Multiple Couriers Planning Problem

Davide Crociati davide.crociati2@studio.unibo.it
Lorenzo Pellegrino lorenzo.pellegrino2@studio.unibo.it
Davide Sonno davide.sonno@studio.unibo.it

July 2024

1 Introduction

The following report contains the main ideas to solve the multiple couriers planning problem using the proposed optimization techniques. All three of us collaborated on developing the CP model, as well as contributing to the other approaches. However, each of us has mostly done one of the remaining approaches:

- Lorenzo Pellegrino worked mainly on SAT model
- Davide Sonno worked mainly on SMT model
- Davide Crociati worked mainly on MIP model

Completing the project required two months of work. The main challenges involved solving the larger instances in CP and dealing with the Python APIs for integrating with the different solvers.

1.1 Instances

The given instances have a fixed structure with the following input parameters:

- m: number of couriers
- n: number of items
- l: the array of load capacities l_c for each courier
- s: the array of item sizes
- D: the matrix of distances between distribution points

1.2 Bounds

Bounds on the objective function were introduced considering the formulation of the problem. Since each item must be delivered, the minimum value the longest trip can take is achieved by delivering only the furthest item and going back to the depot. So, let r_i, c_i be the i-th element of the last row and of the last column, then the lower bound is:

$$lb = max(r_i + c_i)$$

Since the triangular inequality holds by definition of the problem, to define the upper bound we exploited the fact that exists at least one optimal solution where each courier deliver at least one item.

Proof. Given the condition that for all i, j, k the distances satisfy the triangle inequality $D_{i,k} \geq D_{i,j} + D_{j,k}$, Consider a delivery route for a courier visiting a sequence of locations starting and ending at a depot Dep, represented as $\{Dep, I_1, ..., I_k, ..., I_n, Dep\}$, and the distances between consecutive locations in the route are given by $\{D_{Dep,1}, D_{1,2}, ..., D_{k-1,k}, ..., D_{n,Dep}\}$.
If we suppose that another courier delivers some of the items it's possible to split the initial route into the two following routes:$\{Dep, I_1, ..., I_k, Dep\}$ and $\{Dep, I_{k+1}, ..., I_n, Dep\}$ with the following relation on the distances:

$$\begin{cases} \sum D_{Dep,1}, D_{1,2}, ..., D_{k-1,k}, ..., D_{n,Dep} > \sum D_{Dep,1}, D_{1,2}, ..., D_{k-1,Dep} \\ \sum D_{Dep,1}, D_{1,2}, ..., D_{k-1,k}, ..., D_{n,Dep} > \sum D_{Dep,k}, D_{k,k+1}, ..., D_{n,Dep} \end{cases}.$$

Since it's verified that $n \geq m$, it follows that it is always possible to find an optimal solution where each courier delivers at least one item. □

In particular, supposing that $m-1$ couriers deliver just one item (the closer ones) and the remaining courier deliver all the furthest $n - m + 1$ remaining items, then the longest possible distance is computed as the sum of the maximum for each row (sorting the maximum of each row we can also say that the last $n - m + 1$ items are the furthest and consider only those ones).

$$ub = \sum_{i=m-1}^{n} max(D_i)$$

The times to compute lower bounds and upper bounds are relatively small compared to the total time available to solve an instance, hence they won't be considered in the timeout computation (0.05 seconds needed to compute the bounds for all the 21 instances).

1.3 Hardware

All the experiments were conducted using the following hardware: Intel Core i3 10600KF, 16 GB RAM 3600 MHz, running Windows 10.

2 CP Model

The Constraint Programming model has been built using the MiniZinc language and tested with two different solvers.

2.1 Decision variables

The CP Model is based on the following decision variables:

- A matrix $stops \in \mathbb{N}^{m \times (n-m+1)}$ with domain $\{1..n+1\}$ that represent the order stops that each courier must complete in its route (the depot value is $n+1$). Example with 2 couriers and 6 items:

$$\begin{bmatrix} 1 & 3 & 4 & 7 & 7 \\ 5 & 2 & 6 & 7 & 7 \end{bmatrix}$$

  The width of this matrix assumes the following: given that each couriers delivers at least one item, then the maximum number of items that a single courier could deliver is given by the number of items n minus the number of couriers m plus 1. Using m less variables the models perform a little better.

- An array $item\_responsibility \in \mathbb{N}^n$ with domain $\{1..m\}$ that stores for each item who is the courier responsible for the delivery

- An array $distances\_traveled \in \mathbb{N}^m$ with domain $\{1..ub\}$ that stores for courier the total distance covered

2.2 Auxiliary variables

We have introduced an auxiliary array $successors \in \mathbb{N}^n$ with domain $\{1..n+1\}$ that stores for each item which is the next one to be delivered. Originally we computed the distances traveled by any courier c summing on $D_{stops_{c,i},stops_{c,i+1}}$. However, this process was extremely slow. By using this auxiliary array, we have significantly speeded up the computation. It allows access to D as $D_{i,successors_i}$.

2.3 Objective function

The objective function to minimize is $longest\_trip = max(distances\_traveled)$.

2.4 Constraints

To reduce the search space we tried to use global constraints whenever it was possible.

- First of all we impose that the sum of item sizes assigned to each courier must not exceed the courier's capacity with the global constraint

$$bin\_packing\_capa(l, item\_responsibility, s)$$

- Each courier must be responsible for delivering at least one item with

$$nvalue(m, item\_responsibility)$$

- Each courier must stops in at least one distribution point

$$stops_{c,1} \neq n+1 \quad \forall c \in 1..m$$

- The number of items delivered by a courier in '*item_responsibility*' matches the stops in '*stops*'

$$i \leq count(item\_responsibility, c) \leftrightarrow stops[c,i] \neq n{+}1, \quad \forall c \in 1..m, \forall i \in 1..n{-}m{+}1$$

- All the items assigned to a courier in '*item_responsibility*' must appear in the courier's route in '*stops*'

$$item\_responsibility_i = c \implies member(row(stops, c), i) \quad \forall c \in 1..m, \forall i \in 1..n$$

- Compute the successor nodes for the auxiliary variable

$$stops_{c,i} \neq n{+}1 \implies successors_{stops_{c,i}} = stops_{c,i+1} \quad \forall c \in 1..m, \forall i \in 1..n{-}1$$

- The distance traveled by each courier is the sum of the distances between each item they are responsible for and its successor, plus the distance from the depot to the first item, and the distance from the last item back to the depot

$$distances\_traveled_c = (\sum_{\substack{i=1 \\ item\_responsibility_i = c}}^{n} D_{i,successors_i}) + D_{n+1,stops_{c,1}} + D_{stops_{c,n-m+1},n+1} \quad \forall c \in 1..m$$

Implied constraints

- All items must be delivered just one time

$$count(stops, i, 1) \quad \forall i \in 1..n$$

Symmetry breaking constraints

A symmetry breaking constraint has been imposed: if two couriers have same capacity, then the first items delivered by the two couriers should have their indexes ordered according to the couriers indexes.

$$l_{c_1} = l_{c_2} \implies stops_{c1,1} < stops_{c2,1}, \forall c1, c2 \in 1..m, c1 < c2, c1 \neq m$$

It has also been considered a symmetry only valid for the instances where the distance matrix is symmetric (we have verified that this property is true for instances 11-21). In this case solutions for *stops* of the following type:

$$\begin{bmatrix} 1 & 3 & 4 & 7 & 7 \\ 5 & 2 & 6 & 7 & 7 \end{bmatrix}, \begin{bmatrix} 4 & 3 & 1 & 7 & 7 \\ 6 & 2 & 5 & 7 & 7 \end{bmatrix}$$

are equivalent. We tried to impose an order on the first and on the last item different from the depot but without improving the performances, hence we discarded it.
Due to the role and construction of *stops*, we had another symmetry regarding the fact that a solution of the type $[1, 2, D, D, D]$ and $[1, 2, 2, D, D]$ were both possible and feasible. This is avoided with a constraint matching the number of items delivered by a courier in '*item_responsibility*' and the relative '*stops*'. In other worlds each item can only appear once in *stops*, except for the *depot*.

2.5 Validation

We tested the CP model with the following solver:

- Gecode
- Chuffed

Every solver has been tested with and without symmetry breaking constraints, and for the Chuffed solver we also tested the use of free search.

Experimental design

Four different models have been designed in total: two for Gecode, one with Symmetry Breaking and one without, and two for Chuffed, likewise with and without Symmetry Breaking.
For Gecode's search strategy, we utilized first fail as the variable choice heuristic, and *indomain_split*, for the search on the variables stops and item responsibility. The strategy used for Chuffed is solely a first fail choice heuristic and *indomain_min* for the variable *stops*.
Lastly, we introduced Luby restarts, with a factor of 200, and the annotation *relax_and_reconstruct* for "simple large neighbourhood search strategy: upon restart, for each variable in x, the probability of it being fixed to the previous solution is percentage" [6] and using $percentage = 90$. Moreover we used *indomain_random* for variable assignments to introduce random effects when restarting. This was possible only using Gecode since Chuffed doesn't support neither *relax_and_reconstruct* nor *indomain_random*.

Experimental results

In the following table 1 are presented the results for the two solver with and without Symmetry Breaking and for Chuffed with and without free search:

INST	C	C+SB	G	G+SB	G+restart	G+SB+restart
1	**14**	**14**	**14**	**14**	-	-
2	**226**	**226**	**226**	**226**	-	-
3	**12**	**12**	**12**	**12**	-	-
4	**220**	**220**	**220**	**220**	-	-
5	**206**	**206**	**206**	**206**	-	-
6	**322**	**322**	**322**	**322**	-	-
7	**167**	**167**	**167**	**167**	-	-
8	**186**	**186**	**186**	**186**	-	-
9	**436**	**436**	**436**	**436**	-	-
10	**244**	**244**	**244**	**244**	-	-
11	*1051*	*1051*	*1175*	*1175*	*312*	*354*
12	*553*	*555*	*754*	*756*	**346**	**346**
13	*1706*	*1706*	*1188*	*1188*	*494*	*558*
14	*1749*	*1749*	*1884*	*1884*	*616*	*535*
15	*1081*	*1081*	*1175*	*1175*	*726*	*N/A*
16	**286**	**286**	**286**	**286**	**286**	**286**
17	*1414*	*1403*	*1396*	*1396*	*N/A*	*1365*
18	*1374*	*1374*	*1736*	*1736*	*456*	*424*
19	**334**	**334**	*450*	*450*	**334**	**334**
20	*1664*	*1664*	*1685*	*1685*	*N/A*	*N/A*
21	*1298*	*1298*	*1407*	*1407*	**374**	**374**

Table 1: C stands for chuffed, G for Gecode, SB for Symmetry Breaking. Bold values are the optima.

The symmetry breaking constraints are doing (almost) no difference except for Gecode with restarts. In this case, the values differ with respect to the base model, sometimes providing better objective values.

3 SAT Model

To solve the problem only using logic propositional formulas we have tried two different models and explored many search strategies.

3.1 Decision variables

- The first model has only one decision variable $stops \in \{0,1\}^{m \times n \times (n-m+1)}$ where $stops_{c,i,o} = 1$ if the courier c delivers the item i as his o-th delivered item, 0 otherwise. Since we had to use only boolean variables the third dimension is the one-hot encoding of the numbers.
- The third one, the circuit model, has a $stops \in \{0,1\}^{m \times n}$ where $stops_{c,i} = 1$ if the courier c delivers the item i. Then the order of the delivers is modelled by $successor \in \{0,1\}^{n \times n}$ where $successor_{i,j} = 1$ if the item i is delivered before the item j. In addition we used $predecessor = successor^T$

where $predecessor_{i,j} = 1$ if the item i is delivered after the item j. In the end we added also $reach_{i,j} \in \{0,1\}^{n \times n}$ in order to map the concept of reachability, useful for avoiding subcircuits in *successor* .

3.2 Objective function

The Z3 [2] solver has been uses to solve the model. It doesn't have a SAT optimizer, so in order to implement the minimization of the maximum among the total distances traveled by the couriers in their tour, we have tried many of them:

- incremental_lower_upper: the search starts by giving the solver the lower bound and if it is 'unsat' start increasing the lower bound by an incremental factor until it is 'sat' and in that case we decrease by 1 since we could have incremented too much.
- binary_search : the search starts by giving the solver the middle point among lower and upper bounds computed as $(upper\_bound + lower\_bound)/2$, then when a sat solution is found, the upper bound is updated; when an unsatisfiable solution is found the lower bound is updated.

3.3 Constraints

The circuit model resulted to be the best in terms of performance, both from the runtime side and the result side, so we'll report only its constraints. In order to represent the 'if' formulation in the expression we adopted the product formulation. For example

$$stops_{c,i} \cdot item\_sizes_i \equiv \begin{cases} item\_sizes_i & \text{if } stops_{c,i} = True \\ 0 & \text{otherwise} \end{cases}$$

1. Each items must be delivered once by one courier

$$\bigwedge_{i=1}^{n} \text{exactly\_one}(\{stops_{c,i} \mid c \in 1..m\})$$

2. Each courier must deliver at least one item

$$\bigwedge_{c=1}^{m} \text{at\_least\_one}(\{stops_{c,i} \mid i \in 1..n\})$$

   (a) so there will always be one first element, that has no predecessors.

$$\bigwedge_{c=1}^{m} \text{exactly\_one}\left(\left\{ stops_{c,i} \wedge \neg \bigvee_{j=1}^{n} predecessor_{i,j} \;\middle|\; i \in 1..n \right\}\right)$$

(b) and there will always be one last element, that has no successors.

$$\bigwedge_{c=1}^{m} \text{exactly\_one}\left(\left\{ stops_{c,i} \wedge \neg \bigvee_{j=1}^{n} successor_{i,j} \;\middle|\; i \in 1..n \right\}\right)$$

3. Items cannot succeed and precede themselves

$$\bigwedge_{i=1}^{n} (\neg successor_{i,i}) \wedge (\neg predecessor_{i,i})$$

4. Each item must not have more than one successor and one predecessor

$$\bigwedge_{i=1}^{n} \left(\text{at\_most\_one}(\{successor_{i,j} \mid j \in 1..n\})\right)$$

$$\bigwedge_{i=1}^{n} \left(\text{at\_most\_one}(\{predecessor_{i,j} \mid j \in 1..n\})\right)$$

5. Each item and its successor must be delivered by the same courier

$$\bigwedge_{i=1}^{n} \left(\left\{ (successor_{i,j} \wedge stops_{c,i}) \rightarrow stops_{c,j} \mid j \in 1..n \right\}\right)$$

6. Bin packing: the sum of items sizes must not exceed the max size of the courier

$$\bigwedge_{c=1}^{m} \left(\sum_{i=1}^{n} (stops_{c,i} \cdot item\_sizes_i) \leq courier\_capacities_c \right)$$

7. Symmetry breaking: the higher your capacity the higher your load

$$\bigwedge_{\substack{c_1=1 \\ courier\_capacities_{c_1} \\ > \\ courier\_capacities_{c_2}}}^{m} \bigwedge_{c_2=1}^{m} \left(\sum_{i=1}^{n} (stops_{c_1,i} \cdot item\_sizes_i) > \sum_{i=1}^{n} (stops_{c_2,i} \cdot item\_sizes_i) \right)$$

8. Break subcircuit

   (a) An item i can reach itself (base of reachability)

$$\bigwedge_{i=1}^{n} reach_{i,i}$$

(b) If item i can reach item j and item j can reach item k, then item i can reach item k

$$\bigwedge_{i=1}^{n}\bigwedge_{\substack{j=1\\ i\neq j}}^{n}\bigwedge_{\substack{k=1\\ k\neq j\\ k\neq i}}^{n}\Big((reach_{i,j}\wedge reach_{j,k})\rightarrow reach_{i,k}\Big)$$

(c) If item i has a successor j, then i can reach j

$$\bigwedge_{i=1}^{n}\bigwedge_{j=1}^{n}\Big(successor_{i,j}\rightarrow reach_{i,j}\Big)$$

(d) Prevent cycles (an item should not be able to reach itself through any other item)

$$\bigwedge_{i=1}^{n}\bigwedge_{\substack{j=1\\ i\neq j}}^{n}\neg(reach_{i,j}\wedge reach_{j,i})$$

9. Check if the distances traveled are at most the maximum distance given to the solver, and compute the distance by summing:

(a) The distance between the depot and the first item

$$D_{depot,c,i}=(stops_{c,i}\wedge\neg\bigvee_{j=1}^{n}predecessor_{i,j})\cdot distances_{n+1,i}$$

(b) The distance between the first and the last item

$$D_{c,i}=\sum_{j=1}^{n}\Big[(stops_{c,i}\wedge successor_{i,j})\ \cdot\ distances_{i,j}\Big]$$

(c) The distance between the last item and the depot

$$D_{c,i,depot}=(stops_{c,i}\wedge\neg\bigvee_{j=1}^{n}successor_{i,j})\cdot distances_{i,n+1}$$

Each courier must travel a maximum distance of MAX

$$\bigwedge_{c=1}^{m}\Big(\sum_{i=1}^{n}(D_{depot,c,i}+D_{c,i}+D_{c,i,depot})\leq MAX\Big)$$

3.4 Validation

In the following tables we report the results obtained with the two models: the circuit encoding (C) and the 1-hot encoding (1). In the first one we used both the

search strategies: the incremental lower upper (ILU) with $incremental\_factor = 30$ and the binary search (BS). In the other model we only got promising results when using binary search.

INST	C+ILU+SB	C+ILU	C+BS+SB	C+BS	1+BS+SB	1+BS
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**
13	*N/A*	*N/A*	1802	1178	*N/A*	*N/A*
16	*N/A*	*N/A*	416	479	*N/A*	*N/A*
19	*N/A*	*N/A*	539	1685	*N/A*	*N/A*

Table 2: SB stands for Symmetry Breaking. Bold values are the optima.. Instances with no solution found are not reported

ILU resulted to be way quicker with small instances (1-10) while binary-search is more efficient with bigger instances.

4 SMT Model

The language used to construct the model is SMT-LIB [8], widely used and compatible with the majority of the solvers. The instances are parsed into a model, retrieved after the solving phase.

4.1 Decision variables

Before introducing the variables used, we would like to show the equivalence between the use of arrays and variables representing the elements of the array. The two structures are going to hold the exact same values, so we can always express the values with one or the other.
The only notable difference is the way an array can be accessed: since it is enough to use (assert(= 1(select $array$ $index$))) to impose $array[index] = 1$, when using the variables we need to iterate over all the possible indexes to find a match.
The decision variables used are the same defined for CP.
Practically we kept one more column to simplify the distance computation.
Some differences are in the way they are represented: the model can be constructed with or without the use of arrays. For example *courier_capacities* can

be represented as an (*Array Int Int*) or with one variable for each index, therefore *courier_1_capa*, *courier_2_capa*, and so on.
This same argument is valid for all the other variables except for the *distances* and *stops*. Being matrices, each row is adapted into a single array, having as many arrays as the number of rows.

4.2 Objective function

In SMT-LIB there is no native concept of minimization or maximization of an objective. To be able to execute a search, we start from the upper bound of the instance and whenever a sat solution is found, the distances traveled are analyzed and the longest one, minus one, becomes the new upper-bound, until the timeout or an unsat model.
To be able to achieve this behaviour, the model is computed on the instance data, by defining all the constants and assertions. Inside a loop, the objective is added as an upper bound for the distances traveled and the model is checked. The last sat model produced is saved in *.smt2* format, so that it could be run using different solvers.

4.3 Optimizer

We also tried a model with the same variables and constraints, but entirely built on *Z3* and using its optimizer. In this case we got really close to the optimal value, unfortunately this can be only exploited while solving with Z3.

4.4 Constraints

Most of the constraints are inspired from the CP model. They are going to be expressed in array notation; for simplicity arrays starts with index one and $depot = num\_items + 1$. The width of *stops* is $n - m + 2$.

- Bin packing over couriers:

$$\bigwedge_{c=1}^{m} loads[c] = \sum_{i=1}^{n} \Big(\text{if } items\_resp[i] = c \text{ then } items\_size[i] \text{ else } 0\Big)$$

- Every courier delivers at least one item:

$$\bigwedge_{c=1}^{m} \left(\bigvee_{i=1}^{n} (items\_resp[i] = c) \right)$$

- Every item has to be delivered:

$$\bigwedge_{i=1}^{n} \left(\sum_{c=1}^{m} \sum_{j=1}^{n-m+1} \Big(\text{if } stops_c[j] = i \text{ then } 1 \text{ else } 0\Big) = 1 \right)$$

- Construct *stops* using *item_resp*:

$$\text{let } i \leq \sum_{j=1}^{n} \Big(\text{if } items\_resp[j] = c \text{ then } 1 \text{ else } 0\Big) \tag{a}$$

$$\text{let } stops_c[i] < depot \tag{b}$$

$$\bigwedge_{c=1}^{m} \bigwedge_{i=1}^{n-m+2} \Big((a \implies b) \wedge (b \implies a)\Big)$$

- If an item has to be delivered by a certain courier, that item has to appear in the stops of that particular courier:

$$\bigwedge_{c=1}^{m} \bigwedge_{i=1}^{n} \Big((items\_resp[i] = c) \implies \bigvee_{k=1}^{n-m+2} (stops_c[k] = i)\Big)$$

- Construct *successors* from *stops*:

$$\bigwedge_{c=1}^{m} \bigwedge_{i=1}^{n-m+1} \Big((stops_c[i] \neq depot) \implies (successors[stops_c[i]] = stops_c[i{+}1])\Big)$$

- Calculate distances using *successors*:

$$\bigwedge_{c=1}^{m} distances\_traveled[c] =$$
$$= \sum_{i=1}^{n} \sum_{k=1}^{N+1} \Big(\text{if } (items\_resp[i] = c) \wedge (successors[i] = k) \text{ then } distances_i[k] \text{ else } 0\Big)$$
$$+ distances_{depot}[stops_c[1]]$$

Symmetry breaking constraints

Once again the same symmetries of the CP model are present, namely the equivalence of the two routes $1, 1, 2 \leftrightarrow 1, 2, 2$. This cannot happen due to way stops is constructed.
In this model a new symmetry is introduced, constraining that couriers with higher capacities deliver more weight.
The previously said is ensured by:

$$\bigwedge_{c1=1}^{m} \bigwedge_{c2=1}^{m} \Big((couriers\_capa[c1] > couriers\_capa[c2]) \implies (loads[c1] > loads[c2])\Big)$$

4.5 Validation

The produced models have been tested using Z3 solver and cvc5 [7]. In the case of cvc5 there is no optimizer at the moment, therefore the same upper bound strategy has been used.

INST	Z3 OPT	Z3+A+SB	Z3+A	Z3+Best	C+A+SB	C+A	C+Best
1	**14**	**14**	**14**	**14**	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**	**226**	**226**	**226**
3	**12**	**12**	**12**	**12**	**12**	**12**	**12**
4	**220**	**220**	**220**	**220**	**220**	**220**	**220**
5	**206**	**206**	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**	**322**	**322**	**322**
7	**167**	*207*	*256*	*189*	*285*	*497*	*263*
8	**186**	**186**	**186**	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**	**244**	**244**	**244**
13	*1484*	*N/A*	*1616*	*1294*	*N/A*	*N/A*	*N/A*
16	*N/A*	*N/A*	*1337*	*479*	*N/A*	*N/A*	*N/A*

Table 3: C stands for cvc5. SB stands for Symmetry Breaking. OPT stands for Optimizer.A is when using arrays. Bold values are the optima. Instances with no solution found are not reported

5 MIP Model

The Mixed Integer Linear Programming model has been built using the library PuLP in python and tested with three different solvers.

5.1 Decision variables

The MIP model is based on the following decision variables:

- A 3D boolean variable x with shape $n \times n \times m$. $x_{i,j,c}$ indicates if courier c goes through the arc (i, j).
- A 2D real matrix $u \in \mathbb{R}^{n \times m}$ for the Miller-Tucker-Zemlin [5] subtour elimination.

5.2 Objective function

To minimize the longest distance travelled by the couriers, we have introduced a variable called *longest_trip* which is the one that will be minimized.

$$\sum_{i=0}^{n} \sum_{j=0}^{n} x_{i,j,c} \cdot D_{i,j} \leq longest\_trip, \ \forall c \in \{1..m\}$$

5.3 Constraints

- Each item is delivered exactly once

$$\sum_{c=1}^{m} \sum_{\substack{i=1 \\ i \neq j}}^{n+1} x_{i,j,c} = 1, \ \forall j \in \{1..n\}$$

- Each courier leaves and enters the depot once

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

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

- Each courier enters and leaves each location the same number of times

$$\sum_{\substack{i=1 \\ i \neq j}}^{n+1} x_{i,j,c} - \sum_{\substack{i=1 \\ i \neq j}}^{n+1} x_{j,i,c} = 0 \ \forall c \in \{1..m\}, \forall j \in \{1..n+1\}$$

- Each courier doesn't exceed its capacity

$$\sum_{i=1}^{n+1} \sum_{\substack{j=1 \\ i \neq j}}^{n} s_j \cdot x_{i,j,c} \leq l_c, \ \forall c \in \{1..n\}$$

For the subtour elimination, two different strategies were tested:

- Subtour elimination using full enumeration

$$\sum_{\substack{(i,j) \in S \\ i \neq j}} x_{i,j,c} \leq |S| - 1, \quad \forall S \subseteq \{1..n+1\}, |S| \geq 2, \forall c \in \{1..m\}$$

- Subtour elimination using Miller-Tucker-Zemlin formulation

$$u_{i,c} - u_{j,c} + 1 \leq (n-1)(1 - x_{i,j,c}), \ \forall i, j \in \{1..n\}, \forall c \in \{1..m\}, i \neq j$$

5.4 Validation

First of all the model was tested with the subtour elimination that uses a full enumeration of all possible subtours. This introduce a number of constraints exponential in n and lead to exhausted memory with big instances.
A possible way to obtain a polynomial number of constraint is to adapt the Miller-Tucker-Zemlin formulation for TSP to MCP problem. [9]

Both of the formulation were tried using CBC [1], GLPK [3] and HiGHS [4] solvers. The CBC solver has shown some problems with respecting the time-out while trying to solve the bigger instances (here are shown only the results computed within the time limit).

INST	C-EA	C-MTZ	G-EA	G-MTZ	H-EA	H-MTZ
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	*N/A*	**167**	*N/A*	*184*	*N/A*	**167**
8	**186**	**186**	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**	**244**	**244**
13	*N/A*	*586*	*N/A*	*N/A*	*N/A*	*482*
16	*N/A*	**286**	*N/A*	*N/A*	*N/A*	*323*
19	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*683*

Table 4: C stands for CBC. G for GLPK. H stands for HiGHS. EA stands for enumerate all the subtours. MTZ stands for Miller-Tucker-Zemlin

6 Conclusions

- With all the different approaches we were able to solve the first 10 instances to optimality.
- With CP we were able to find a solution for all the instances with both the solvers. In particular 14 instances were solved to optimality.
- With all the approaches we were able to find solutions for instances 13 and 16.
- Further developments could explore some benefits from the symmetry distance matrix, which is a property that holds in the bigger instances (11-21).
- More accurate upper/lower bounds could also be helpful in decreasing the search space.

References

[1] COIN-OR Foundation. COIN-OR Branch-and-Cut Solver (Cbc). https://github.com/coin-or/Cbc, 2024.

[2] Leonardo de Moura, Nikolaj Bjørner, and contributors. Z3: An Efficient SMT Solver. https://github.com/Z3Prover/z3, 2024.

[3] GNU Project. GNU Linear Programming Kit (GLPK). https://www.gnu.org/software/glpk/, 2024.

[4] Julian Hall and contributors. HiGHS - Linear Optimization Software. https://highs.dev/, 2024.

[5] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *J. ACM*, 7(4):326–329, oct 1960.

[6] MiniZinc Documentation. Relax and reconstruct - minizinc documentation. https://docs.minizinc.dev/en/stable/lib-gecode-annotations.html#relax-and-reconstruct, 2024.

[7] The cvc5 Development Team. cvc5 solver. https://cvc5.github.io/.

[8] The SMT-LIB Initiative. SMT-LIB Standard. https://smt-lib.org/.

[9] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, 2002.