Combinatorial Decision Making & Optimization Project Report

Irene Burri
irene.burri@studio.unibo.it
Laura Lucchiari
laura.lucchiari@studio.unibo.it
Luca Mercuriali
luca.mercuriali2@studio.unibo.it
Luigi Manieri
luigi.manieri2@studio.unibo.it

1 Introduction

In this report, we present our work on the Multiple Couriers Planning problem, which we approached using four different computational methods: Constraint Programming (CP), Satisfiability Testing (SAT), Satisfiability Modulo Theories (SMT), and Mixed-Integer Programming (MIP).

The input instances were provided in `.dat` files. To streamline data processing, we developed a universal parsing function applicable across all approaches, enabling us to extract key variables from each instance. Specifically, for each instance we obtained: the number of couriers m, the number of packages n, the list of maximum load capacities l of the couriers, the list of package sizes s, and the distance matrix D between distribution points.

Our objective is to minimize the maximum distance traveled by the couriers. To achieve this, we defined a common objective function across all approaches, each aiming to find the optimal solution for the given instance.

We began by defining some key concepts related to variables and constraints, which are common across the four methods:

- Weight Constraint: We define a weight variable so that the total weight of packages assigned to each courier cannot exceed the courier's maximum load capacity.
- Distance Optimization: The distance between the starting and ending points for each courier, which we aim to minimize in the objective function.
- Courier Assignment Variable: A variable that associates each courier with the specific packages or route it will follow.

A time limit of 300 seconds was imposed on the solving process. When this limit is reached, the best solution found so far is returned, even if optimality has not been achieved. For some instances, no solution could be found within the given time limit.

We started by working together to understand the overall goals of the project. Then, we divided the workload, with each team member focusing on a specific method. Specifically, Luca Mercuriali worked on CP, Laura Lucchiari on SAT, Luigi Manieri on MIP, and Irene Burri on SMT. The project took approximately five weeks, with the main challenges being familiarizing ourselves with the different types of implementations and setting up the required environments for each approach.

2 CP Model

Before defining the decision variables, we introduce a critical constraint to limit the maximum number of stops per courier. Initially, we considered enforcing a rule where each courier must carry at least one item. Under this approach, the maximum number of stops a courier could make was calculated as $(n-m+1)+2$, where the additional 2 accounts for the starting and ending points. However, this constraint did not significantly reduce the search space.
To achieve a more balanced distribution of items among couriers, we refined our approach and introduced the variable:

$$limit = ceil(1.5 * n/m) + 2 \tag{1}$$

Here, the factor 1.5 is manually tuned to prevent the constraint from being overly restrictive while still ensuring a fair allocation of stops among couriers.

2.1 Decision Variables

The solution of this problem in CP is modelled around the key decision variables:

$$journeys[i,j] = p \in \{1,\dots,n+1\} \tag{2}$$

where $journeys[i,j]$: is a matrix with m rows representing each courier and ($limit$) columns representing the number of stops. This matrix captures the sequence of deliveries completed by the couriers therefor its entries denote the index of each package delivered by a specific courier. n+1 represents the starting and ending point (the deposit).
We defined the decision variables:

$$item\_bin[k] = i \in \{1,\dots,m\} \tag{3}$$

where $item\_bin[k]$ indicates which courier carries package $k \in \{1,\dots,n\}$. The relationship between item_bin and journeys is expressed by the following constraint:

$$\forall i\,\forall j\,\Big(\big(1 \le i \le m \;\wedge\; 2 \le j \le \text{limit}-1 \;\wedge\; \text{journeys}(i,j) \ne n+1\big) \rightarrow \text{item\_bin}\big(\text{journeys}(i,j)\big) = i\Big) \tag{4}$$

We also defined the variable:

$$distances[i] = \sum_{j=1}^{limit-1} D\big[journeys[i,j], journeys[i,j+1]\big] \tag{5}$$

which represents the distance traveled by each courier. We considered as lower bound for the distance variable:

$$low\_bound\_dist = \min_{i \in 1..n} \big(D[n+1,i] + D[i,n+1]\big) \tag{6}$$

while the upper bound is given by:

$$up\_bound\_dist = D[n+1,1] + \sum_{i=1}^{n-1} D[i,i+1] + D[n,n+1] \tag{7}$$

2.2 Objective function

The objective is to minimize the maximum distance traveled by any courier. We have therefore defined the objective function as follows:

$$\text{z} = \max_{i=1}^{m} \text{distances}[i]]) \tag{8}$$

The lower bound for the objective function is given by:

$$\text{low\_bound\_obj} = \max_{i \in 1..n} \big(D[n+1,i] + D[i,n+1]\big) \tag{9}$$

The upper bound is the equal to the one of the variable distance.

2.3 Constraints

We have defined the following constraints:

1. Each courier tour must start and end at the base (n+1)

$$\forall i \in 1..m \big(journeys[i,1] = n+1\big) \tag{10}$$

$$\forall i \in 1..m \big(journeys[i,limit] = n+1\big) \tag{11}$$

2. Each courier can carry a weight less than the maximum load they can support. The weight constrain is defined with the following global constraint:

$$\text{bin\_packing\_capa}(l, item\_bin, s) \tag{12}$$

3. If a courier's journey reaches the origin point (n + 1), all subsequent points in the row must also be the origin point, ensuring that once a courier ends its tour, no further deliveries are made.

$$\forall i \in 1..m \Big(\forall j \in 2..limit-1 \big((journeys[i,j] = n+1) \rightarrow \forall k \in j+1..limit\,(journeys[i,k] = n+1)\big)\Big) \tag{13}$$

4. Each courier must deliver at least an item:

$$\forall i \in 1..m \Big(\exists j \in 2..limit-1 \big(journeys[i,j] \neq n+1\big)\Big) \tag{14}$$

5. The second position in each journey is not the origin point:

$$\forall i \in 1..m \left(\text{journeys}[i,2] \neq n+1\right) \tag{15}$$

6. Each item is assigned to exactly one courier and appears only once:

$$alldifferent\Big(\big[journeys[i,j] \mid i \in 1..m,\ j \in 2..limit-1 \\ where journeys[i,j] \neq n+1\big]\Big) \tag{16}$$

7. **Symmetry breaking constraint** : If two couriers have the same maximum capacity, we have a symmetry:

$$\forall c_1, c_2 \in \{1,\ldots,m\} \text{ with } c_1 < c_2,\ (l[c_1] = l[c_2]) \rightarrow \\ \rightarrow \text{lex less}(\text{row}(\text{journeys}, c_1), \text{row}(\text{journeys}, c_2)) \tag{17}$$

This constraint considers the situation of two couriers with same capacity: in that case they are symmetric so we can impose an ordering between them for the packages they pick up.

2.4 Validation

Experimental design The CP model has been tested using the Gecode and the Chuffed solvers. We experimented with different type of search heuristic; the integer search strategy which resulted more effective was first fail with indomain_split.

Experimental results Notably, with the Gecode solver, we managed to find results for all the 21 instances. In particular, we obtained optimal results for all the "easy instances" (those up to the tenth) and also for the 16th and 19th instances. Similar results were obtained with the Chuffed solver.

Instances	Gecode	Chuffed	Instances	Gecode	Chuffed
1	**14**	**14**	12	**346**	**346**
2	**226**	**226**	13	1806	1706
3	**12**	**12**	14	1321	1505
4	**220**	**220**	15	1038	1108
5	**206**	**206**	16	**286**	**286**
6	**322**	**322**	17	1363	1389
7	**167**	**167**	18	1085	1122
8	**186**	**186**	19	**334**	**334**
9	**436**	**436**	20	1680	1748
10	**244**	**244**	21	673	880
11	572	736			

Table 1: Results for CSP method, bolded elements indicating the optimal result.

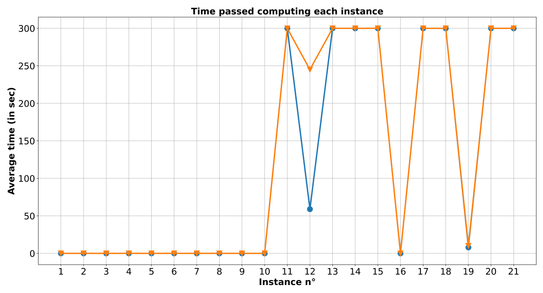

Figure 1: CSP Computing time for each instance (blue line stands for Gecode, orange for Chuffed).

3 SAT Model

3.1 Decision Variables

In our SAT model implementation, we define three multi-dimensional arrays as decision variables, consistent with those introduced earlier since they are common to all implementations.

The first decision variable links the courier, the package they are carrying, and the time at which the courier carries the package. The other two are auxiliary variables: one computes the total weight carried by each courier, while the other calculates the total distance traveled by each courier.

- journeys$[c, t, p] = 1 \Leftrightarrow$ the courier **c** carries the package **p** at time **t**

  Here, **c** represents the courier, which has a domain of $[0, \ldots, m-1]$, where m is the total number of couriers. **t** denotes the time, with a domain of $[0, \ldots, ceil(1.5 * n/m) + 2]$ (the reason for this boundary is explained in the CP part). Finally, **p** represents the package, which has a domain of $[0, \ldots, n]$, where n is the total number of packages.
  Once a courier has delivered all of their packages, they will remain at the base for the rest of the time.

- weight$[c, p] = 1 \Leftrightarrow$ the courier **c** carries the package **p**

- distances$[c, s, e] = 1 \Leftrightarrow$ the courier **c** goes from start to end at some time

3.2 Objective function

The optimization is performed using a loop that determines the maximum allowable distance for each courier in each iteration. Initially, the minimum and maximum distances are calculated using a simple heuristic function.

Then, for each courier, we create a list representing the total distance traveled. Ensure that this total distance does not exceed k:

$$\text{at\_most\_k}(\text{c}_d, k) \tag{18}$$

where c_d is the list calculated as:

$$[\text{repeat}(d_{c,p_1,p_2}, D[p_1][p_2]) \mid \forall p_1, p_2 \in [0, n]] \tag{19}$$

The solver checks the satisfiability of the model:

- If not satisfiable, update the **minimum distance** to k.
- If satisfiable, store the current best model and construct a solution matrix, indicating the package carried by each courier at each time step.

This process continues until the value of k stabilizes, ensuring the problem remains satisfiable and minimizing the total distance traveled before becoming unsatisfiable (**UNSAT**).

3.3 Constraints

In our optimization model, we define a set of constraints and symmetry-breaking rules that ensure the solution adheres to the requirements of the courier and package assignment problem. Each constraint plays a critical role in maintaining the feasibility of the model while guiding the search for an optimal solution.

1. Weight Variable Constraint:

$$\forall c \in \text{courier\_range}, \forall t \in \text{time\_range}, \forall p \in \text{package\_range} :$$

$$(y[c][t][p] \Rightarrow \text{weights}[c][p]) \tag{20}$$

If courier c carries package p at time t, then the weight variable associated with that package must be true. This ensures that the total weight carried by the courier is accurately represented.

2. Distances Variable Constraint:

$$\forall c \in \text{courier\_range}, \forall t \in \text{time\_range\_no\_zero}, \forall p_1, p_2 \in \text{package\_range} :$$

$$(p_1 \neq p_2 \Rightarrow (y[c][t-1][p_1] \wedge y[c][t][p_2] \Rightarrow \text{distances}[c][p_1][p_2])) \tag{21}$$

If courier c transitions from package p_1 at time $t-1$ to package p_2 at time t, the distance variable between these packages must be recorded. This ensures that the couriers' movements are accurately tracked in terms of distances traveled.

3. Single Package or Base Constraint:

$$\forall c \in \text{courier\_range}, \forall t \in \text{time\_range} : (\text{exactly\_one}(y[c][t][:])) \tag{22}$$

At each time t, a courier c can either carry exactly one package or be at the base. This ensures that a courier does not carry more than one package simultaneously.

4. Package Assignment Constraint:

$$\forall p \in \text{package\_range}: \\ (p \neq \text{base\_package} \Rightarrow \text{exactly\_one}(y[c][t][p] \mid c \in \text{courier\_range}, t \in \text{time\_range})) \tag{23}$$

This ensure that each package, with the exception of the base package, must be assigned to exactly one courier.

5. Weight Capacity Constraint:

$$\forall c \in \text{courier\_range}: \\ (\text{at\_most\_k}([\text{weights}[c][p] \mid p \in \text{package\_range}, \text{for } s[p]], l[c])) \tag{24}$$

The total weight carried by each courier c must not exceed their maximum capacity $l[c]$.

6. Base Start and End Constraint:

$$\forall c \in \text{courier\_range}: (y[c][0][\text{base\_package}] \wedge y[c][\text{last\_time}][\text{base\_package}]) \tag{25}$$

Each courier is required to begin and end their journey at the base location.

3.3.1 Optimization Constraints:

7. Delivery Before Base Return Constraint:

$$\forall c \in \text{courier\_range}, \forall t \in \text{time\_range\_no\_zero}: \\ (y[c][t][\text{base\_package}] \Rightarrow \forall t_2 (t < t_2 < \text{last\_time} \Rightarrow y[c][t_2][\text{base\_package}])) \tag{26}$$

Couriers must first deliver all their assigned packages before returning to the base.

8. Stay at Base After Return Constraint:

$$\forall c \in \text{courier\_range}, \forall t \in \text{time\_range\_no\_zero}: \\ (y[c][t][\text{base\_package}] \Rightarrow \bigwedge_{t'=t}^{\text{last\_time}} y[c][t'][\text{base\_package}]) \tag{27}$$

Couriers must remain at the base once they have returned.

3.3.2 Symmetry Breaking Constraints:

9. Lexicographic constraint:

$$\forall c_1, c_2 \in \text{courier\_range}: (c_1 < c_2 \wedge l[c_1] = l[c_2] \Rightarrow \text{lex\_less}(y[c_1], y[c_2])) \tag{28}$$

If two couriers have the same capacity, imposing an order on the packages they pick up helps to break symmetry. This is essential for optimizing search algorithms and reducing redundant solutions.

Conclusion

The combination of these constraints ensures that our optimization model accurately reflects the real-world requirements of the courier and package assignment problem. By employing first-order logic to express these constraints, we provide a rigorous and formal framework that guarantees the feasibility of the solution. Furthermore, the added symmetry-breaking constraints enhance the efficiency of the optimization process, enabling quicker convergence to the optimal solution.

3.4 Validation

3.4.1 Experimental design

The experimental design optimizes courier routes to minimize travel distance in package deliveries. Using iterative optimization with binary search, the model adjusts distance bounds in each iteration. Constraints ensure couriers meet distance and weight limits, deliver each package once, and return to base. Symmetry-breaking constraints prevent identical solutions for couriers with the same capacity. The solver narrows the range until an optimal solution is found, minimizing travel distance just before reaching unsatisfiability (UNSAT).

3.4.2 Experimental results

Instances	SAT
1	**14**
2	**226**
3	**12**
4	**220**
5	**206**
6	**322**
7	**168**
8	**187**
9	**436**
10	**244**
13	1672
16	517
19	843

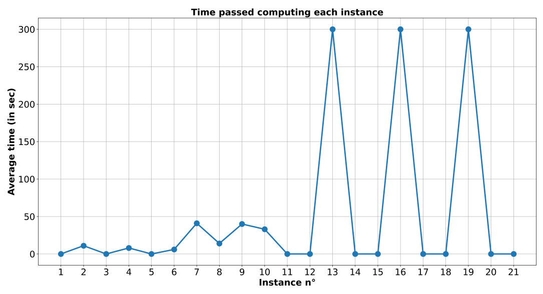

Figure 2: In the table, the results for SAT method, bolded elements indicate optimal results, while missing elements stand for N/A.
The graph shows SAT computing time for each instance.

4 SMT Model

4.1 Decision Variables

Also for this model, the decision variables are represented by a 3-dimensional binary array, where each element is a binary variable,

$$journeys_{p,t,c} \in \{0,1\} \tag{29}$$

that indicates whether a courier c is assigned to deliver a package p at a specific time t.

The domains of these variables are defined as follows:

$$\begin{aligned} c &\in \{0,\dots,m-1\} && \text{for each courier,} \\ t &\in \{0,\dots,ceil(1.5\cdot n/m)+2\} && \text{for each discrete time slot (explained in 1),} \\ p &\in \{0,\dots,n\} && \text{for each package.} \end{aligned}$$

including a dummy package with a weight of zero representing the base station.

This binary structure allows us to determine whether a courier is actively delivering a package at any given time, represented by $journeys_{p,t,c} = 1$ if courier c takes package p at time t and $journeys_{p,t,c} = 0$ otherwise.

To support optimization constraints and objectives, we created auxiliary variables that collectively allows the model to define and optimize the assignment:

- *courier_weights* a weights array to track each courier's total load
- *travel_distances* a distances array for the distance each courier travels
- *max_distance* a variable to monitor the maximum distance traveled by any courier

4.2 Objective function

For the same logic explained in SAT we used an iterative loop where we set an initial limit on the maximum distance traveled by any courier, whether the solver cannot find a valid solution, we gradually increase this limit until a solution is found.

$$\text{Minimize} \quad max\_distance \tag{30}$$

4.3 Constraints

1. Ensure each package is picked up exactly once, except the final package:

$$\sum_{c\in courier\_indices} \quad \sum_{t\in time\_slots} journeys_{p,t,c} = 1 \tag{31}$$

$$\forall p \in package\_indices, p \neq \text{final\_package}$$

2. Ensure valid time slots and couriers are assigned to packages, each courier can deliver only one package at a time:

$$\sum_{p \in package\_indices} journeys_{p,t,c} = 1 \tag{32}$$
$$\forall c \in courier\_indices, \quad \forall t \in time\_slots$$

3. Each courier carries at least one package, except for the final package:

$$\sum_{\substack{p \in package\_indices \\ p \neq \text{final\_package}}} \sum_{t \in time\_slots} journeys_{p,t,c} \geq 1 \qquad \forall c \in courier\_indices \tag{33}$$

4. Ensure the final package is the first and last in the route for each courier:

$$(journeys_{\text{final\_package},0,c} = 1) \quad \text{AND} \quad (journeys_{\text{final\_package},\text{last\_time},c} = 1) \tag{34}$$
$$\forall c \in courier\_indices$$

5. Enforce weight constraint:

$$weights[c] \leq weight\_limits[c] \qquad \forall c \in courier\_indices \tag{35}$$

6. Each courier can return to the base only after delivering all the packages they are carrying. We have added this constraint to reduce the search space:

$$(journeys_{\text{final\_package},t,c} = 1) \Rightarrow (journeys_{\text{final\_package},t',c} = 1) \tag{36}$$
$$\forall c \in courier\_indices, \quad \forall t \in time\_slots, \quad \forall t' \in \{t+1, \ldots, \text{final\_time\_slot}\}$$

4.3.1 Symmetry breaking constraints

To eliminate symmetries in the solution space, we impose the following constraint:

7. If two couriers c_1 and c_2 have the same capacity, enforce lexicographical ordering. In the case of couriers having the same maximum load capacity, an increasing number of equivalent solutions will arise as the number of couriers with the same capacity increases.

$$\begin{aligned} &\text{if } c_1 < c_2 \text{ and } l[c_1] = l[c_2] \\ &\qquad \Longrightarrow \text{lexicograph\_less}(journeys_{c_1}, journeys_{c_2}) \quad \forall c_1, c_2 \in C \end{aligned} \tag{37}$$

where:

$$journeys_{c_1} = [journeys_{p,t,c_1} \mid \forall t \in \text{time\_slots}, \forall p \in \text{package\_indices}]$$
$$journeys_{c_2} = [journeys_{p,t,c_2} \mid \forall t \in \text{time\_slots}, \forall p \in \text{package\_indices}]$$

4.4 Validation

4.4.1 Experimental design

The model was implemented using Z3Py because it supports the SMT solver. Like in other methods, we set a runtime limit of 300 seconds. During the process, we noticed that adding a symmetry-breaking constraint made the performance slower. This constraint created symmetry when the maximum capacity of one courier was smaller than the minimum capacity of another.

4.4.2 Experimental results

Instances	SMT
1	**14**
2	**226**
3	**12**
4	**220**
5	**206**
6	**322**
7	190
8	**186**
9	**436**
10	**244**

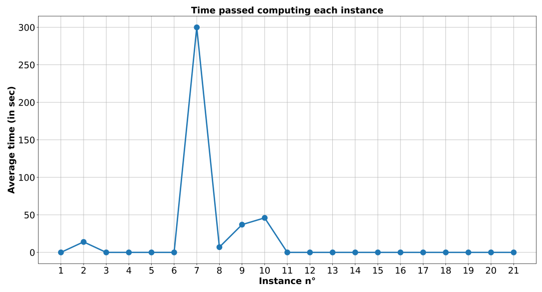

Figure 3: In the table, the results for SMT method, bolded elements indicate optimal results, while missing elements stand for N/A.
The graph shows SMT computing time for each instance.

5 MIP Model

5.1 Decision Variables

For this model, we use a single decision variable *journeys*, which is a 3D boolean matrix of size $[1..m, 1..n+1, 1..n+1]$ representing couriers along one dimension and locations along the other two dimensions.

$$journeys_{c,l_1,l_2} \in \{0,1\}$$

$$\text{where} \quad journeys_{c,l_1,l_2} = 1 \Leftrightarrow \text{courier } c \text{ travels from } l_1 \text{ to } l_2. \tag{38}$$

Additionally, the model includes three auxiliary variables:

- An array *weights* of length m, defining the weight carried by each courier.
- An array *distances* of length m, capturing the total distance traveled by each courier.

- A 2D matrix of size $[1..m, 1..n]$ *path_increment*, recording the time steps at which each courier visits each location l. Specifically, $path\_increment_{c,l}$ denotes the time step when courier c visited location l.
- A variable *max_distance* storing the maximum distance traveled by any courier.

5.2 Objective Function

The objective is to minimize the maximum distance traveled by any courier.

$$\text{Minimize} \quad max\_distance$$

$$\text{with} \quad max\_distance \geq distances[i] \quad \forall\, i \in \{1, 2, \ldots, m\} \tag{39}$$

5.3 Constraints

In MIP we use some of the same constraints used in the other models plus some others in order to impose the correctness of the path of each courier.

1. If we have the transition $l1 \longrightarrow l2$, then we have to have a transition $l3 \longrightarrow l1$ that took us to l1:

$$\text{If } journeys_{c,l1,l2} = 1 \text{ then } \exists l_3 \in L \text{ s.t } journeys_{c,l3,l1} = 1$$

More formally:

$$\sum_{l3 \in L} journeys_{c,l3,l1} \geq journeys_{c,l1,l2} \quad \forall c \in C, \forall l_1, l_2 \in L \tag{40}$$

2. We impose the value of base to 0 in path_increment variable for each courier meaning that it is always the starting point of a path:

$$path\_increment_{c,n+1} = 0 \quad \forall c \in C \tag{41}$$

3. We want to make sure that if $journeys_{c,l1,l2}$ is 1 than $path\_increment_{c,l2} = path\_increment_{c,l1} + 1$:

$$\begin{gathered} path\_increment_{c,l2} \geq path\_increment_{c,l1} + 1 - n \cdot (1 - journeys_{c,l1,l2}) \\ path\_increment_{c,l2} \leq path\_increment_{c,l1} + 1 + n \cdot (1 - journeys_{c,l1,l2}) \\ \forall c \in C, \forall l_1 \in L, \forall l_2 \in L \setminus \{n\} \end{gathered} \tag{42}$$

4. Since path_increment represent the time-step at which each courier take a package we want the package not taken by a courier to have value 0 and the other to have max value n:

$$path\_increment_{c,l} \leq \sum_{l_2 \in L} journeys_{c,l,l_2} \cdot (n+1) \quad \forall c \in C, \forall l \in L \tag{43}$$

5. Each courier can't exceed it's weight limit:

$$weight_c \leq max\_weight[c] \quad \forall c \in C \tag{44}$$

6. No cycles in the same location:

$$journeys_{c,l1,l1} \leq 0 \quad \forall l_1 \in L \quad journeys_{c,l1,l1} \geq 0 \quad \forall l_1 \in L \tag{45}$$

7. Every courier starts his journey at the base:

$$\sum_{l \in L} journeys_{c,l,n} \leq 1 \quad \sum_{l \in L} journeys_{c,l,n} \geq 1 \forall c \in C \tag{46}$$

8. Every courier ends his journey at the base:

$$\sum_{l \in L} journeys_{c,n,l} \leq 1 \quad \sum_{l \in L} journeys_{c,n,l} \geq 1 \quad \forall c \in C \tag{47}$$

9. Package are carried only once:

$$\sum_{c \in C} \sum_{l_2 \in L} journeys_{c,l_1,l2} \leq 1 \quad \sum_{c \in C} \sum_{l_2 \in L} journeys_{c,l_1,l2} \geq 1 \quad \forall l_1 \in L \setminus \{n\} \tag{48}$$

10. Each courier leaves each location:

$$\sum_{l3 \in L} journeys_{c,l2,l3} \geq journeys_{c,l1,l3} \quad \forall c \in C, \forall l_1 \in L, \forall l_2 \in L \setminus \{n\} \tag{49}$$

11. The last constraint we impose is on *max_distance*:

$$max\_distance \geq distance_c \quad \forall c \in C \tag{50}$$

5.4 Validation

5.4.1 Experimental Design

The model was implemented using the library python-mip and it was tested using: BalancedMIP emphasis, which uses the default search settings, FeasibilityMIP emphasis, which focuses on finding a feasible solution and improving it, and OptimalityMIP, which immediately starts searching for the optimal solution. Both CBC and Gurobi were used as solvers. For Gurobi solver, an academic license was required.

The three different emphasis approaches obtained the same results in the exact same times, so they are not reported here, and we will use the default option (Balanced) in the results shown below. Additionally, only the Gurobi results are displayed since they were better (shorter times) than CBC for all the instances.

MIP approach was limited to computational time: the solved instances were solved optimally within the timeout and with all the different emphasis, while for the bigger ones no solution was ever found within the timeout (300s), meaning that increasing the time limit could bring to more optimal solution.

5.4.2 Experimental Results

Instance	MIP
1	**14**
2	**226**
3	**12**
4	**220**
5	**206**
6	**322**
7	**167**
8	**186**
10	**244**

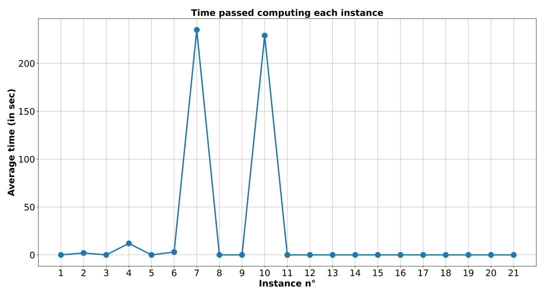

Figure 4: In the table, the results for MIP method, bolded elements indicate optimal results, while missing elements stand for N/A.
The graph shows MIP computing time for each instance.

6 Conclusions

This project helped us to understand how to approach a Combinatorial Decision Making problem, exploring different approaches and techniques. By implementing CP, SAT, SMT and MIP models, we compared their performance in terms of solution quality and computational efficiency.

CP consistently provided solutions within the time limit, often providing the optimal one, SAT performed well while SMT and MIP were effective for smaller cases but struggled with larger instances due to computational complexity. Overall, all the approaches were able to find the optimal solution on the first 10 instances.

Many constraints were specific to the problem itself and so they are common in all the presented models, while some were model specific. At the same time, it was interesting to notice that increasing the number of constraints didn't always correspond to better performances, getting them even worse in some cases.

CP model was the one that made the most use of global constraints, hence improving greatly the results, it was the only model able to find a solution for all the instances. A significant improvement in CP was obtained by tweaking the limit dimension of the main decision variables (1); applying the same bound to the decision variable also in other models brought as well an improvement of performances in SAT and SMT.

For the starting point and the definition of the courier assignment variable, used in each approach, we found the idea in some projects of the previous year, such as [1]

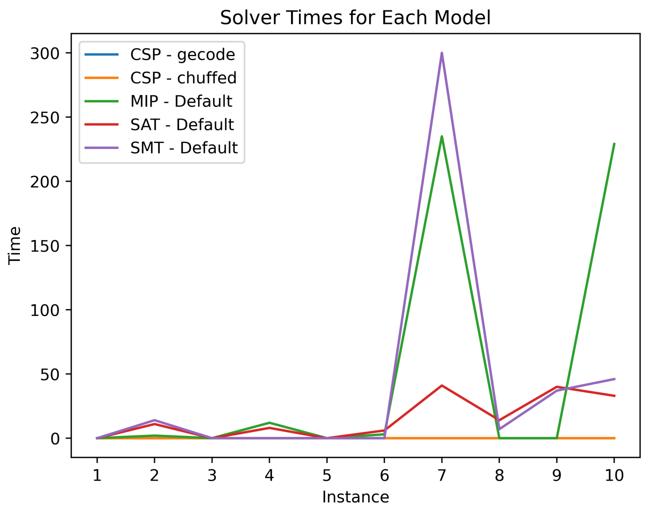

Figure 5: Computing time for first 10 instances with all models

This project improved our understanding of optimization methods, solver behaviors, and problem modeling, highlighting the importance of choosing the right approach based on problem constraints and resources.

References

[1] Umberto Carlucci, Giuseppe Carrino, Matteo Vannucchi, *GitHub "Multiple Couriers CDMO" repository*. Disponibile su: `https://github.com/JosephCarrino/Multiple_Couriers_CDMO` (Accesso: 2 febbraio 2025).