CDMO project

Armando Renzullo, Cosimo Emanuele Russo,
Tommaso Perniola

September 2024

1 Introduction

This report addresses the Multiple Courier Planning Problem, a key challenge in logistics and transportation management. In this problem, the objective is to efficiently assign and route multiple couriers to deliver items across various locations. The goal is to optimize routes to minimize the total distance traveled, while adhering to constraints like courier capacities and delivery time windows. The problem combines elements of vehicle routing, scheduling, and load balancing to ensure effective and economical distribution strategies. This report details our group's efforts to address this problem by proposing and exploring a potential solution through advanced optimization techniques. We outline the methodology, design choices, and results of our approach, aiming to offer a feasible and effective strategy for solving the multiple courier planning challenge.

1.1 Model Foundations

This subsection outlines the foundational strategies and common methodologies employed across the models presented in this report.

1.1.1 Objective function

The goal is to assign items to couriers such that the maximum distance traveled by any courier is minimized. In a more formal notation:

$$\min_{c \in \text{Couriers}} \max(D_c)$$

where D_c represents the total distance traveled by courier c.

1.2 Preprocessing steps

To ensure the efficiency and feasibility of our models, we employed several simplification and preprocessing steps. This included checking mathematical properties such as the triangle inequality and matrix symmetry, as well as establishing lower and upper bounds for key variables. The following sections detail these steps.

1.2.1 Package Distribution

To effectively distribute packages among couriers, we employed a **heuristic** that ensures each courier carries a number of packages within a specified range. Specifically, we set a constraint so that each courier picks at least 1 and at most $\lfloor \frac{n}{m} \rfloor + 3$ packages, where n is the total number of packages and m is the number of couriers. This approach balances the package distribution while allowing for some flexibility.

1.2.2 Bounds

This section outlines the lower and upper bounds for the objective function, which are essential in guiding the search for an optimal solution.

The upper bound (ub) and lower bound (lb) for the distances are calculated based on the minimum ($minD$) and maximum ($maxD$) distances in the distance matrix D, and the maximum number of nodes per courier, as denoted in the previous subsection 1.2.1. They are determined as:

$$lb = (\left\lfloor \frac{n}{m} \right\rfloor + 3) \times minD$$

$$ub = (\left\lfloor \frac{n}{m} \right\rfloor + 3) \times maxD$$

Where $minD$ and $maxD$ are computed iterating through all pairs of indices (i, j) in the distance matrix D, excluding diagonal elements where $i = j$ and updating $maxD$ with the maximum value found and $minD$ with the minimum value found among these distances.

1.2.3 Triangle Inequality and Symmetry

We safely assume that the triangle inequality always holds for all distances. This property ensures that, for any three points i, j, and k, the direct distance from i to k is not greater than the sum of the distances from i to j and from j to k:

$$D[i, k] \leq D[i, j] + D[j, k] \tag{1}$$

Additionally, our scripts confirmed that distance matrices were always non-symmetric, meaning:

$$D[i, j] \neq D[j, i] \tag{2}$$

1.3 Project execution overview

Completion time and workload division The project spanned three months with intermittent work sessions. Experimental tasks involving different solvers and strategies were collectively discussed before assigning specific responsibilities to each team member. Any challenges or issues encountered were reviewed and addressed by the entire group to ensure comprehensive input and oversight. The development of each model was a collaborative effort, though each member contributed significantly to specific models:

- CP: Contributions were primarily from Renzullo.
- SAT: Each team member contributed equally.
- SMT: Perniola focused on this part.
- MIP: Russo was primarily responsible for developing this model.

Main difficulties The primary challenges revolved around the SAT workflow, particularly in developing an effective and efficient model capable of solving a range of instances. Additionally, the MiniZinc models were critical, as they laid the groundwork for subsequent models. The team's collaboration was essential in overcoming these difficulties and advancing the project.

2 CP Model

Constraint programming is an optimization paradigm that systematically explores all potential combinations of variable values through a backtracking tree search. Using the MiniZinc language, we developed a constraint programming model which was subsequently tested with two different solvers: Gecode and Chuffed. We experimented with various problem encodings, particularly comparing those that utilize symmetry-breaking constraints with those that do not.

2.1 Decision Variables

Our model relies on the following decision variables:

- $x_{i,j}$: A decision variable representing the path of courier i. Specifically, it indicates the node that courier i will visit immediately after node j. The domain of $x_{i,j}$ is $\{1, 2, \ldots, n+1\}$.
  - If $x_{i,j} = j$, then courier i does not visit node j.
  - If $x_{i,n+1} \neq n+1$, then courier i must return to the depot after visiting node $n+1$.
- $max\_distance\_per\_courier_i$: A variable representing the maximum distance traveled by courier i. This variable is defined over the range [lb ub], where total is the sum of all distances in the distance matrix D.
- $max\_distance$: A decision variable representing the maximum distance among all couriers, which serves as the objective function.

2.2 Objective Function

The objective function is constrained by the equality constraint defining $max\_distance$ as the maximum value among all $max\_distance\_per\_courier[i]$ values. This is detailed in Section 1.1.1.

2.3 Constraints

The constraint programming model includes the following constraints:

1. **Subcircuiting:**

$$\forall i \in \{1, \ldots, m\} : \text{subcircuit}([x_{i,j} \mid j \in \{1, \ldots, n+1\}]) \tag{3}$$

   This constraint ensures that each courier forms a valid subcircuit. (This ensure that the couriers will take every package). The subcircuit constraint is a well-known method in optimization problems, particularly for solving multiple tours problems, as detailed by Vismara and Briot [1].

2. **Unique delivery:**

$$\forall j \in \{1, \ldots, n\} : \text{count}_{i \in \{1,\ldots,m\}}(x_{i,j} \neq j) = 1 \tag{4}$$

   This constraint ensures that each package (node) is delivered exactly once.

3. **Return to the depot and starting:**

$$\forall i \in \{1, \ldots, m\} : x_{i,n+1} \neq n+1 \tag{5}$$

   Each courier must start from the depot.

$$\forall i \in \{1, \ldots, m\} : \text{count}_{j \in \{1,\ldots,n\}}(x_{i,j} = n+1) = 1 \tag{6}$$

   Each courier must visit the depot exactly once apart from the starting.

4. **Courier load not exceeded:**

$$\forall i \in \{1, \ldots, m\} : \sum_{j_1 \in \{1,\ldots,n\}} (\text{if } x_{i,j_1} = j_1 \text{ then } 0 \text{ else } s_{j_1} \text{ endif}) \leq l_i \tag{7}$$

   This constraint ensures that the total load carried by each courier does not exceed their capacity.

5. **Maximum distance:**

$$\forall i \in \{1, \ldots, m\} : \text{max\_distance\_per\_courier}_i = \sum_{j_1 \in \{1,\ldots,n+1\}} D_{j_1, x_{i,j_1}} \tag{8}$$

   This constraint defines the maximum distance traveled by each courier.

Implied Constraints

1. It ensures that for each row i, the number of elements j where $x[i, j]$ does not equal j is at most $\lceil \frac{n}{m} \rceil + 1$.

$$\forall i \in \{1, \ldots, m\}, \quad \text{count}\left(\{j \in \{1, \ldots, n\} \mid x[i, j] \neq j\}\right) \leq \left\lceil \frac{n}{m} \right\rceil + 1$$

2. This constraint ensures that the elements $x[i, n+1]$ for all $i \in \{1, \ldots, m\}$ are all different, and it uses domain propagation for efficient solving.

$$\text{alldifferent}\left([x[i, n+1] \mid i \in \{1, \ldots, m\}]\right) \quad :: \text{domain\_propagation}$$

Symmetry-breaking Constraints The following constraint enforces a lexicographical order on the paths of couriers with equal load capacity, reducing the symmetry in the solution space:

$$\forall i \in \{1, \ldots, m-1\}, \forall z \in \{i+1, \ldots, m\} \text{ where } l[i] = l[z],$$
$$\text{lex\_lesseq}\left([x[z,k] \mid k \in \{1, \ldots, n+1\}], [x[i,k] \mid k \in \{1, \ldots, n+1\}]\right) \quad (9)$$

This constraint uses domain propagation to efficiently enforce the lexicographical order.

2.4 Validation

The model is implemented in MiniZinc and it has been tested using two different solvers, Gecode and Chuffed. In addition, it has been made a comparison between different search strategies over the integer space.

Experimental Design The most effective search strategy turned out to be the one which implements:

- On x: "domWdeg" as variable selection heuristic with "indomain_random" as value selection heuristic.
- On *ind*: "first fail" as variable selection heuristic with "indomain_random" as value selection heuristic.
- a "restart_luby" as restart strategy, in order to periodically restart the search.
- a "relax and reconstruct" heuristic over y to improve the efficiency of finding solutions by partially relaxing the current assignment and then reconstructing it.

We implemented a restart strategy and a "relax and reconstruct" heuristic in order to explore different regions of the search space, helping to escape from local optima and improve the search diversity.

Experimental Results The following table shows the results for two different solvers, each tested with different search strategies on y.

3 SAT model

In this SAT-based model, we aimed to implement an iterative optimization process that adjusts upper bounds to converge on an optimal solution. However, due to challenges in formulating the required distance constraint, we were unable to perform this iterative refinement. Consequently, the model is designed to halt at the first feasible solution identified by the solver, without further exploration for the optimal value. Note that no pseudo-Boolean constraints of Z3 were used.

Instance	Gecode with SB	Gecode w/out SB	Chuffed with SB
1	**14**	**14**	**14**
2	**226**	**226**	**14**
3	**12**	**12**	**14**
4	**220**	**220**	**14**
5	**206**	**206**	**14**
6	**322**	**322**	**14**
7	**167**	**167**	**14**
8	**186**	**186**	**14**
9	**436**	**436**	**14**
10	**244**	**244**	**14**
11			**14**
12			**14**
13			**14**
14	-	-	**14**
15	-	-	**14**
16	**286**	**286**	**14**
17	-	-	**14**
18	-	-	**14**
19			**14**
20	-	-	**14**
21	-		**14**

Table 1: Results using Gecode and Chuffed with different search strategies.

3.1 Decision Variables

- *pickup_schedule*[c][i][t]: This is a 3D Boolean array that is **True** if courier c picks up item i at time step t; otherwise, it is **False**. This variable is used to model and enforce all the model constraints.

Structure

- **Courier index** c: Refers to the courier, where c ranges from 0 to $m-1$, with m being the total number of couriers.
- **Item index** i: Refers to the item being picked up, where i ranges from 0 to n, with n being the total number of items. The index $i = n$ is reserved for the depot or starting/ending position.
- **Time step index** t: Represents the time step at which an action is taken, where t ranges from 0 to max_times, with max_times being the maximum number of time steps a courier can perform pickups or deliveries.

3.2 Objective Function

Since no distance constraint is present in the model, there is no objective function to minimize.

3.3 Constraints

1. **Load capacity:** For each courier $c \in \{1, \ldots, m\}$, the total weight of the items picked up at any time t should not exceed the courier's capacity $l[c]$:

$$\sum_{i=1}^{n} \sum_{t=1}^{T} \sum_{k=1}^{s[i]} \text{pickup\_schedule}_{c,i,t} \leq l[c]$$

   where $s[i]$ is the size/weight of item i and T is the maximum time step.

2. **Start and End position:** Each courier $c \in \{1, \ldots, m\}$ must start at position $i = n$ at time $t = 0$ and return to position $i = n$ at the final time step T:

$$\text{pickup\_schedule}_{c,n,0} = \text{True}$$
$$\text{pickup\_schedule}_{c,n,T} = \text{True}$$

3. **At least one pickup:** For each courier $c \in \{1, \ldots, m\}$, at least one item $i \in \{1, \ldots, n\}$ must be delivered during the time steps $t \in \{1, \ldots, T-1\}$:

$$\sum_{i=1}^{n} \sum_{t=1}^{T-1} \text{pickup\_schedule}_{c,i,t} \geq 1$$

4. **No revisit constraint:** For each courier $c \in \{1, \ldots, m\}$ and each item $i \in \{1, \ldots, n\}$, the item can only be picked up at most once across all time steps $t \in \{1, \ldots, T-1\}$:

$$\sum_{t=1}^{T-1} \text{pickup\_schedule}_{c,i,t} \leq 1$$

5. **Unique item assignment:** For each item $i \in \{1, \ldots, n\}$, the item must be picked up exactly once by one courier across all time steps $t \in \{1, \ldots, T-1\}$ and all couriers $c \in \{1, \ldots, m\}$:

$$\sum_{c=1}^{m} \sum_{t=1}^{T-1} \text{pickup\_schedule}_{c,i,t} = 1$$

6. **All items are considered:** For each item $i \in \{1, \ldots, n\}$, the item must be picked up at least once by some courier at some time $t \in \{1, \ldots, T-1\}$:

$$\sum_{c=1}^{m} \sum_{t=1}^{T-1} \text{pickup\_schedule}_{c,i,t} \geq 1$$

Symmetry Breaking For any pair of couriers c_1 and c_2 with the same load size $l[c_1] = l[c_2]$, impose a symmetry-breaking constraint:

$$\forall t \in \{1, \ldots, T-1\}, \forall i \in \{1, \ldots, n\}, \quad \text{pickup\_schedule}_{c_1,i,t} \implies \text{pickup\_schedule}_{c_2,i,t}$$

3.4 Validation

Experimental Design After incorporating preprocessing steps to enhance the search process, such as calculating the maximum number of items each courier can pick up (see Subsection 1.2.1), the model was able to provide the first feasible solution it encountered more effectively.

Experimental Results Out of the first 10 instances, the model successfully found solutions for only three of them. Of these, only the solution provided for the 5th instance is optimal. These results are displayed in Table 3.

4 SMT Model

The SMT (Satisfiability Modulo Theories) approach addresses optimization problems using solvers specialized in different theories. For the couriers problem, linear integer theory and array theory were employed, which minimized spatial complexity and facilitated advanced modeling.

Instance	Z3 SAT
1	16
2	-
3	14
4	-
5	**206**
6	-
7	-
8	-
9	-
10	-

Table 2: First feasible results obtained by the SAT model

4.1 Decision Variables

Decision variables represent the choices in solving the problem and impact the objective function and constraints.

- *b_path*: A boolean matrix where *b_path[c][j]* is *True* if courier *c* takes the *j-th* item. This variable uses Array theory and Boolean Logic theory, with 'ArraySort(IntSort(), BoolSort())' indices.
- *path*: A two-dimensional array of integers representing the sequence of items taken by each courier. *path[c][j]* denotes the *j-th* item taken by courier *c*. This variable employs Arrays and Arithmetic theories, with 'ArraySort(IntSort(), IntSort())' indices.
- *load*, *size*, *path_length*, and *total_distance* are 1-dimensional integer arrays using Arrays and Arithmetic theories:
  - *load[c]*: Load capacity of courier *c*.
  - *size[j]*: Size of item *j*.
  - *path_length[c]*: Length of the path for courier *c*.
  - *total_distance[c]*: Total distance traveled by courier *c*.
- *D_func*: A function representing the distance matrix, mapping each pair of items to an integer distance. This function uses EUF (Uninterpreted Functions) theory.
- *max_dist*: A single integer representing the maximum distance traveled by any courier, to be minimized. It uses the 'Int' sort and Arithmetic theory.

4.2 Objective Function

As discussed in the Subsection 1.1.1, the goal is to minimize the maximum distance traveled by any courier. Formally, the objective is:

$$\min_{c \in \text{Couriers}} \max(D_c)$$

where D_c represents the total distance traveled by courier c, computed as:

$$D_c = \sum_{j=1}^{k_c} D_{\text{func}}(p_{c,j-1}, p_{c,j})$$

Here:

- k_c is the number of nodes visited by courier c.
- $p_{c,j}$ is the j-th location visited by courier c.
- $D_{\text{func}}(i, j)$ is the distance function between locations i and j.

4.3 Constraints

Constraints ensure the solution is valid and respects the problem's requirements. First, we have to list the constraints defined just for convenience, to address the 0-indexing in Python:

- **Distance matrix mapping:** $\forall i, j \in \{0, 1, \ldots, n\} \quad D_{\text{func}}(i, j) = D[i][j]$
- **Items' sizes:** $\forall i \in \{0, 1, \ldots, n\} \quad \text{size}[i+1] = s[i]$
- **Couriers' load capacities:** $\forall c \in \{0, 1, \ldots, m\} \quad \text{load}[c+1] = l[c]$

Now we give a mathematical formulation of those constraints that actually model the problem. Note that the MAX_ITEMS constant is the variable defined in 1.2.1.

1. **Path range constraints:** The path values for each courier must range between 0 and $n+1$:

$$0 \leq \text{path}[c][j] \leq n+1 \quad \forall c \in \text{Couriers}, \forall j \in \{1, \ldots, \text{MAX\_ITEMS}\} \quad (10)$$

2. **Path length boundaries:** The length of each courier's path must be between 3 (because we have to consider the origin node twice) and MAX_ITEMS:

$$3 \leq \text{path\_length}[c] \leq \text{MAX\_ITEMS} \quad \forall c \in \text{Couriers} \quad (11)$$

3. **Start and End position:** Each courier's path must start and end at the depot (node $n+1$):

$$\text{path}[c][1] = n+1 \quad \text{and} \quad \text{path}[c][\text{path\_length}[c]] = n+1 \quad \forall c \in \text{Couriers} \quad (12)$$

4. **Unvisited items set to zero:** Any positions in the path beyond the path length are set to zero:

$$i > \text{path\_length}[c] \implies \text{path}[c][i] = 0 \quad \forall c \in \text{Couriers}, \forall i \in \{1, \ldots, \text{MAX\_ITEMS}\} \tag{13}$$

5. **Unique item assignment.** Each item must be assigned to exactly one courier:

$$\sum_{c \in \text{Couriers}} \text{If}(b\_path[c][j], 1, 0) = 1 \quad \forall j \in \text{Items} \tag{14}$$

6. **Maximum items constraint:** No courier can carry more than the maximum number of items:

$$\sum_{j \in \text{Items}} \text{If}(b\_path[c][j], 1, 0) \leq \text{MAX\_ITEMS} \quad \forall c \in \text{Couriers} \tag{15}$$

7. **No revisit constraint:** Each courier cannot visit the same node more than once:

$$\text{distinct\_except}(\{\text{path}[c][j] : 1 \leq j \leq \text{MAX\_ITEMS}\}, \{0\}) \quad \forall c \in \text{Couriers} \tag{16}$$

8. **Channeling constraints:** Ensures consistency between the boolean assignment and the actual path taken by each courier c, such that the *i-th* item is in the path if and only if the corresponding *b_path[c][i]* is evaluated to *True*.

$$b\_path[c][i] \implies \left(\bigvee_{j=1}^{\text{MAX\_ITEMS}} \text{path}[c][j] = i \right) \quad \forall c \in \text{Couriers}, \forall i \in \text{Items} \tag{17}$$

$$\neg b\_path[c][i] \implies \left(\bigwedge_{j=1}^{\text{MAX\_ITEMS}} \text{path}[c][j] \neq i \right) \quad \forall c \in \text{Couriers}, \forall i \in \text{Items} \tag{18}$$

9. **Load capacity:** Ensures the load does not exceed the courier's capacity, taking into account also the visited nodes:

$$\sum_{j \in \text{Items}} \text{If}(b\_path[c][j], \text{size}[j], 0) \leq \text{load}[c] \quad \forall c \in \text{Couriers} \tag{19}$$

$$\sum_{j=2}^{\text{MAX\_ITEMS}} \text{If}(j < \text{path\_length}[c]-1, \text{size}[\text{path}[c][j]], 0) \leq \text{load}[c] \quad \forall c \in \text{Couriers} \tag{20}$$

The two constraints, while related, are not entirely redundant (which is a consequence of the channeling constraint): they address the load capacity issue from different perspectives, providing the solver with additional structure that can guide it toward better solutions.

10. **Distance computation:** Calculates the total distance for each courier's path, including the additional distance if the path includes zeros between non-zero items:

$$\begin{aligned}\text{total\_distance}[c] = &\sum_{j=1}^{\text{MAX\_ITEMS}-1} \text{If}(\text{path}[c][j] \neq 0 \land \text{path}[c][j+1] \neq 0, \\ &D(\text{path}[c][j]-1, \text{path}[c][j+1]-1), 0) \\ &+ \sum_{j=1}^{\text{MAX\_ITEMS}} \text{If}(\text{path}[c][j] = 0 \land \text{path}[c][j-1] \neq 0 \land \text{path}[c][j+1] \neq 0, \\ &D(\text{path}[c][j-1]-1, \text{path}[c][j+1]-1), 0) \quad \forall c \in \text{Couriers}\end{aligned} \tag{21}$$

Symmetry breaking: Prevents symmetric solutions by ordering the assignment of items to couriers with equal load capacities:

$$\begin{aligned}&\forall c_1, c_2 \in \text{Couriers}, c_1 < c_2 \\ &\quad \text{load}[c_1] = \text{load}[c_2] \implies \text{lexleq}([b\_path[c_1][j] : j \in \text{Items}], [b\_path[c_2][j] : j \in \text{Items}])\end{aligned} \tag{22}$$

4.4 Validation

Experimental Design The SMT model was initially developed using Python's Z3 library and its optimizer. However, scaling to larger instances and exploring the search space proved challenging. To address this, we implemented a *bounded search strategy with iterative improvement* using the Z3 solver. This method involves progressively tightening the maximum distance, a concept akin to iterative deepening. While not identical, this approach systematically explores the search space and refines solutions within a time limit until the model's constraints render further solutions unsatisfiable.

Experimental Results For the first 10 instances, we obtained the same results for both the model that uses only the Z3 optimizer and the one that iterates over the search space. These results are displayed in Table 3.

5 MIP Model

In this study, we implemented a mixed-integer programming (MIP) model using both the Gurobi solver and the CBC (Coin-or branch and cut) solver (default solver of PuLp) to optimize courier routes.
Gurobi employs several advanced techniques to find optimal solutions, including the simplex method and the dual simplex method.
The dual simplex method allows Gurobi to efficiently handle problems that are modified incrementally, such as adding or relaxing constraints, and this allows

Instance	**Z3 + SB**	**Z3 w/out SB**
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**
11	932	952
12	686	514
13	1188	1110
14	-	-
15	-	-
16	**286**	**286**
17	-	-
18	-	-
19	454	425
20	-	-
21	-	1064

Table 3: Comparison of Z3 solver results with and without symmetry breaking constraints using a bounded iterative search and a 300-second timeout. Bold numbers represent the known optimal/best results.

us to use callback functions in order to update the upper bound of the objective function dinamically, enforcing in this way Gurobi's implementation of branch and bound algorithm.
In particular, Gurobi uses a branch-and-bound algorithm combined with cutting planes, known as branch-and-cut.
This section describes the model's decision variables, objective function, and constraints of the MIP implementation of the problem.

5.1 Decision Variables

The primary decision variables used in the model are:

- **Path Selection Variables** x_{cij}:
  - **Definition:** x_{cij} is a binary 3D matrix where $x_{cij} = 1$ if the path from node i to node j is selected in the optimal solution by courier c, and $x_{cij} = 0$ otherwise.
  - **Dimension:** $m \times n + 1 \times n + 1$, where m is the total number of couriers, n is the total number of nodes in the network, and $+1$ is given by the adding of the depot node.
  - **Purpose:** These variables construct the actual tour by indicating the paths taken between nodes by each courier.
  - **Mathematical Representation:**

$$x_{cij} \in \{0, 1\}, \quad \forall i, j \in \{1, 2, \ldots, n+1\}, \quad \forall c \in \{1, 2, \ldots, m$$

- **Distance Auxiliary Variable** d_{ci}:
  - **Definition:** d_c is an integer variable representing the total distance traveled by courier c
  - **Dimension:** m, where m is the total number of couriers.
  - **Mathematical Representation:**

$$u_c \in \{0, \ldots, max\_distance\}, \quad \forall c \in \{1, 2, \ldots, m\}$$

- **Load Auxiliary Variable** l_c:
  - **Definition:** l_c is an integer variable representing the total load carried by courier c
  - **Dimension:** m, where m is the total number of couriers.
  - **Mathematical Representation:**

$$l_c \in \{0, \ldots, max\_load\}, \quad \forall c \in \{1, 2, \ldots, m\}$$

- **Visit Order Variable** p_{ci}:

- **Definition:** p_{ci} is an integer variable representing the visit order of node i by courier c.
- **Dimension:** $m \times n$, where m is the total number of couriers and n is the total number of nodes.
- **Purpose:** These variables are used to keep track of the order in which nodes are visited (note that the order may be useless in order to reconstruct the path). They are needed to implement subtour elimination constraints.
- **Mathematical Representation:**

$$p_{ci} \in \{0, \ldots, \infty\}, \quad \forall c \in \{1, 2, \ldots, m\} \quad \forall i \in \{0, \ldots, n\}$$

- **Objective Variable** z:
  - **Definition:** z is an integer variable.
  - **Purpose:** This variables keeps track of the maximum distance traveled by all the couriers.

In the context of the parsed instance, these variables will adapt to the specific nodes and distances provided.

5.2 Objective Function

The objective function aims to minimize the maximum distance traveled. Mathematically, it is represented as:

$$\text{Minimize} \quad \max_{c \in \{1,\ldots,m\}} \sum_{i=1}^{n+1} \sum_{j=1}^{n+1} D_{ij} \cdot x_{cij}$$

where D_{ij} is the distance between node i and node j.

5.3 Constraints

- **Objective Variable Update**:

$$d_c \leq z \quad \forall c \in \{1, \ldots, m\}$$

- **Every node must be visited exactly once (except for the depot n+1)**:

$$\sum_{c=1}^{m} \sum_{i=1}^{n} x_{cij} \leq 1 \quad \forall j \in \{1, \ldots, n\}$$

$$\sum_{c=1}^{m} \sum_{i=1}^{n} x_{cij} \geq 1 \quad \forall j \in \{1, \ldots, n\}$$

- **Every courier must visit the depot as the last node**:

$$\sum_{i=1}^{n} x_{ci,n+1} = 1 \quad \forall c \in \{1, \ldots, m\}$$

- **Each courier must start from the depot**:

$$\sum_{j=1}^{n} x_{c,n+1,j} = 1 \quad \forall c \in \{1, \ldots, m\}$$

- **Couriers cannot stay still**:

$$x_{cii} \leq 0.5 \quad \forall c \in \{1, \ldots, m\}, \forall i \in \{1, \ldots, n+1\}$$

- **Distance Update**:

$$d_c \geq \sum_{\mathrm{i}=0}^{n+1} \sum_{\mathrm{j}=0}^{n+1} D_{ij} \times x_{c,i,j}, \quad \forall c \in \{0, \ldots, m-1\}$$

- **Load Update**:

$$l_c \geq \sum_{\mathrm{i}=0}^{n+1} \sum_{\mathrm{j}=0}^{n} L_j \times x_{c,i,j}, \quad \forall c \in \{0, \ldots, m\}$$

  Where L_j is the weight of the item in node j.
  This equation (i.e. dot product) holds since we are sure that there is at most one 1 in each column and in each row.

- **Path Contiguity**:

$$\sum_{k=1}^{n+1} x_{cki} \geq 1 = \sum_{k=1}^{n+1} x_{cik} \geq 1 \quad \forall c \in \{1, \ldots, m\}, \forall i \in \{1, 2, \ldots, n+1\}$$

  This is like a forward and backward check in order to further speed up the search.

- **Bind the number of 1's for each row and column**:

$$\sum_{j=1}^{n+1} x_{cij} \leq 1 \quad \forall c \in \{1, \ldots, m\}, \forall i \in \{1, 2, \ldots, n+1\}$$

$$\sum_{i=1}^{n+1} x_{cij} \leq 1 \quad \forall c \in \{1, \ldots, m\}, \forall j \in \{1, 2, \ldots, n+1\}$$

  It is an explicit constraint used just in order to delete useless assignments from the search space.

- **Loop Avoidance**:

  $p_{c,i} - p_{c,j} + n \times x_{c,i,j} \leq n-1 \quad \forall c \in \{1,\ldots,m\}, \forall i \in \{1,2,\ldots,n\}, \forall j \in \{1,2,\ldots,i\}$

  In the literature this constraint is known as subtour elimination constraint and ensures that the path is continuous and does not have isolated closed loops.
  Each variable $p_{c,i}$ represents the index (or "position") of the node i in the courier's c path. If the courier visits node i first and then node j, we expect that $u_i < u_j$.

- **Symmetry Breaking**:

  $$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{cij} \geq \sum_{i=1}^{n}\sum_{j=1}^{n} x_{c+1,ij} \quad \forall c \in \{1,\ldots,m-1\}$$

  Under the assumption that the couriers (i.e. channels of the matrix) are disposed in decreasing order of load size.

- **Maximum Load**:

  $$l_c \leq L_c \quad \forall c \in \{1,\ldots,m\}$$

5.4 Validation

We can see the results of the MIP model in the Table below. As we can see, Gurobi provides better result with respect to CBC. This is because of the optimization power of the solver and because of the callback function that helps the naive branch and cut of Gurobi. Since CBC doesn't offer callback functions, we were unable to reproduce the same experiment in order to better compare the two.
The generalization power of PuLp allows us to use different solvers just by changing one single parameter.

6 Conclusions

In conclusion, our project successfully utilized Constraint Programming (CP) with MiniZinc, Satisfiability Modulo Theories (SMT) using Z3, and Mixed-Integer Programming (MIP) with Gurobi to optimize routing for multiple couriers across 21 instances. Notably, our results were promising, demonstrating effective solutions across diverse scenarios. Surprisingly, we achieved improved outcomes without relying on symmetry breaking constraints, highlighting the robustness and efficiency of our approach. This project underscores the versatility and effectiveness of combining different optimization techniques to tackle complex logistics and routing challenges.

Instance	GurobiPy+SB	GurobyPy w/out SB	PuLp (CBC)
1	**14**	**14**	**14**
2	**226**	**226**	**226**
3	**12**	**12**	**12**
4	**220**	**220**	**220**
5	**206**	**206**	**206**
6	**322**	**322**	**322**
7	**167**	**167**	**184**
8	**186**	**186**	**186**
9	**436**	**436**	**436**
10	**244**	**244**	**244**
11	N/A	N/A	N/A
12	541	N/A	N/A
13	**420**	**474**	**800**
14	N/A	N/A	N/A
15	N/A	N/A	N/A
16	**286**	**286**	N/A
17	N/A	N/A	N/A
18	N/A	N/A	N/A
19	N/A	N/A	N/A
20	N/A	N/A	N/A
21	N/A	N/A	N/A

References

[1] Nicolas Briot Philippe Vismara. *A Circuit Constraint for Multiple Tours Problems*. Conference paper, 2018.