The Multiple Couriers Planning problem

Rubin Carkaxhia - rubin.carkaxhia@studio.unibo.it

August 31, 2024

1 Introduction

The following report is comprised of 5 parts and it contains the main ideas and strategies deployed to solve the MCP using different optimization techniques. The project took a month and a half of daily work to complete. The main difficulties encountered were related to the lack of online resources and the significant amount of time required to test the various configurations, particularly in the CP part, where different models and search strategies were explored. Another difficulty was due to the fact that i had to kept trying to improve the models to try solving the larger instances, especially the SMT and MIP models suffered from memory issues when dealing with such instances.

1.1 Input and common variables

All the three different models used have the same input parameters and naming conventions:

- **num_couriers**: the number of couriers.
- **num_items**: the number of items to be distributed.
- **courier_size**: an array of integers representing the loading capacity of each vehicle.
- **item_size**: an array of integers which represents the size of each item.
- **distances**: which is the matrix of distances between items and between items and depot.

And the same objective variable:

1. **max_dist** with domain[LB,...,UB] which represents the length of the longest route of any of the couriers.

1.2 Bounds

To define the bounds on the objective variable an assumption was made: every courier must deliver at least one item. Thus, there can be two different types of paths a courier can travel: a simple path where the courier delivers only one item and then goes back to the depot, or a path in which it delivers more than one item which implies multiple stops for deliveries. Mathematically, the costs of these paths can be defined as:

$$cost_{simple} = distances[num\_items + 1, i] + distances[i, num\_items + 1], \quad \forall i \in num\_items$$

$$cost_{complex} = distances[num\_items + 1, i] + distances[i, z_1] + distances[z_1, z_2] + \cdots + distances[z_n, num\_items + 1],$$
$$\forall i, z_1, \ldots, z_n \in num\_items$$

Since the triangle inequality holds for all distances, we know that a direct path to deliver any item i is always the shortest possible route compared to any path passing through intermediate points. Mathematically, this is defined as:

$$distances[num\_items + 1, i] \leq distances[num\_items + 1, z_1] + distances[z_1, z_2] + \cdots + distances[z_n, i],$$
$$\forall i, z_1, \ldots, z_n \in num\_items$$

And the same also applies for the direct path that goes from an item i to the depot:

$$distances[i, num\_items + 1] \leq distances[i, z_1] + distances[z_1, z_2] + \cdots + distances[z_m, num\_items + 1],$$
$$\forall i, z_1, \ldots, z_m \in num\_items$$

Therefore we can consider as a lower bound the case where a courier is required to deliver only the most distant item. Which can be defined as:

$$LB = \max_{i \in num\_items} distances[num\_items + 1, i] + distances[i, num\_items + 1]$$

Considering the assumption we made before, we can say that a courier can deliver a maximum of Q items:

$$Q = num\_items - num\_couriers + 1$$

The upper bound was defined as the sum of 3 different costs:

1. The maximum cost to leave the depot to any distribution point

$$from\_depot = \max_{i \in num\_items} distances[num\_items + 1, i]$$

2. The maximum cost to return to the depot from any distribution point

$$to\_depot = \max_{i \in num\_items} distances[i, num\_items + 1]$$

3. The maximum cost between distribution points

$$between\_points = \max_{i,j \in num\_items, i \neq j} (distances[i, j])$$

   Since a courier can deliver up to Q items, we sort the values of *between_points* in a descending order and select the $Q-1$ largest values. The reason for that is that a courier delivering Q items will travel Q-1 times between the distribution points.

Therefore the upper bound is defined as:

$$UB = from\_depot + to\_depot + \sum_{k=1}^{Q-1} sorted\_between\_points[k]$$

1.3 Hardware specifications

All the experiments were conducted on a computer with the following specifications:

- Processor: AMD Ryzen 5 4600H with Radeon Graphics @ 3.00 GHz
- Ram: 16 GB
- Cores: 6
- Logical processors: 12

2 CP Model

The CP part was highly influenced by the first approach of the paper: "Using Constraint Programming to solve the Vehicle Routing Problem with Time Windows"[1]. Several integer index sets are defined for the CP models:

- **NODES**: represents all possible locations, including the depot, and is defined as 1..num_items+1
- **ITEMS**: represents the items, defined as 1..num_items
- **VEHICLES**: represents the couriers, defined as 1..num_couriers

An integer constant '**max_load**' was used to define an upper bound for the decision variable '**couriers_load**', and it calculates the maximum value in the "**courier_size**' input array.

2.1 Decision variables

The decision variables used are:

- A matrix **succ** $\in \mathbb{N}^{(\text{VEHICLES})\times(\text{NODES})}$, with domain [1,...,num_items+1] representing the route of each courier; $\text{succ}_{k,i} = j \ \forall i, j \in num\_items + 1 \iff$ a courier k goes from distribution point i to a distribution point j. Cells where the cell's content matches its own column denote a non-visit. For example, if we have $\text{succ}_{k,2} = 2$ it means the courier k does not visit the distribution point 2.
- An array **courier_assignment** $\in \mathbb{N}^{(\text{ITEMS})}$, with domain assigned to VEHICLES representing the courier assigned to each item; $\text{courier\_assignment}_i = k \iff$ item i is delivered by courier k.
- An array **u** $\in \mathbb{N}^{(\text{ITEMS})}$, that has a domain assigned to ITEMS, which is required for subtour elimination, implemented through the Miller-Tucker-Zemlin (MTZ) formulation[2], it gets a value for each node except the depot.

2.2 Objective function

Since the objective is to minimize the maximum distance traveled by any courier, the search algorithm is set to minimize the variable **max_dist**, the variable and its bounds were previously discussed in section 1.1 and 1.2.

$$\textbf{minimize } max\_dist = \max_{j \in num\_vehicles} \Big(\sum_{i \in NODES} distances[i, \text{succ}_{j,i}] \Big)$$

2.3 Constraints

We define the constraints used as follows:

1. Each vehicle visits the depot exactly once, and we achieve this by using the global constraint *count_eq*:

$$\forall j \in VEHICLES: \ count\_eq(\{\text{succ}_{j,i} \mid i \in \text{ITEMS}\}, \text{num\_items} + 1, 1)$$

2. A constraint that links the decision variable for courier assignment to the *succ* variable:

$$\forall i \in ITEMS, \ \forall j \in VEHICLES: \ (\text{succ}_{j,i} \neq i) \implies (\text{courier\_assignment}_i = j)$$

3. The capacity constraint of each vehicle is achieved by using the global constraint *bin_packing_capa*:

$$bin\_packing\_capa(courier\_size, courier\_assignment, item\_size)$$

4. To eliminate subtours, we use the variable u previously defined. If a vehicle drives from node i to node j, the value of u_j has to be bigger than the value of u_i. So each time a new node is being visited, the value for u_i increases ensuring that a vehicle will not drive in a circle. Since the depot does not get a value of u_i, it is possible to drive in a circle if the vehicle starts and ends at the depot.[2]

$$\forall j \in VEHICLES, \ \forall i \in ITEMS: \ u_i < u_{succ_{j,i}}$$

5. To ensure that each item is distributed only once, the global constraint *count_eq* was used:

$$\forall i \in ITEMS: \ count\_eq(\{\text{succ}_{j,i} \mid j \in \text{VEHICLES}\}, i, \text{num\_couriers} - 1)$$

6. The constraint that ensures that a vehicle cannot visit the same node again is achieved by using the global constraint *alldifferent*:

$$\forall j \in VEHICLES: \ alldifferent(\{\text{succ}_{j,i} \mid i \in \text{NODES})\}$$

7. We also use a constraint that ensures each vehicle to deliver at least one item:

$$\forall j \in VEHICLES: \ succ_{j,num\_items+1} \neq num\_items + 1$$

Symmetry breaking constraints

When couriers have the same size, they are interchangeable, meaning that swapping their assignments does not affect the solution's cost. A symmetry breaking constraint was used to eliminate these redundant equivalent solutions by enforcing an ordering among couriers with the same capacity. In our model, we utilize the function *row* to extract the full route of each vehicle from the successor array *succ* when two vehicles have the same capacity, and then we order their routes lexicographically.

$$\forall c1, c2 \in VEHICLES,\ c1 < c2,\ courier\_size_{c1} = courier\_size_{c2} \implies lex\_less(row[succ, c1], row[succ, c2])$$

2.4 Validation

Experimental design

The models were all implemented in MiniZinc and ran by using both Gecode and Chuffed. Different search strategies and restarts techniques were implemented to assess their performance.
For Gecode, the best leading experimental setup is:

- **model_fail_rand_luby**: *int_search* with $first\_fail$ and $indomain\_random + restart\_luby$ with scale of 50.

for Chuffed:

- **model_fail_split_chuffed**: *int_search* with $first\_fail$ and *indomain_split*.

Additionally, several other experimental setups were explored, for Gecode:

- **model_dom_rand_linear**: *int_search* with *dom_w_deg* and $indomain\_random + restart\_linear$ with scale of 250.
- **model_dom_rand_luby**: *int_search* with *dom_w_deg* and $indomain\_random + restart\_luby$ with scale of 250.
- **model_fail_rand_lin**: *int_search* with $first\_fail$ and $indomain\_random + restart\_linear$ with scale of 250 with and without SB.

for Chuffed:

- **model_fail_min_chuffed**: int_search with $first\_fail$ and *indomain_min* with and without SB.

The computer specifications were already discussed in Section 1.3. The version of MiniZinc used is 2.8.4, while the versions of Gecode and Chuffed are 6.3.0 and 0.13.2, respectively. A time constraint of 300 seconds was established for solving.

Experimental results

The best models were tested both with and without the symmetry-breaking constraint, and the results are available in Table 1. The results of the other experimental setups can be found in Table 4 at the end of the report.

3 SMT

For the SMT model, we started from the formulation of the CP model and we took inspiration from the constraints of [3] to improve the model, the solver used is Z3.

3.1 Decision Variables

- A 3-dimensional list of binary variables used to determine the path each courier takes:

$$visit_{i,j,k} \in \{0, 1\}\ \forall i \in num\_couriers,\ \forall j \in num\_items + 1,\ \forall k \in num\_items + 1$$

  We have $visit_{i,j,k} = 1 \iff$ courier i travels from distribution point j to distribution point k

ID	Chuffed + SB	Chuffed w/out SB	Gecode + SB	Gecode w/out SB
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	1173	1203	N/A	834
12	956	963	N/A	598
13	1746	1746	936	914
14	1664	1926	N/A	1158
15	1084	1169	N/A	1075
16	286	359	N/A	351
17	1414	1379	N/A	1415
18	1342	1450	N/A	1005
19	764	873	N/A	481
20	1390	1685	N/A	1430
21	1566	1584	N/A	958

Table 1: Best CP models results.

- A 2-dimensional list of binary variables used to determine which courier is responsible for delivering which items:

$$load_{i,j} \in \{0,1\} \ \forall i \in num\_couriers, \ \forall j \in num\_items$$

  $load_{i,j} = 1 \iff courier_i$ is assigned to deliver $item_j$

- A 2-dimensional list of Integer variables to implement subtours elimination using MTZ formulation:

$$u_{i,j} \in \mathbb{Z}, \ \forall i \in num\_couriers, \ \forall j \in num\_items + 1$$

3.2 Objective Function

The goal of the objective function is to minimize the objective variable max_dist defined in section 1.1.
The constraint used is:

$$\sum_{j=0}^{num\_items} \sum_{k=0}^{num\_items} visit_{i,j,k} \cdot distances_{j,k} \leq max\_dist \quad \forall i \in num\_couriers$$

3.3 Constraints

1. Each item is assigned to exactly one courier:

$$\sum_{i=1}^{num\_couriers} load_{i,j} = 1 \ \ \forall j \in num\_items$$

2. For each distribution point i, exactly one courier should depart from it and arrive at it:

$$\sum_{k=1}^{num\_couriers} \sum_{j=0}^{num\_items} visit_{k,i,j} = 1 \ \ \forall i \in num\_items$$

$$\sum_{k=1}^{num\_couriers} \sum_{j=0}^{num\_items} visit_{k,j,i} = 1 \quad \forall i \in num\_items$$

3. For each distribution point i, if a courier enters the distribution point, they must also leave that location:

$$\sum_{j=0}^{num\_items} visit_{k,i,j} = \sum_{j=0}^{num\_items} visit_{k,j,i} \quad \forall i \in num\_items, \ \forall k \in num\_couriers$$

4. If a courier k either arrives at or departs from the distribution point of an item i, then the item i is part of that courier's delivery route:

$$\sum_{j=0}^{num\_items} visit_{k,i,j} = load_{k,i} \quad \forall i \in num\_items, \ \forall k \in num\_couriers$$

$$\sum_{j=0}^{num\_items} visit_{k,j,i} = load_{k,i} \quad \forall i \in num\_items, \ \forall k \in num\_couriers$$

5. MTZ constraint for subtour elimination[4]:

$$u_{i,j} - u_{i,k} + num\_items \cdot visit_{i,j,k} \leq num\_items - 1, \quad \forall i \in num\_couriers, \ \forall j \in num\_items, \ \forall k \in num\_items,$$
$$\text{where } j \neq num\_items \text{ and } j \neq k$$

6. Capacity constraint:

$$\sum_{j=0}^{num\_items-1} load_{i,j} \cdot item\_size_j \leq courier\_size_i \quad \forall i \in num\_couriers$$

7. No self-loop constraint:

$$\sum_{j=0}^{num\_items-1} visit_{i,j,j} = 0 \quad \forall i \in num\_couriers$$

8. Ensure each courier starts and ends at the depot exactly once:

$$\sum_{i=0}^{num\_items-1} visit_{k,num\_items,i} = 1 \ \forall k \in num\_couriers$$

$$\sum_{i=0}^{num\_items-1} visit_{k,i,num\_items} = 1 \ \forall k \in num\_couriers$$

3.4 Validation

Experimental design

Same hardware specifications already discussed in section 1.3, the version of Z3 that was used is: 4.12.1. To avoid memory issues when dealing with large instances, a 180 seconds time limit for constraints definition was used, plus the 300 seconds time limit for the solver.

Experimental results

The results of the SMT model can be found in Table 2.

ID	Z3
1	**14**
2	**226**
3	**12**
4	**220**
5	**206**
6	**322**
7	167
8	**186**
9	436
10	**244**
11	N/A
12	N/A
13	1366
14	N/A
15	N/A
16	N/A
17	N/A
18	N/A
19	N/A
20	N/A
21	N/A

Table 2: SMT results.

4 MIP

The MIP model follows the same approach of the previous ones, again we took inspiration from [3] to improve the model and the results.

4.1 Decision Variables

- A 3-dimensional list of binary variables used to determine the path each courier takes:

$$visit_{i,j,k} \in \{0,1\}\ \forall i \in num\_couriers,\ \forall j \in num\_items + 1,\ \forall k \in num\_items + 1$$

  We have $visit_{i,j,k} = 1 \iff$ courier i travels from distribution point j to distribution point k

- A 2-dimensional list of binary variables used to determine which courier is responsible for delivering which items:

$$load_{i,j} \in \{0,1\}\ \forall i \in num\_couriers,\ \forall j \in num\_items$$

  $load_{i,j} = 1 \iff courier_i$ is assigned to deliver $item_j$

- A 2-dimensional list of Integer variables to implement subtours elimination using MTZ formulation:

$$u_{i,j} \in \mathbb{Z}, \forall i \in num\_couriers,\ \forall j \in num\_items + 1$$

4.2 Objective Function

The goal of the objective function is to minimize the objective variable max_dist defined in section 1.1.
The constraint used is:

$$\sum_{j=0}^{num\_items} \sum_{k=0}^{num\_items} visit_{i,j,k} \cdot distances_{j,k} \leq \text{max\_dist}, \quad \forall i \in num\_couriers$$

4.3 Constraints

1. Each item is assigned to exactly one courier:

$$\sum_{i=1}^{num\_couriers} load_{i,j} = 1 \quad \forall j \in num\_items$$

2. For each distribution point i, exactly one courier should depart from it and arrive at it:

$$\sum_{k=1}^{num\_couriers} \sum_{j=0}^{num\_items} visit_{k,i,j} = 1 \quad \forall i \in num\_items$$

$$\sum_{k=1}^{num\_couriers} \sum_{j=0}^{num\_items} visit_{k,j,i} = 1 \quad \forall i \in num\_items$$

3. For each distribution point i, if a courier enters the distribution point, they must also leave that location:

$$\sum_{j=0}^{num\_items} visit_{k,i,j} = \sum_{j=0}^{num\_items} visit_{k,j,i} \quad \forall i \in num\_items, \ \forall k \in num\_couriers$$

4. If a courier k either arrives at or departs from the distribution point of an item i, then the item i is part of that courier's delivery route:

$$\sum_{j=0}^{num\_items} visit_{k,i,j} = load_{k,i} \quad \forall i \in num\_items, \ \forall k \in num\_couriers$$

$$\sum_{j=0}^{num\_items} visit_{k,j,i} = load_{k,i} \quad \forall i \in num\_items, \ \forall k \in num\_couriers$$

5. MTZ constraint for subtour elimination[4]:

$$u_{i,j} - u_{i,k} + num\_items \cdot visit_{i,j,k} \leq num\_items - 1, \quad \forall i \in num\_couriers, \ \forall j \in num\_items, \ \forall k \in num\_items,$$
$$\text{where } j \neq num\_items \text{ and } j \neq k$$

6. Capacity constraint:

$$\sum_{j=0}^{num\_items-1} load_{i,j} \cdot item\_size_j \leq courier\_size_i \quad \forall i \in num\_couriers$$

7. No self-loop constraint:

$$\sum_{j=0}^{num\_items-1} visit_{i,j,j} = 0 \quad \forall i \in num\_couriers$$

8. Ensure each courier starts and ends at the depot exactly once:

$$\sum_{i=0}^{num\_items-1} visit_{k,num\_items,i} = 1 \ \forall k \in num\_couriers$$

$$\sum_{i=0}^{num\_items-1} visit_{k,i,num\_items} = 1 \ \forall k \in num\_couriers$$

ID	OR-Tools	PuLP_CBC	PuLP_HIGHS	Docplex
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	185	167	167	**167**
8	**186**	**186**	**186**	**186**
9	436	436	436	**436**
10	**244**	244	**244**	**244**
11	N/A	N/A	N/A	N/A
12	N/A	N/A	N/A	N/A
13	680	N/A	N/A	546
14	N/A	N/A	N/A	N/A
15	N/A	N/A	N/A	N/A
16	500	N/A	286	286
17	N/A	N/A	N/A	N/A
18	N/A	N/A	N/A	N/A
19	N/A	N/A	N/A	443
20	N/A	N/A	N/A	N/A
21	N/A	N/A	N/A	N/A

Table 3: MIP results.

4.4 Validation

Experimental design

Same hardware specifications discussed in section 1.3. The MIP model is implemented using OR-Tools' python library, PuLP and the CPLEX Python API Docplex. For OR-Tools was used the solver SCIP while for PuLP were used both CBC and HIGHS.

Experimental results

The results of the MIP model are available in Table 3.

5 Conclusions

In this project, several approaches and libraries were used, each with its own advantages and disadvantages. MiniZinc offered the most powerful features and provided the best results, consistently finding a solution for all instances. The SMT with Z3 was able to find an optimal solution for most of the simpler instances. The MIP model using OR-Tools produced solutions for all the simpler instances and also for two of the more complex ones. With PuLP, the CBC solver was only able to find solutions for the simpler instances, while the Highs solver also provided a solution for instance 16. Docplex, being a commercial solver, delivered the best results for the MIP models, although it failed to solve several of the more complex instances. To improve results in CP, one could either discover new symmetry-breaking constraints or use a warm start. For the MIP model, there are two potential ways to enhance outcomes: using another commercial solver like Gurobi or employing an initial heuristic solution as a warm start.

References

[1] W. D. Frohlingsdorf, "Using constraint programming to solve the vehicle routing problem with time windows," University of Glasgow, April 2018. https://www.jacktex.eu/research/material/master_thesis.pdf

[2] AIMMS, "Miller-tucker-zemlin formulation," November 2020. https://how-to.aimms.com/Articles/332/332-Miller-Tucker-Zemlin-formulation.html

[3] Bruno Scalia C.F. Leite, "The vehicle routing problem: Exact and heuristic solutions", Towards Data Science, August 2023. https://towardsdatascience.com/the-vehicle-routing-problem-exact-and-heuristi

[4] Miller, Tucker, Zemlin, "Integer Programming Formulation of Traveling Salesman Problems" https://dl.acm.org/doi/abs/10.1145/321043.321046

ID	fail_rand_lin + SB	fail_rand_lin w/out SB	dom_rand_linear	dom_rand_luby	Chuffed + SB	Chuffed w/out SB
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	N/A	894	948	969	1189	1203
12	N/A	623	609	609	956	959
13	1020	984	1188	1156	1760	1760
14	N/A	1255	1287	1275	1664	1926
15	N/A	1055	1060	1042	1090	1175
16	N/A	336	416	417	286	359
17	N/A	1528	1475	1465	1428	1387
18	N/A	1025	1116	1047	1342	1462
19	N/A	510	474	474	764	880
20	N/A	1435	1434	1454	1390	1685
21	N/A	1007	1049	1056	1565	1633

Table 4: Results of experimented CP models with Gecode and Chuffed.