchanged without affecting solution validity. We address this through lexicographic ordering: for couriers whose workloads can be interchanged, impose lexicographic order on their item assignments.

$$(\max(\text{load}_{c_1}, \text{load}_{c_2}) \leq \min(l[c_1], l[c_2])) \Rightarrow \text{assignments}_{c_1} \leq_{\text{lex}} \text{assignments}_{c_2} \tag{33}$$

where $c_1 < c_2$ and the lexicographic ordering is implemented as:

$$\bigvee_{i=0}^{n-1} \left(\bigwedge_{k=0}^{i-1} \text{item\_order}(c_1, k) = \text{item\_order}(c_2, k) \wedge \text{item\_order}(c_1, i) \leq \text{item\_order}(c_2, i) \right) \tag{34}$$

The condition ensures that symmetry breaking is only applied when couriers can actually interchange their workloads without violating capacity constraints.

3.4 Validation

3.4.1 Experimental results

The table below summarizes the results obtained across all SMT models. For instances not listed, no solution was found by any of the tested models within the time limit. All solutions are optimal.

INST	Z3	SB-Z3
1	**14**	**14**
2	**226**	**226**
3	**12**	**12**
4	**220**	**220**
5	**206**	**206**
6	**322**	**322**
8	**186**	**186**
9	**436**	**436**
10	**244**	**244**

When a solution is found, it is also optimal. This is a direct consequence of the timeout mechanism we implemented (the z3 timeout did not work): solvers were executed in a separate thread, while the main thread enforced the time limit by interrupting execution upon expiration. Although this ensured compliance with the timeout constraint, it prevented the retrieval of suboptimal solutions.

Both the standard and symmetry-breaking models produced comparable results, with the standard model often performing slightly better overall.

Execution times are illustrated in Figure 1. All instances up to the 10th, except for instance 7, were solved across all configurations.

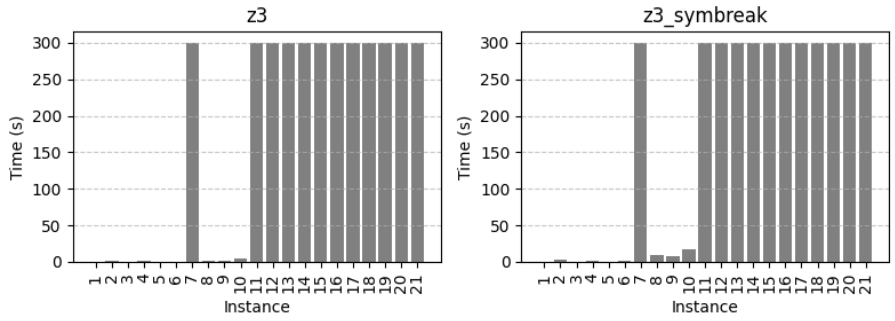

Figure 2: SMT solvers execution times

4 MIP Model

We implemented a Mixed Integer Programming model with binary decision variables to handle routing decisions.

Input Parameter	Description
`int: m`	Number of couriers
`int: n`	Number of items
`dict: capacities`	Dictionary mapping each courier i to its maximum load capacity
`dict: sizes`	Dictionary mapping each item j to its size requirement
`dict: distances`	Dictionary containing the distance matrix between all locations (items and depot)
`int: depot_location`	Depot location (equal to n)

The depot is represented as node n, while items are numbered from 0 to $n-1$. Each courier must start and end their route at the depot.

4.1 Decision Variables

The MIP model uses the following decision variables:

- $x_{i,j}$: Binary variable with domain $\{0,1\}$ for $i \in \{0,\ldots,m-1\}$ and $j \in \{0,\ldots,n-1\}$. The variable $x_{i,j} = 1$ if and only if courier i is assigned to deliver item j, and $x_{i,j} = 0$ otherwise.
- $y_{i,j,k}$: Binary variable with domain $\{0,1\}$ for $i \in \{0,\ldots,m-1\}$ and $j,k \in \{0,\ldots,n\}$. The variable $y_{i,j,k} = 1$ if and only if courier i travels directly from location j to location k, where locations 0 to $n-1$ represent items and location n represents the depot.
- $u_{i,j}$: Integer variable with domain $[1,n]$ for $i \in \{0,\ldots,m-1\}$ and $j \in \{0,\ldots,n-1\}$. The variable $u_{i,j} = p$ indicates the position (order) in which courier i deliver item j during their route, where $p = 1$ means it is the first item delivered and $p = n$ means it is the last possible position.

- Z: Continuous variable with domain $[0, +\infty)$. This variable represents the maximum distance traveled by any courier, which is our objective function to minimize.

The binary flow variables $y_{i,j,k}$ model the routing decisions, while the assignment variables $x_{i,j}$ determine which courier delivers each item. The ordering variables $u_{i,j}$ are used in Miller-Tucker-Zemlin (MTZ) constraints to eliminate subtours, ensuring each courier follows a valid path from the depot through their assigned items and back.

4.2 Objective Function

The objective function implements a minimax approach:

$$\min Z \tag{35}$$

where Z is constrained to be at least as large as each courier's total travel distance:

$$Z \geq \sum_{j=0}^{n} \sum_{k=0}^{n} \text{distances}[j,k] \cdot y_{i,j,k} \quad \forall i \in \{0, \ldots, m-1\} \tag{36}$$

This formulation minimizes the maximum distance traveled by any courier, which helps balance the workload across all couriers.

4.3 Constraints

To ensure the basic requirements of the MCP problem, the model imposes that each item is assigned to exactly one courier. This is achieved through the constraint $\sum_{i=0}^{m-1} x_{i,j} = 1$ for all items $j \in \{0, \ldots, n-1\}$, where $x_{i,j}$ represents the binary assignment decision.

Moreover, our MIP model enforces the fact that each courier cannot exceed its maximum carrying capacity. So, the model imposes that $\sum_{j=0}^{n-1} s_j \cdot x_{i,j} \leq L_i$ for all couriers $i \in \{0, \ldots, m-1\}$, which means that a given courier can only be assigned items whose total size does not exceed its maximum capacity.

The model connects assignment decisions with routing through flow variables $y_{i,j,k}$ that represent whether courier i travels directly from location j to location k. To maintain consistency between assignments and routing, the model ensures that for each assigned item, exactly one incoming and one outgoing flow exists:

$$\sum_{k=0, k\neq j}^{n} y_{i,k,j} = x_{i,j} \quad \text{and} \quad \sum_{k=0, k\neq j}^{n} y_{i,j,k} = x_{i,j}$$

for all couriers i and items j.

Additionally, the model restricts each courier to leave the depot at most once through $\sum_{k=0}^{n-1} y_{i,n,k} \leq 1$ for all couriers i. To ensure round trips, it also enforces that the number of times a courier leaves the depot equals the number of times it returns: $\sum_{k=0}^{n-1} y_{i,n,k} = \sum_{k=0}^{n-1} y_{i,k,n}$ for all couriers i.

The model also ensures that couriers only leave the depot if they have assigned items:

$$\sum_{k=0}^{n-1} y_{i,n,k} \leq \sum_{j=0}^{n-1} x_{i,j} \quad \forall i \in \{0, \ldots, m-1\}$$

This constraint prevents inactive couriers (those with no assigned items) from creating empty routes.

A challenging issue when dealing with vehicle routing problems is sub-tour elimination. In order to solve that, the model imposes Miller-Tucker-Zemlin constraints:

$$u_{i,j} - u_{i,k} + n \cdot y_{i,j,k} \leq n - 1$$

for all couriers i and distinct items j, k. These ordering variables $u_{i,j}$ prevent the formation of disconnected cycles among the assigned items.

Furthermore, the model prohibits self-loops through $y_{i,j,j} = 0$ for all couriers i and locations j, as these would not represent meaningful travel decisions.

To implement the minimax objective, the model bounds the maximum distance variable Z from below by each courier's total travel distance: $\sum_{j=0}^{n} \sum_{k=0,k\neq j}^{n} d_{j,k} \cdot y_{i,j,k} \leq Z$ for all couriers i.

As described in previous sections, we bounded the search space to improve computational performance using a precomputed lower bound. Therefore, the model includes the constraint $Z \geq \max_{j\in\{0,\ldots,n-1\}}\{d_{n,j} + d_{j,n}\}$ that imposes this lower bound.

Finally, to address symmetry when multiple couriers have identical capacities, the model includes constraints that lexicographically order their loads: $\sum_{j=0}^{n-1} s_j \cdot x_{i,j} \geq \sum_{j=0}^{n-1} s_j \cdot x_{i+1,j}$ for consecutive couriers with the same capacity. This particular approach reduces the solution space by eliminating equivalent solutions that differ only in courier index permutations.

4.4 Validation

We tested the MIP model with both Gurobi and HiGHS solvers, with and without symmetry breaking constraints.

4.4.1 Experimental Results

The table below summarizes the results obtained across all MIP model configurations. For instances not listed, no solution was found within the time limit. Optimal solutions are highlighted in bold. The MIP model performed well, finding optimal solutions for all instances from 1 to 10. Gurobi consistently outperformed HiGHS, especially on larger instances (12, 16, and 19), where only Gurobi found solutions within the time limit. The symmetry breaking constraints showed mixed results - sometimes helping performance but occasionally preventing solutions from being found within the timeout.

INST	Gurobi	SB-Gurobi	HiGHS	SB-HiGHS
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**
12	**346**	N/A	N/A	N/A
16	**286**	**286**	N/A	N/A
19	**334**	N/A	N/A	N/A

Execution times are shown in Figure 3. Most small instances (1-6, 8-10) solved very quickly (0-4 seconds). Instance 7 was more challenging, with HiGHS requiring 62s (35s with symmetry breaking) compared to Gurobi's 1-2s. The larger instances took 19-212 seconds with Gurobi.

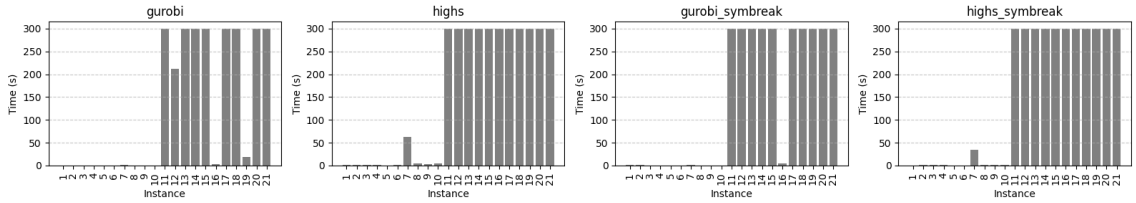

Figure 3: MIP solvers execution times

5 Conclusion

Our experiments revealed pattern: instances 1-10 (excluding instance 7) were relatively easy to solve regardless of the approach used. Instance 7, however, turned out to be a significant challenge for both CP and SMT models. Without symmetry breaking constraints, these approaches either failed to find optimal solutions or took excessive time. The larger instances (12, 16, 19) presented a different story entirely - only the MIP approach managed to solve them within our time limits. The difference between Gurobi and HiGHS was particularly striking on harder instances - HiGHS would time out after 300 seconds while Gurobi found solutions in under a minute. This demonstrates how solver choice can be decisive for NP-hard problems.

Symmetry breaking constraints produced mixed results. While they were essential for solving instance 7 optimally in the CP model, they often hurt performance in other cases. For the SMT model, adding these constraints seemed to create more work for the solver without providing proportional benefits.

References

[1] L. de Moura and N. Bjørner. Z3: An efficient smt solver, 2008.

[2] LLC. Gurobi Optimization. Gurobi optimizer reference manual. `https://www.gurobi.com`, 2023.

[3] Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

[4] Stuckey P. J. Becket R. Brand S. Duck G. J. Nethercote, N. and G. Tack. Minizinc: Towards a standard cp modelling language, 2007.