Project Work Combinatorial Decision Making & Optimization 2023/2024

Leonardo Chiarioni
Jana Nikolovska
Andrea Cristiano
{ leonardo.chiarioni, jana.nikolovska, andrea.cristiano }@studio.unibo.it

February 2, 2025

1 Introduction

In this report, we will explore how the problem of the Multiple Couriers Planning (MCP) is modeled and discuss various approaches to address it: Constraint Programming (CP), Satisfiability Modulo Theories (SMT) and Mixed-Integer Linear Programming (MIP). The project was completed by a group of three, with the modeling phase and boundary formula derivation done collaboratively. Following this, the implementation and specific adjustments for each approach were divided: Leonardo Chiarioni handled CP, Jana Nikolovska focused on SMT, and Andrea Cristiano worked on MIP. This division allowed for both teamwork and individual specialization.

1.1 Description of the problem

Given a set of items and a fleet of couriers, together with the respective weights and capacities, the MCP problem consists in finding a route for each courier so to deliver each item to its corresponding location. The goal is to minimize the maximum distance travelled by any of the courier. The couriers' journey start and end at the dispatch center, defined as *depot*, and the distance matrix D between the nodes is known to be asymmetric and in which triangle inequality holds:

$$\forall i, j, k \in N : D_{i,k} \leq D_{i,j} + D_{j,k} \tag{1}$$

That said, it's easy to derive that is always convenient for each courier to deliver at least an item.

1.2 Parameters and Domains

Each of the approach tackled in solving the problem rely on the specification of the parameters for every instance to solve. In table 1 they are presented with their respective meaning.

Name	Dimension	Description
m	1	number of couriers
n	1	number of items
l	m	maximum load for each courier
w	n	weight of each item
D	$(n+1) \times (n+1)$	distance matrix

Table 1: Parameters of an instance of the problem.

To maximize the readability of the document, also the sets shown in table 2 have been defined. We will refer to these tables in our formulas across the entire document.

Name	Elements CP	Elements SMT/MIP	Description	Items of the set
C	$[1..m]$	$[0..m-1]$	Couriers	c
I	$[1..n]$	$[0..n-1]$	Items	p, q
N	$[1..n+1]$	$[0..n]$	Nodes	i, j, k

Table 2: Sets used for readability purposes.

1.3 Objective function

The objective function is the maximum distance traveled by any of the couriers, which needs to be minimized. The exact formulation and variables used to model this objective may vary depending on the specific implementation or approach, but the ultimate goal remains the same.

1.4 Bounds

To effectively reduce the search space of the solutions it is fundamental to constrain the objective value between an upper and a lower bound.

Lower bound Since we are minimizing the maximum distance travelled by any of the courier, we can consider the best possible scenario ($m = n$) in which every courier carries a package. In this specific case the maximum distance will be:

$$LB = \max_{i \in N} D_{depot,i} + D_{i,depot} \tag{2}$$

which can be set as the lower bound.

Upper bound Even though the upper bound is not as critical as the lower bound, we can consider the worst case scenario in which all the packages are assigned to only a courier, which will cover the longest route to visit all the nodes. In this case the maximum distance will be:

$$UB = \max_{i \in N} \sum_{j \in N} D_{i,j} \tag{3}$$

which can be set as the upper bound. It is worth to note that this will never be the case since (1) holds.

1.5 Hardware

The testing phase for each of the three approaches happened on the personal devices of the developers. For the CP part, Leonardo Chiarioni used an Acer Aspire 5 (Intel Core i7, 8GB RAM); for the SMT part Jana Nikolovska used an Acer ConceptD 3 (Intel Core i7, 16GB RAM); for the MIP part Andrea Cristiano used a 16’ MacbookPro with M1Pro processor, 16GB of RAM.

2 CP model

The CP model is very simple yet effective, mainly relying on channeling as well as implied constraint to find routes for our couriers. The core idea of the model is to find a function *assign* and a function *path* so to assign to each courier the items to deliver and to find its corresponding route.

Starting from this, the model has been improved and adapted by exploiting global constraints and auxiliary variables to improve the overall performance. Every function is represented as a d-dimensional array variable, with d being the number of arguments accepted by the function itself.

2.1 Decision variables

The following decision variables have been used:

- *assign* : I → C; indicates which courier delivers which item. More specifically, $assign(p) = c$ means that item p is delivered by courier c.
- *path* : C x N → N; defines the route for each courier c by means of "next hops". More specifically:

$$path(c, i) = \begin{cases} j & c \text{ travels } i \to j \\ i & i \text{ is not in the path} \end{cases} \tag{4}$$

2.2 Auxiliary variables

In order to ease the modeling of the constraints and the objective of the task, the following auxiliary variables have been used:

- *packages* : C x I → {0,1}; indicates which courier delivers which item. More specifically:

$$packages(c, p) = \begin{cases} 1 & c \text{ delivers } p \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

- *a_l* : C → $\mathbb{Z}$; keeps track of the actual load of each courier. More specifically, $a\_l(c) = 20$ means courier c has been filled with packages whose total weight is 20.
- *total_distance* : C → $\{LB..UP\}$; contains the distance travelled by each courier. More specifically, $total\_distance(c) = 100$ means courier c 's path is such that its total distance is 100.

2.3 Constraints

The constraints will be presented in different sections: *path, load, channeling* and *symmetry breaking* constraints.

2.3.1 Path constraints

We need to ensure that the path of each courier is such that nodes cannot be visited multiple times:

$$\forall c : \textit{alldifferent(path[c,N])} \tag{6}$$

Every courier's path must be an hamiltonian sub-cycle:

$$\forall c : \textit{subcircuit(path[c,N])} \tag{7}$$

Every package must be shipped once and by only one courier:

$$\forall p : \sum_{c} packages[c, p] = 1 \tag{8}$$

Thanks to (1), we know that each courier must ship at least 1 package. We can express this by constraining the last column of *path*, which represent the *depot*:

$$\begin{cases} \textit{alldifferent}(path[C, n+1]) \\ \forall c : path[c, n+1] \neq n+1 \end{cases} \tag{9}$$

Each courier's route must start and end at the depot: this is already encoded by design in the *path* variable since every path is an hamiltonian loop. We can easily constrain the total distance of each courier to be:

$$\forall c : total\_distance[c] = \sum_{i} D_{i,path[c,i]} \tag{10}$$

2.3.2 Load constraints

In order to assign packages to couriers such to respect the maximum capacity, both global and simple constraints have been modeled and **separately tested**:

$$bin\_packing\_capa(l, assign, w) \tag{11}$$

$$\forall c : \sum_{\substack{p: \\ path[c,p] \neq p}} w[p] \leq l[c] \tag{12}$$

2.3.3 Channeling constraints

In order to have consistency between the auxiliary and the decision variables we need to relate them by using channeling constraints:

$$\forall c, \forall p : \begin{cases} assign[p] \neq c \iff path[c,p] = p \iff packages[c,p] = 0 \\ assign[p] = c \iff path[c,p] \neq p \iff packages[c,p] = 1 \end{cases} \tag{13}$$

2.3.4 Symmetry breaking constraints

Symmetry has been broken using 2 constraints. The first one, which will be referred as *simple*, breaks symmetry by imposing lexicographic ordering on all the couriers having the same capacity:

$$\forall c_1 < c_2 : l[c_1] = l[c_2] \implies [assign[I] = c_1] \leq_{lex} [assign[I] = c_2] \tag{14}$$

The second constraint considers interchangeable also all those couriers who haven't filled their capacity (still respecting their maximum load):

$$\begin{split} \forall c_1 < c_2 : \max\{a\_l[c_1], a\_l[c_2]\} \leq \min\{l[c_1], l[c_2]\} \implies \\ [assign[I] = c_1] \leq_{lex} [assign[I] = c_2] \end{split} \tag{15}$$

This requires also the computation of the actual load a_l with the following constraint:

$$\forall c : a\_l[c] = \sum_{\substack{p: \\ packages[c,p]=1}} w[p] \tag{16}$$

2.4 Objective function

The goal of the problem is to minimize the objective function, i.e the maximum distance travelled by any of the couriers:

$$\min \max_{c} total\_distance[c] \tag{17}$$

2.5 Validation

2.5.1 Combining the strategies

In order to test and validate our model we have built different configurations that combine the different strategies presented in the previous sections. We identify the following keywords:

- **plain**: basic model without symmetry breaking constraints. Here, both global (11) and non-global (12) constraints for the capacities have been separately tested[1].
- **sym**: plain model using both symmetry constraints (14) and (15).
- **sym_simple**: plain model using only the simple symmetry constraint defined in (14).
- **lnsN**: plain model implementing *Large Neighborhood Search*. **Sequential search** on (*assign*, *path*) with *dom_w_deg* and *indomain_random*; **relax and reconstruct** on *assign* with N% of reused values upon restart; **luby restart** with *scale* = 100 to avoid getting stuck in deep regions of the search tree.

[1]The results are presented only for hard instances, in which we obtained interesting results.

The models tested are a configuration of one or more of those keywords, combining together their specific features. Moreover, every model have been tested using *gecode* and *chuffed* solvers[2].

2.5.2 Experimental results

Soft instaces Results of the soft instances are shown in Table 3.

- **Lower bounds constraints:** In almost every instance, optimal results have been proven by the lower bound constraint which was able to cut the search space after reaching it, especially in those models where symmetry breaking was not handled.
- **Symmetry breaking constraints:** Turned out to be the key technique to effectively reduce the search space of the solution by quickly lowering the optimal value towards the bound set by the lower bound constraint: models handling symmetry returned the best result, considering the average time to solve an instance.
- **Large Neighborhood Search:** When dealing with relatively small spaces of solution LNS is not the best idea in terms of proving optimality: some of the instances returned an optimal value but without halting in the time limit.
- **Solvers:** In most of the instances, *chuffed* solver was able to effectively reduce the search space even with models that weren't exploiting bound and symmetry constraints, while *gecode* couldn't. Nevertheless, the latter is preferable since most of the search space annotation were not supported in *chuffed*, as well as the LNS.
- **Load constraints:** We haven't noticed drastic improvements nor differences in expressing the load constraint with global rather than custom.

	Gecode					Chuffed		
Inst.	*plain*	*sym*	*sym simple*	*lns*90	*lns*90 *sym*	*plain*	*sym*	*sym simple*
1	**14**	**14**	**14**	14	14	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**	**226**	**226**	**226**	**226**
3	**12**	**12**	**12**	12	12	**12**	**12**	**12**
4	**220**	**220**	**220**	**220**	**220**	**220**	**220**	**220**
5	**206**	**206**	**206**	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**	**322**	**322**	**322**	**322**
7	**167**	**167**	**167**	**167**	**167**	**167**	**167**	399
8	**186**	**186**	**186**	**186**	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**	**244**	**244**	**244**	**244**

Table 3: Results on soft instances. In bold, optimal results.

Hard instaces Results of the hard instances are presented in Table 4.

- **Symmetry breaking constraints:** For solving hard instances they mainly added overhead to the solving process, by halting at the time limit with no solution in more than 80% of the instances. A simpler symmetry breaking constraint consisting in only (14) helped computing the solution but with no improvements with respect to other strategies.
- **Large Neighborhood Search:** In hard instances LNS have been the key to explore better the search space and returning optimal solution for some of the instances.
- **Solvers:** *chuffed* was not able to return solutions on hard instances, therefore only *gecode* results have been presented.

[2] *Chuffed* is not compatible with LNS and couldn't find a solution for any of the hard instances.

- **Load constraints:** When dealing with hard instances the global constraint (11) achieved way worse results in average, with a very interesting result: instance 17 has been solved with the best objective value by our **plain** model with custom constraint (12).

Inst.	Gecode									
	plain (11)	*plain* (12)	*sym*	*sym simple*	*lns*90	*lns*94	*lns*90 *sym*	*lns*94 *sym*	*lns*90 *sym simple*	*lns*94 *sym simple*
11	1093	1131	N/A	N/A	530	505	N/A	N/A	519	575
12	N/A	1072	N/A	N/A	368	**346**	N/A	N/A	368	385
13	1070	1214	1204	1006	636	682	678	668	666	598
14	2319	1513	N/A	2125	783	806	N/A	N/A	773	786
15	1354	1182	N/A	1617	772	773	925	871	754	793
16	N/A	578	N/A	N/A	**286**	**286**	N/A	N/A	**286**	**286**
17	3048	1105	2987	2963	1151	1212	N/A	N/A	1155	1214
18	1462	1206	N/A	2003	697	719	N/A	N/A	665	727
19	N/A	1248	N/A	1717	**334**	**334**	N/A	N/A	336	335
20	4360	1365	N/A	3665	1059	1091	N/A	N/A	1118	1142
21	N/A	1763	N/A	1712	607	602	N/A	N/A	591	587

Table 4: Results on hard instances. In bold, optimal results.

3 SMT Model

For the second approach, using SMT, we adopted a top-down methodology. To solve the task we were given, we began by adding all the constraints that initially seemed logical and made sense in the context of the problem. As we analyzed the model further, we identified and removed constraints that were redundant or unnecessary, improving its overall efficiency and performance.

3.1 Decision variables

The following decision variables have been used:

- $x_{c,p}$; *Domain*: Boolean, for $c \in C$ and $p \in I$; *Semantics*: Represents whether courier c is assigned to item p. The value of $x_{c,p}$ is 'True' if courier c is assigned to item p, and 'False' otherwise.

- $y_{c,i,j}$; *Domain*: Boolean, for $c \in C$ and $i,\ j \in N$; *Semantics*: Representing whether courier c travels from location i to location j. The value of $y_{c,i,j}$ is 'True' if the courier travels directly from location i to j, and 'False' otherwise.

- $order_{c,p}$; *Domain*: Integer ($\mathbb{Z}$), for $c \in C$ and $p \in I$; *Semantics*: Represents the position of item p in the delivery sequence of courier c.

$$order_{c,p} = \begin{cases} > 0 & \text{if } x_{c,p} = \text{True} \\ < 0 & \text{if } x_{c,p} = \text{False} \end{cases}$$

  This enables efficient calculation of the total travel cost and facilitates the reconstruction of the delivery route based on the determined order.

- d_c: *Domain*: List of integers ($\mathbb{Z}$) of size m; *Semantics*: Represents the total distance traveled by courier c on their assigned route, including travel from the depot to the items and back.

- $max\_distance$; *Domain*: Integer ($\mathbb{Z}$); *Semantics*: Represents the maximum distance traveled by any courier. The objective is to minimize this value.

3.1.1 Auxiliary variables

Item sizes and load capacities are encoded in Z3 as decision variables using integer lists, while the distance matrix is represented as a 2D list of integers.

To improve solver performance, we tried different sorting strategies and ultimately sorted the items in ascending order of size, adjusted the distance matrix accordingly, and sorted couriers by their load capacities. This approach achieved efficient results for the first ten instances by organizing the problem to align with Z3's depth-first search, guided by Conflict-Driven Clause Learning (CDCL). Sorting also reduces symmetry, preventing redundant exploration of equivalent solutions and enhancing efficiency.

3.2 Objective function

As given in the problem description, the goal is to minimize the maximum distance traveled among all couriers. In our SMT model implementation is minimizing the variable *max_distance*

$$\min(\text{max\_distance}) = \min\left(\max\left(\{d_0, d_1, \ldots, d_{\text{m}-1}\}\right)\right)$$

, with *max_distance* and d_c as explained in section 3.1

3.3 Constraints

- **Demand Fulfillment**

$$\sum_{c=0}^{\text{m}-1} \mathbf{1}(x_{c,p} = \text{True}) = 1 \quad \forall p \in I$$

  This constraint ensures each item must be assigned to exactly one courier.

- **Capacity Constraint**

$$\sum_{p=0}^{\text{n}-1} \mathbf{1}(x_{c,p} = \text{True}) * \text{w}_p \leq \text{l}_c, \quad \forall c \in C$$

  This ensures that each courier's total load must not exceed their capacity.

- **Early Exclusion of Undeliverable Items**

$$\neg x_{c,p}, \quad \forall c \in C, \forall p \text{ such that } \text{w}_p > \text{l}_c$$

  If an item cannot fit in a courier's load (i.e., item size exceeds the courier's capacity), then that courier cannot be assigned that item. This helps reduce the search space by eliminating impossible assignments.

- **At Least One Item Per Courier**

$$\sum_{p=0}^{\text{n}-1} \mathbf{1}(x_{c,p} = \text{True}) \geq 1, \quad \forall c \in C$$

  This constraint ensures that every courier is assigned at least one item. This has to hold since in any solution where a courier is left empty, we can always improve the solution by transferring one of the items assigned to another courier to the empty courier because of triangle inequality.

- **No Loop Connections**

$$\forall c \in C, \forall i \in N \quad y_{c,i,i} = False$$

  This constraint prevents a courier from staying at the same location.

- **Direction of Route**

$$\forall c \in C,\ \forall p, q \in I, (y_{c,q,p}) \Rightarrow \neg\,(y_{c,p,q})$$

  This prevents a courier from traversing the same pair of items in the opposite direction, ensuring a valid route without inconsistent transitions.

- **Variable Link: Assignments and Route**

$$\sum_{p=0}^{n-1} \mathbf{1}(y_{c,p,i} = \text{True}) = \mathbf{1}(x_{c,i} = \text{True}), \quad \forall c \in C, \forall i \in N,$$
$$\sum_{p=0}^{n-1} \mathbf{1}(y_{c,i,p} = \text{True}) = \mathbf{1}(x_{c,i} = \text{True}), \quad \forall c \in C, \forall i \in N.$$

$$\sum_{p=0}^{n-1} \mathbf{1}(y_{c,p,depot} = \text{True}) = 1, \quad \sum_{p=0}^{n-1} \mathbf{1}(y_{c,depot,p} = \text{True}) = 1, \quad \forall c \in C$$

This constraint establishes the relationship between the boolean variables y and x and assures consistency. It ensures that if an item p is assigned to a courier c (i.e., $x_{c,p} = True$), then the corresponding indicator variable $y_{c,q,p}$ must be activated (i.e., $y_{c,q,p} = True$) for some q.

- **Variable Link: Assignments and Order**

$$\forall c \in C, \forall p \in I : \qquad (x_{c,p} = False \implies \text{order}_{c,p} < 0) \quad \wedge (x_{c,p} = True \implies \text{order}_{c,p} > 0)$$

This constraint establishes the relationship between the boolean variable x and the integer variable *order*, enforcing positive values for *order* when $x_{c,p} = True$, and negative when $x_{c,p} = False$, as explained in the description of the variable *order*.

- **Variable Link: Route and Order**

$$\forall c \in C, \forall p, q \in I : \quad (y_{c,p,q} = True \implies \text{order}_{c,p} < \text{order}_{c,q})$$

This enforces a logical sequencing where items appear in the correct order along the route, respecting the transitions between locations.

- **Uniqueness of Order Variable**

$$\forall c \in C : \quad \text{Distinct}\left([\text{order}_{c,p} \mid p \in I]\right)$$

This guarantees that no two items assigned to the same courier share the same position in the delivery route, maintaining the integrity of the route's sequencing.

- **Preventing Redundant Transitions for Couriers with Multiple Items**

$$\forall c \in C, \forall p \in I : \qquad \left(\sum_{q=0}^{\text{n}-1} \mathbf{1}(x_{c,q} = \text{True}) > 1 \implies \qquad \neg\left(y_{c,depot,p} \wedge y_{c,p,depot}\right)\right)$$

If a courier is assigned multiple items, this constraint ensures at least one additional item is delivered before returning to the depot, reducing inefficient backtracking. While optional, it improved optimization speed in the first 10 instances by reducing the search space and eliminating redundant paths.

- **Distance Per Courier Calculation**

$$d_c = \sum_{i,j}^{\text{n}} \mathbf{1}(y_{c,i,j} = \text{True}) * D_{i,j} \quad \forall c \in C$$

This formula calculates the total distance traveled by any courier c in C by summing up the distances between locations that the courier travels using the variable y. The distances are taken from the distance matrix D.

3.3.1 Symmetry Breaking Constraints

To reduce symmetry and improve solver performance, the model includes the following:

- **Order of Couriers by Distance**

$$d_c \leq d_{c+1} \quad \forall c \in \{0, 1, \ldots, \mathrm{m}-2\}$$

  The distances traveled by couriers are sorted in ascending order to break symmetry. This ensures that the solver does not waste time exploring equivalent solutions that differ only in the assignment of couriers

Inst.	SMT
1	**14**
2	**226**
3	**12**
4	**220**
5	**206**
6	**322**
7	**167**
8	**186**
9	**436**
10	**244**
11-21	N/A

Table 5: SMT Performance on all instances.

3.4 Validation

The SMT model detailed in this section was implemented using the Z3 library in Python. This implementation applies theories in Satisfiability Modulo Theories (SMT) such as linear arithmetic, array theory, and constraints to model the problem. For instances 1 through 10, the model efficiently computes solutions to optimality within a short period. However, for instances 11 through 21, the execution encounters a termination due to a 300-second pre-defined timeout, without discovering any solution. Table 5.

4 MIP

The third approach we used to solve the problem is MIP. The MIP model for the problem mainly relies on the decision variable $x_{i,j,c}$,a boolean variable which contains every decision for every courier. To make the model more efficient we also integrated the Miller-Tucker-Zemlin (MTZ) formulation to solve sub-routes: this approach provide an elegant constraint which makes the solution more efficient.

4.1 Decision variable

To function, the model uses the following decision variable:

- $x_{i,j,c}$; *Domain*: Boolean, for $i, j \in N$ and $c \in C$; *Semantics*: Represents whether the path from i to j is part of the solution for courier c. Thus $x_{i,j,c}$ is *True* if c goes trough $[i, j]$, *False* otherwise.

4.1.1 Auxiliary variables

- $u_{i,c}$ *Domain*: Integer, for $i \in N$, $c \in C$; *Semantics*: Auxiliary variable introduced to apply the Miller-Tucker-Zemlin (MTZ) formulation to solve sub-routes AIMMS 2020.

- *max_route_distance*: *Domain*: Integer ($\mathbb{Z}$); *Semantics*: this variable contains an integer which represents the distance travelled by each courier. Its value is bounded by the same bounds described in subsection 1.4.

- *courier_weights$_c$*: *Domain*: Integer, $c \in C$; *Semantics*: keeps track, for each courier, of the weight that it is carrying trough the route.
- *courier_distance$_c$*: *Domain*: Integer, $c \in C$; *Semantics*:keeps track, for each courier, of the amount of road travelled.

4.2 Constraints

- **Weight Constraints for Each Courier**:
$$\sum_{i=0}^{n}\sum_{j=1}^{n} x_{ijc} \cdot w_j = \text{courier\_weights}_c \quad \forall c \in C$$
Ensures the total weight carried by each courier c matches the specified capacity.
- **Prevent Self-Looping Arcs**:
$$\sum_{i=0}^{n}\sum_{c=0}^{m-1} x_{iic} = 0$$
Ensures no courier travels from a node to itself.
- **Each Node Visited Exactly Once**:
$$\sum_{i=0}^{n}\sum_{c=0}^{m-1} x_{ijc} = 1 \quad \forall j \in \{1, \dots, n\}$$
Ensures each node is visited exactly once by exactly one courier.
- **Each Courier Departs from the Depot**:
$$\sum_{j=1}^{n} x_{0jc} = 1 \quad \forall c \in C$$
Ensures each courier c starts its route from the depot.
- **Each Courier Returns to the Depot**:
$$\sum_{i=1}^{n} x_{i0c} = 1 \quad \forall c \in C$$
Ensures each courier C ends its route at the depot.
- **Path Connectivity Constraints**:
$$\sum_{i=0}^{n} x_{ijc} = \sum_{i=0}^{n} x_{jic} \quad \forall j \in \{1, \dots, n\}, \forall c \in C$$
Ensures that if a courier enters a city, it must also leave that city.
- **Ensure No Double Usage of Arcs**:
$$x_{ijc} + x_{jic} \leq 1 \quad \forall i, j \in \{1, \dots, n\}, \forall c \in C$$
Ensures no courier travels both from i to j and from j to i.
- **Subtour Elimination Constraints**:
$$u_{ic} - u_{jc} + n \cdot x_{ijc} \leq n - 1 \quad \forall i, j \in \{1, \dots, n\}, i \neq j, \forall c \in C$$
This constraint follows the MTZ formulation: it ensures that no subtours ($A \to B \to A$) are formed by enforcing a logical order of visits.
- **Maximum Distance Constraint**:
$$\text{max\_route\_distance} \geq \text{courier\_dist}_c \quad \forall c \in C$$
Ensures no courier's route exceeds the maximum allowed distance.

4.3 Objective function

The objective function is defined as follows:

$$\min \max_{c} \sum_{i,j=1}^{n+1} x_{i,j,c} \cdot D_{i,j}$$

4.4 Validation

The MIP model was implemented in Python, relying on the usage of the Pulp library. To solve the problem we used the CBC solver,with a timeout of 300 seconds, which offered good performances on the first 10 instances, and provided a solution also to instance 13 and 16.

4.4.1 Results

The model performed well on the first 10 instances, providing optimal solutions to 9 out of the first 10 instances, with instance 7 being the only non optimal one. The following 11 instances proved to be more complex, but the model managed sub-optimal solutions for instance 13 and 16.

Inst.	CBC
1	**14**
2	**226**
3	**12**
4	**220**
5	**206**
6	**322**
7	169
8	**186**
9	**436**
10	**244**
11-12	N/A
13	728
14-15	N/A
16	322
17-21	N/A

Table 6: MIP Performance on all instances.

5 Conclusion

For the scope of this project, we explored different approaches to solving the Multiple Couriers Problem, each with its own modeling ideas. Working as a team, it was interesting to see how differently we tackled the problem and how our individual perspectives shaped the solutions.

All of the approaches produced almost equally optimal results for the first 10 instances, with only instance 7 from the MIP solution being suboptimal. However, as the problem size increased with more couriers and more items, the search space grew significantly, demanding more computational power and better strategies to navigate the complexity. In these larger instances, the CP approach delivered by far the best results, successfully solving all cases by effectively exploring the search space through Large Neighborhood Search. In contrast, the SMT approach was unable to find solutions for these instances, while the MIP approach solved only 2 out of the remaining 11, both with suboptimal results.

Thus, we can conclude that the CP approach is the most effective for solving the MCP problem. Beyond the results, this project was a valuable learning experience. It was our first encounter with this type of problem, and working through it gave us an engaging introduction to tackling NP-hard challenges in optimization.

References

[AIM20] AIMMS. *Miller-Tucker-Zemlin Formulation*. 2020. URL: https://how-to.aimms.com/Articles/332/332-Miller-Tucker-Zemlin-formulation.html.