Multiple Couriers Planning Problem

Francesco Pivi francesco.pivi@studio.unibo.it
Matteo Fusconi matteo.fusconi4@studio.unibo.it
Shola Oshodi shola.oshodi@studio.unibo.it
Niccolò Marzi niccolo.marzi2@studio.unibo.it

December 2023

1 Introduction

The following report contains the strategies used to solve the multiple courier planning problem alongside the results obtained. Different optimization techniques and different models have been adopted and they will be discussed in the next sections.

Our project extended over about four months, during which team members collaborated on problem modeling. Together, we delved into various models encompassing a spectrum of optimization techniques, frequently designating specific members to implement each approach. The main challenges we faced were predominantly linked to solving intricate instances through Python-based interfaces (using Pulp and Z3). Memory consumption emerged as a notable concern, especially as the problem's scale—specifically, the number of items—increased.

In the next paragraphs we present the main commonalities between the models, namely the preprocessing steps.

1.1 Variables

The input variables are the number of couriers m, the number of items n, the array of maximum load capacity of each courier l, the array of item size s and the matrix of distances D, whose elements satisfy the triangle inequality relation:

$$\|D_i - D_j\| \leq \|D_i - D_k\| + \|D_k - D_j\|$$

In the pre-processing step, made right after data loading, we sort the couriers basing on their respective load sizes in descending order.

The model variables differ based on the approach used, and on its specific implementation, and will be explained on detail in the next sections.

1.2 Structure

We observed that if the minimum capacity of each courier ($\min l$) is greater than or equal to the weight of the lightest item ($\min s$), it implies that every courier has the potential to be beneficial in transporting at least one item. Given this hypothesis, and recognizing that the triangle inequality must hold for the distance matrix between each item, initiating more couriers will inevitably lead to a reduction in the objective function. In this context, the objective function is defined, for each solving approach, as the maximum distance traveled by any courier. For this reason, in each model we have implemented, we demanded the departure of each courier.

1.3 Bounds

From an implementation perspective, it is essential to establish lower and upper bounds for the variables within the models. The following considerations are in common between each model, and are based on considerations of the problem's structure.

- **Distance bounds:**
  Since we made the assumption that all couriers must start, the minimum distance travelled by any of them won't be 0, but we have to consider a path where a courier carries just one item, namely a path in the form DEPOSIT -> ITEM -> DEPOSIT. Thus, the minimum distance that any courier can travel will be the minimum of all possible paths of this kind. In addition, since the triangle inequality must hold, and we know that carrying more items will lead to travel more, we can also infer that the minimum value for the objective function ρ that we know a-priori will be the longest path of this kind. The resulting formulas are:

$$dist\_lb = \min_i (D_{n+1,i} + D_{i,n+1})$$

$$\rho\_lb = \max_i (D_{n+1,i} + D_{i,n+1})$$

  Regarding the upper bound, we have considered the scenario where a single courier transports all the items. Specifically, we have taken into account the distance covered in delivering the packages in numerical order from package 1 to package n. This works both for the couriers' distance and objective upper bounds. The resulting formula is as follows:

$$dist\_ub = \rho\_ub = D_{n+1,1} + \sum_{i=1}^{n} D_{i,i+1}$$

  Note that in this formula we consider also the "return to deposit" distance, namely when $i = n$ we have $D_{n,n+1}$.

- **Load bounds:**
  The minimum load size each courier can carry (min_load) is simply the minimum item size: $\min(s)$, whereas the maximum one (max_load) is the maximum of load capacities: $\max(l)$.

1.4 Hardware

We ran all the experiments presented below on a computer with the following specifications:

- RAM: 8 GB
- Processor: Intel(R) Core(TM) i5-8750H CPU @ 1.80GHz
- Clock Speed: 1.99 GHz
- Cores: 4
- Logical Processors: 8

2 CP model

This section is dedicated to the constraint programming (CP) technique. Different models were built using the MiniZinc language and then tested with two different solvers. We discuss now the model that performed better respect to the others. It takes inspiration from the work on Minizinc benchmarks repository [1], making the necessary adjustments to fit our problem.

2.1 Decision variables

In order to make the problem formulation more fluent and comprehensible, we firstly define the following sets:

- **Items**: goes from (1) to (n) and represent the position of every item that we have to deploy.
- **StartNodes**: goes from ($n + 1$) to ($n + m$) and it represent the start for each courier.
- **StartNodesPred**: ($n + 2$) to ($n + m$) it is the range of the start nodes except the first one.
- **EndNodes**: goes from ($n + m + 1$) to ($n + 2 \cdot m$) and it represents the end of the travel of all the other couriers.

Now we can define the model that is based on the following variables. All of them are one dimensional arrays of size $n + 2 \cdot m$.

[1] Available at: `https://github.com/MiniZinc/minizinc-benchmarks/blob/master/cvrp/cvrp.mzn`

- **Successor**: Depicts the selected route for each courier, with each position i indicating the subsequent destination reached by the courier.
- **Predecessor**: Signifies the same concept as the vector above, but at each index i, it denotes the previously visited destination.
- **Courier Route**: At each position i, it holds the index of the courier that visited that particular place.
- **Incremental_Dist**: The distance traveled by each courier is determined by incrementally adding the distances between the packages they are tasked to carry.

2.2 Constraints

We are now listing the constraints employed in our model, classified into four types: Path, Vehicle, Load, Distance and symmetry constraints.
We explore the first ones.

- Firstly we need to give $Successor$ and $Predecessor$ the structure of a circuit. It means that each element of the vector has, as its value, the index of another element of that vector without repetitions and without creating sub-cycles. This can be easily implemented via the following global constraints:
  1. $circuit(Successor)$.
  2. $circuit(Predecessor)$.
- Secondly we want to connect the end of the i^{th} with the start of the $(i+1)^{th}$. To do so we developed the following constraints:
  1. $\forall i \in EndNodes, \quad Successor[i] = i - m + 1$.
  2. $\forall i \in StartNodesPred, \quad Predecessor[i] = i + m - 1$.
- Then we have to connect the end of the path the last courier with the start of the first one and vice versa.
  1. $Successor[n + 2 \cdot m] = m + 1$.
  2. $Predecessor[n + 1] = n + 2 \cdot m$.
- Now we force each courier to star.
  1. $\forall i \in StartNodes, \quad Successor[i] = Successor[i + m]$.
  2. $\forall i \in EndNodes, \quad Predecessor[i] = Predecessor[i - m]$.

Now, let's consider the constraint involving the vehicle variable.

- Firstly we have to state that each courier will have an unique starting and ending node.

1. $\forall i \in StartNodes, \quad CourierRoute[i] = i - n.$
2. $\forall i \in EndNodes, \quad CourierRoute[i] = i - n - m.$

- Secondly a courier cannot change until it's arrived at the next $StartNode$.

1. $\forall i \in Items, \quad CourierRoute[Successor[i]] = CourierRoute[i].$
2. $\forall i \in Items, \quad CourierRoute[Predecessor[i]] = CourierRoute[i].$

Regarding the load constraint we used a single global constraint.

- This constraint mandates that each item, with weight $item\_size[i]$, is assigned to the courier $CourierRoute[i]$ in a way that the sum of weights of items assigned to the given courier does not exceed his capacity.

  $$bin\_packing\_capa(courier\_capacity, CourierRoute[Items], item\_size)$$

  where we considered only the variables in $CourierRoute$ that belong to the set $Items$.

Finally we are able to calculate the distances for each courier in our model.

- We choose to define them cumulatively. At first we have to set the distance for each of our courier to zero. Then we add the distance from the depot to the first visited place. Finally we sum all the remaining distances in the courier path.

1. $Dist\_C[i] = 0, \quad \forall i \in StartNodes.$
2. $Dist\_C[Successor[i]] = distances[n{+}1, Successor[i]] \forall i \in StartNodes,.$
3. $Dist\_C[Successor[i]] = Dist\_C[i]{+}distances[i, min(Successor[i], n{+}1)] \ \forall i \in Items$

Finally we developed a single symmetry breaking constraint.

- From the observation that if the distance matrix is symmetric, each path will be equal to its inverse, so the number of solutions actually halves. To break such symmetry, we impose that the first package carried is smaller than or equal to the last one.

  $$\forall i \in StartNodes \quad Successor[i] \leq Predecessor[i]$$

After that we take the maximum of the distance of each courier and we try to minimize it.

2.3 Validation

Experimental design

The CP model has been tested using the **Gecode** and the **Chuffed** solvers. We tested the solvers separately with and without symmetry breaking constraints. Regarding the Gecode solver, we were able to include the option of some search heuristics. We applied the "relax and reconstruct" heuristic to the Vehicle Route variable, conducting a neighborhood search with a 90% probability of fixing the variable to its previous solution at each restart. Moreover, since the integer search strategy which resulted more effective was first fail with a split domain hierarchy, we used that one.

Experimental results

In Table 1 there are the results obtained using the two solvers with and without symmetry breaking.

Instances	Gcd	Gcd+sym	Gcd+heu	Gcd+heu+sym	Cfd	Cfd+sym
1	**14**	**14**	**14**	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**	**226**	**226**
3	**12**	**12**	**12**	**12**	**12**	**12**
4	**220**	**220**	**220**	**220**	**220**	**220**
5	**206**	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**	**322**	**322**
7	**167**	**167**	**167**	**167**	**167**	**167**
8	**186**	**186**	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**	**244**	**244**
11	719	726	760	727	1109	1115
12	364	381	381	381	594	593
13	1030	1062	1076	1076	1806	1806
14	1075	1070	1083	1060	1762	1762
15	1232	1232	1029	1075	1471	1471
16	**286**	**286**	**286**	**286**	**286**	**286**
17	1439	1422	1344	1481	1571	1571
18	882	876	827	827	1561	1561
19	390	390	390	390	**334**	**334**
20	1421	1421	1504	1504	1529	1529
21	892	892	892	892	1474	1409

Table 1: CP Model distances results using Gecode (Gcd) and Chuffed (Cfd)

Notably, we managed to obtain a result for every instance in each of the methods explained above. We obtained optimal results for all the "easy instances" (those up to the tenth), for the 16-th and for the 19-th with Chuffed

solver. We can observe that sometimes, the search heuristics help in finding better results, while some other times it does not.

2.4 Other experiments

As previously mentioned, we developed alternative models. The first model closely resembles the MIP implementation, while the second mirrors the SMT approach. However, these alternatives yielded inferior performance compared to the described model, primarily because they entail a significantly larger number of decision variables.

3 SAT model

This section examines the model implemented to solve the Multiple Couriers Planning Problem through the Boolean Satisfiability Theory (B-SAT). The particularity of this theory is that it is designed to use only boolean variables, and to handle only constraints encoded in Propositional Logic.

3.1 Decision variables

The structure of the model makes use of the following variables:

- **X :** a 3D matrix of the form $(m \times n \times (n+1))$. The element X_{ijk} is True if the courier $i+1$ carries item k. The second index is used to handle the order of package delivery. If $k = 0$, i.e. $X_{i,j,0}$ means that courier $i+1$ does not carry any item in position $j+1$.

  E.g. $X_{0,1,3}$ means that courier 1 delivers item 3 as its second package in his route.

$$X_{i,j,k} \in \{0,1\} \qquad \forall k \in \{0,...,n\}, \forall j \in \{0,...,n-1\}, \forall i \in \{0,...,m-1\}$$

- **W_par (partial weights):** a $(m \times n)$ array of binary encoded numbers. In particular, $W\_par_{i,k}$ contains the binary encoding of item k, if it is carried by courier $i+1$, 0 otherwise.
- **W_tot (total weights):** a vector of length m of binary encoded numbers. It stores the total weight carried by each courier.
- **D_par (partial distances) and D_tot (total distances):** similarly w.r.t. W_tot and D_par, but for the couriers' travelled distances, respectively with shape $(m \times (n+1))$ and m.
- **ρ (objective function)**: The maximum distance travelled among all the couriers.

3.2 Constraints

The constraints necessary for the implementation of the model are the following:

- Ensure that each courier brings exactly one item at each allowed position or it does not carry anything at that place:

$$\bigwedge_{i=0}^{m-1} \bigwedge_{j=0}^{n-1} \text{exactly\_one}(X_{i,j,k} \mid k \in 0, \ldots, n)$$

- Each item is carried by one and only one courier:

$$\bigwedge_{k=1}^{n} \text{exactly\_one}(X_{i,j,k} \mid i \in 0, \ldots, m-1, j \in 0, \ldots, n-1)$$

- If any courier doesn't deliver any package at position j in the path, also at position $(j+1)$ he doesn't:

$$\bigwedge_{i=0}^{m-1} \bigwedge_{j=0}^{n-2} (X_{i,j,0} \Rightarrow X_{i,j+1,0})$$

- Additional constraints are present to ensure that *W_par*, *W_tot*, *D_par* and *D_tot* are filled according to the rules defined above in the section (3.1).

- Each courier can't exceed his load size limit:

$$\text{lesseq}(W\_tot_i, l_i) \quad \forall i \in \{0, \ldots, m-1\}$$

In addition to the model formulation constraints, we also developed other two symmetry breaking constraints (1 and 2), and a heuristic constraint (3).

1. If two couriers have the same load capacity, there is no possibility to distinguish them; this means that their paths could always be swapped. To break such symmetry, we impose a lexicographic ordering between them.

$$\forall i \in \{0, \ldots, m-2\} \quad (l_i = l_{i+1}) \Rightarrow \text{ohe\_less}(X_{i,0}, X_{i+1,0})$$

   where ohe_less is an implementation of the less operator between one-hot encoded numbers.

2. else if the couriers don't share the same load capacity, it can happen that their path can be swapped, if their capacity is high enough. To disambiguate them, we force the first couriers (those whose capacity is higher) to carry a higher weight.

$$\forall i \in \{0, \ldots, m-2\} \quad lesseq(W\_tot_{i+1}, W\_tot_i)$$

3. We propose the following heuristic constraint based on the fact that carrying more packages correlates with carrying more weight. We impose that the couriers with the largest load capacity must carry the highest number of packages.

$$\bigwedge_{i=0}^{m-2} \bigwedge_{j=0}^{n} X_{i,j,0} \Rightarrow X_{i+1,j,0}$$

If all the packages had the same size, this would act as a symmetry breaking constraint and we would be guaranteed that the search would yield to the optimal solution. Although we know that this assumption is not true in our case, this constraints actually helped a lot in the results without compromising optimality.

3.3 Search strategies

As the Z3 solver lacks native optimization capabilities and functions solely as a checker, we developed a custom optimizer from scratch. This was accomplished through two distinct strategies:

- **linear search**: iteratively impose a stricter lower bound for the objective function until the problem is deemed unsatisfiable. The optimal distance is considered as the one obtained in the previous step.
- **binary search**: improves time complexity by iteratively halving the search space, dynamically updating upper and lower bounds. If the difference surpasses 1, the upper bound decreases until a satisfiable solution is found. In case of unsatisfiability, it reverts to the previous state and increases the lower bound. The solver explores solutions within this refined interval. The search stops when the upper and lower limits coincide.

3.4 Validation

Experimental design and results

The SAT model has been implemented using the Z3 python library.
Table 2 displays the results obtained by our SAT model using both linear and binary search with and without symmetry breaking, and with heuristics (without symmetry breaking).

We managed to obtain an optimal solution to at least one strategy for each instance. From these results we can see that both symmetry breaking and heuristic constraints are very useful and not only they speed up the search, but often they are fundamental to get in time to the optimal solution. The heuristic constraint did not compromise finding the optimal result.

Instance	LS	LS + SB	LS + HEU	BS	BS + SB	BS + HEU
1	**14**	**14**	**14**	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**	**226**	**226**
3	**12**	**12**	**12**	**12**	**12**	**12**
4	**220**	220	**220**	**220**	**220**	**220**
5	**206**	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**	**322**	**322**
7	207	181	**167**	191	**167**	**167**
8	**186**	**186**	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**	**244**	**244**

Table 2: Results obtained using SAT with the strategies defined above. LS and BS stand for linear and binary search, SB and HEU are symmetry breaking and heuristics. Some instances were not reported because they failed to encode, or neither strategy yielded any solution.

4 SMT model

This section is dedicated to the Satisfiability Modulo Theories (SMT), an approach which attempts to solve optimization problems by merging solvers of different theories [1]. Two models were built exploiting the Linear Integer Theory and the Vector Array Theory.

4.1 Decision variables

The structure of the main model makes use of the following variables:

- **x:** vector of length m comprising Z3 Arrays, each containing integer variables sized $n+1$. These variables represent the travel schedules for individual couriers. The domain of each variable is $\{1, \ldots, n+1\}$, indicating the distribution centers visited by the courier. If $x_{i,j} = k$, it signifies that the i^{th} courier is at the k^{th} distribution center during the j^{th} time step. Notably, when $k = n+1$, it denotes the courier's return to the depot.
- **couriers_loads**: list of integers that contain the total weight carried by each courier.
- **m_dist**: list of integers that contain the distance travelled by each courier.
- **ρ (objective function):** an integer variable encoding the maximum distance travelled among the couriers.

4.2 Constraints

The constraints necessary for the implementation of the model are the following:

- Once a courier gets to the deposit, then he stays there for all the subsequent time steps:

$$\bigwedge_{i=0}^{m-1} \bigwedge_{j=0}^{n-1} \left(x_{i,j} = n+1 \Rightarrow \left(\bigwedge_{k=j+1}^{n} x_{i,k} = n+1 \right) \right)$$

- Each distribution point must be visited exactly once and by only one courier:

$$\bigwedge_{k=1}^{n} \text{exactly\_one}(x_{i,j} = k \mid i \in \{0, \ldots, m-1\}, j \in \{0, \ldots, n\})$$

- Compute *couriers_loads* for each courier as the sum of the size of those items which he delivers:

$$\bigwedge_{i=0}^{m-1} \left(couriers\_loads_i = \sum_{j=0}^{n} \sum_{k=1}^{n} \text{ite}(x_{i,j} == k, s_{k-1}, 0) \right)$$

  where "ite" (if-then-else) is utilized with three arguments: the condition, an expression when the condition is true, and an expression otherwise.

- Ensure that each courier respects its maximum load size:

$$\bigwedge_{i=0}^{m-1} couriers\_loads_i \leq l_i$$

- Finally we compute the distance travelled by every courier (m_dist). From them we extract the maximum distance ρ and we try to minimize it.

- Similarly to the model formulation used for the SAT approach, we add two symmetry breaking constraint to disambiguate couriers with the same or different load capacities, and a heuristic constraint that impose couriers with the most capacity to carry the largest amount of items:

  1. $\bigwedge_{i=0}^{m-2} \quad (l_i = l_{i+1} \Rightarrow x_{i,0} \leq x_{i+1,0})$
  2. $\bigwedge_{i=0}^{m-2} \quad (l_i \neq l_{i+1} \Rightarrow couriers\_loads_{i+1} \leq couriers\_loads_i)$
  3. $\bigwedge_{i=0}^{m-2} \bigwedge_{j=0}^{n} ((x_{i,j} = n+1) \Rightarrow (x_{i+1,j} = n+1))$

4.3 Validation

Experimental design

The SMT model has been implemented using the Z3 Python library. Additionally, we introduced a SMTlib version to enable the execution of code across different solvers. This implementation allowed us to test the CVC4 solver using our pre-existing model. To simplify the experimentation process, we employed bash files to iteratively add SMTLib's constraints to support binary and linear search within the SMTlib environment.

Experimental results

	Z3						CVC4	
Inst	LS	LS+SB	LS+HEU	BS	BS+SB	BS+HEU	LS	BS
1	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**	**226**	**226**	N/A	N/A
3	**12**	**12**	**12**	**12**	**12**	**12**	N/A	**12**
4	**220**	**220**	**220**	**220**	**220**	**220**	N/A	N/A
5	**206**	**206**	**206**	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**	**322**	**322**	**322**	**322**
7	302	278	288	316	291	239	N/A	N/A
8	**186**	**186**	**186**	**186**	**186**	**186**	N/A	N/A
9	**436**	**436**	**436**	**436**	**436**	**436**	N/A	N/A
10	**244**	**244**	**244**	**244**	**244**	**244**	N/A	N/A

Table 3: SMT results with Z3 solver and CVC4 solver. LS and BS stand for linear and binary search, SB and HEU are symmetry breaking and heuristics.

As we can see in Table 3, the Z3 solver was able to optimally solve all the first 10 instances, except for number 7, for which the found solution is sub-optimal. The performance of the CVC4 solver is worse, but it can still find an optimal solution for some initial instances. As noted earlier, the binary search is faster and gives better results.

4.4 Other experiments

As already mentioned we developed another model similar to the one developed for the MIP approach. However, this model performed worse than the one explained due to the fact that has higher spacial complexity.

5 MIP model

The Multiple Couriers Planning Problem can be represented as a mixed integer programming model, for which we employed the Python library PuLP. Initially, we attempted to use an encoding based on the SMT model formulation; however, due to efficiency concerns, we opted for an alternative model formulation, which is presented below.

5.1 Decision variables

This model is founded on the assumption that the items can be represented by nodes of a directed graph, whose arcs can form a path which represent the path the couriers do to deliver them. The structure of such model makes use of the following variables:

- **x:** a binary three dimensional tensor variable of shape $((n+1)\times(n+1)\times m)$. x_{ijk} is set to 1 if there exists an arc from node i to node j in the optimal route and it is driven by the courier k. x_{iik} is set to 0 since it's not possible for a courier k to travel from a node i to itself. The index n is considered as depot.
- **load_courier:** an integer one dimensional array of size m, which stores the load carried by each courier.
- **dist_courier:** an integer one dimensional array of size m, which encodes the distance travelled by every courier.
- $\boldsymbol{\rho}$ **(objective function)**: The maximum distance travelled among all the couriers.

5.2 Constraints

Next, in the following lines, we will analyze the constraints used by the model.

- Ensure that the number of times a courier enters a node is equal to the number of times it leaves that node

$$\sum_{i=0}^{n} x_{ijk} = \sum_{i=0}^{n} x_{jik} \qquad \forall j \in \{0,..,n\}, k \in \{0,..m-1\}$$

- As a second constraint we have to ensure that every node, except the depot, is entered just once.

$$\sum_{k=0}^{m-1} \sum_{i=0}^{n} x_{ijk} = 1 \qquad \forall j \in \{1,..,n\}$$

- Every courier has to leave the depot:

$$\sum_{j=0}^{n-1} x_{n-1jk} = 1 \qquad \forall k \in \{0,..m-1\}$$

  The combination with the first constraint allows for each courier to come back to the depot.

- The load capacity of couriers must not be exceeded.

$$\sum_{i=0}^{n} \sum_{j=0}^{n-1} s_j x_{ijk} \leq l_i \qquad \forall k \in \{0,..m-1\}$$

However, the solution obtained by considering only these constraints may still be infeasible for the actual problem, as it may contain sub-tours [2]. A sub-tour in the vehicle routing problem refers to a portion of a vehicle's route that doesn't start and end at the depot.
We have implemented three formulations for the elimination of sub-tours:

1. The initial concept involves introducing a new variables Y_{ki} to represent the position of the place i visited by the courier k during the tour. When x_{ijk} is set to 1 it means that j is visited right after i, so, we impose to Y_{ki} to be at least $Y_{ki} + 1$. This constraint can be formulated as follows:

   $$x_{ijk} \times (n-1) + Y_{kj} - Y_{ki} \leq n-2 \quad \forall k \in \{0, ..., m-1\}, i, j \in \{0, ..., n-1\}$$

   This is because if $x_{ijk} = 1$, then the constraint becomes $(n-1)+Y_{ki}-Y_{kj} \leq n-2 \implies Y_{kj} \geq Y_{ki} + 1$; if instead $x_{ijk} = 0$, it is always verified because $Y_{ki} - Y_{kj} \leq n-2$ and we know that $Y_{ki} \in \{1, .., n-1\}$.

2. Miller, Tucker, and Zemlin proposed a similar formulation [2] with the same temporal variable Y_{ki}. The difference comes in the constraint formulation:

   $$Y_{kj} \geq Y_{ki} + 1 - (2n)(1 - x_{ijk}) \quad \forall k \in \{0, ..., m-1\}, i, j \in \{0, ..., n-1\}, i \neq j$$

   Whenever $x_{ijk} = 1$, it becomes $Y_{kj} \geq Y_{ki} + 1$, which aligns with our intended condition; if instead $x_{ijk} = 0$ the constraint is always verified.

3. The last one is the Dantzig Fulkerson Johnson (DFJ) formulation [2] that eliminates sub-tours through subsets. Initially, the model is computed without any sub-tour elimination constraint. Then we repeatedly enforce, until no sub-tour is present, that the number of $x_{ijk} = 1$ inside the sub-tour is less than the number of places visited by the sub-tour. If no sub-tour are present then we have found the optimal solution.

5.3 Validation

Experimental design

Our choice to utilize CBC and GLPK solvers for testing was primarily driven by their cost-effectiveness as they are open-source and their versatility in solving a wide range of linear integer programming problems.

Experimental results

The MIP model has been implemented using the PuLP library. In Table 4, we present the results obtained with the CBC and GLPK solvers, with a focus on the performances of the three sub-tour elimination formulations. We also applied a symmetry-breaking constraint similar to the one discussed for the CP model. However, we chose not to include a discussion on it since it did not lead to any improvement in the model's performance. The absence of certain instances in the table is due to the model's inability to solve them within the required time frame (300 s).

Instance	CBC			GLPK		
	lin	MTZ	DFJ	lin	MTZ	DFJ
1	**14**	**14**	**14**	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**	**226**	**226**
3	**12**	**12**	**12**	**12**	**12**	**12**
4	**220**	**220**	**220**	**220**	**220**	**220**
5	**206**	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**	**322**	**322**
7	**167**	183	**167**	186	239	N/A
8	**186**	**186**	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**	**436**	N/A
10	**244**	**244**	**244**	N/A	**244**	**244**
13	828	786	N/A	N/A	N/A	N/A
16	507	437	N/A	N/A	N/A	N/A

Table 4: MIP results with PuLP solver.

5.4 Other experiments

As mentioned before, we also developed a model similar to the SMT formulation described in section 4. However, the results were unsatisfactory, even for the smallest instances.

6 Conclusions

This project has allowed us to delve into the realm of optimization and explore various methodologies and infrastructures to achieve satisfying results. We made our best efforts to tackle the Capacitated Vehicle Routing Problem from different angles, enabling us to grasp the significance of not only the solver we employ but also the problem's proper modeling.

Future works could expand this project by incorporating more complex or specific variants of the Vehicle Routing Problem. This could involve handling additional constraints or considering new aspects in the decision-making process. We developed also a graphic visualization for the solution of the problem, reported in the appendix (attachment paper of this project).

References

[1] C. Barrett and C. Tinelli, *Satisfiability modulo theories*. Springer, 2018.

[2] "Capacitated vehicle routing problem formulation," `https://how-to.aimms.com/Articles/332/332-Formulation-CVRP.html`.