CDMO: Multiple Couriers Problem

Daniele Santini - daniele.santini2@studio.unibo.it
Muhammad Saleem Ghulam - muhammad.ghulam@studio.unibo.it
Master's Degree in Artificial Intelligence, University of Bologna

1 Introduction

We approach the Multiple Courier Problem (MCP) using three different methods: Constraint Programming (CP), Satisfiability Modulo Theories (SMT) and Mixed Integer Programming (MIP). For each method, we explain how the decision variables are defined and what constraints are implemented. Next, we define the problem, the objective function, and its lower_bound and upper_bound.

1.1 Problem definition

The MCP consists of delivering n items using m couriers, where each courier has a maximum load capacity and each item has a specific size, so the total size of the items assigned to any courier must not exceed its capacity. All the couriers start their route from the depot, deliver the assigned items, and after the last delivery, they must return to the depot. The input instances includes information about the number of couriers, number of items, sizes of items, capacities of couriers and a distance matrix for every pair of locations, including the depot, which is used to compute the objective function.

1.2 Objective function

The objective is to minimize the maximum distance traveled by any courier. Defining lower and upper bounds for the objective function is crucial, especially the lower bound, as it reduces the search space and helps to find the solution faster. The bounds are computed as follows:

- $lower\_bound = \max_{i \in items} (dist_{depot,i} + dist_{i,depot})$, the maximum round-trip distance from depot to any item and back.
- $upper\_bound = \sum_{j=n-m+1}^{n} SD_j$, where
  $SD = \text{sorted}\left((\text{dist}_{depot,i} + \text{dist}_{i,depot} \mid i \in \{1, \ldots, n\})\right)$, as the upper_bound we consider the worst case scenario in which $m-1$ items are delivered by $m-1$ couriers, and the last courier deliver $n-m+1$ items. Furthermore the items delivered by the last couriers are those with the largest round-trip distances.

2 CP

The CP model is implemented using the MiniZinc language within the MiniZinc IDE.

2.1 Decision Variables

We used the following variables:

- The method we adopt is called the successor approach[reference], and it consists of defining a matrix $succ \in N^{m \times n+1}$, where $succ[k, i] = j$ means that courier k travels from location i to location j, with $k \in 1, ..., m$ and $i, j \in 1, ..., n+1$. If $succ[k, i] = i$, it means that courier k does not visit location i.
- The variable $tot\_dist \in N^m$, is an auxiliary array of size m used to store the total distance traveled by each courier.
- The variable *obj* is an integer variable used to store the maximum distance traveled by any courier, which is then minimized.

2.2 Constraints

The following constraints are implemented:

- $\sum_{k \in 1..\mathrm{m}} (succ_{k,i} \neq i) = 1 \; \forall i \in 1..n$, each item is delivered by exactly one courier
- $\sum_{\substack{i \in 1..n \\ succ_{k,i} \neq i}} size_i \leq capacity_k \; \forall k \in 1..m$, each courier can't exceed its maximum load capacity
- $succ_{k,depot} \neq depot \; \forall k \in 1..m$, each courier should leave the depot. From this constraint is implied that each courier deliver at least one item.
- Each courier's route must form an Hamiltonian tour, which starts and ends at the depot, and must avoid the formation of unconnected routes. To ensure this, the Minizinc global constraint `subcircuit` is applied to the *succ* variable for each courier, leveraging the successor approach. The constraint `subcircuit` covers other constraints, such as all couriers return to the depot and that each courier visits each delivery location at most once (i.e. no two locations can have the same successor).
- $tot\_dist_k = \sum_{\substack{i \in 1..n+1 \\ succ_{k,i} \neq i}} dist_{i,\, succ_{k,i}} \; \forall k \in 1..m, \quad$, distance traveled by each courier

2.3 Objective

The objective function is to minimize the maximum distance traveled by any courier, minimize $obj = max_{k \in 1..m}(tot\_dist_k)$. The objective function is constrained between *lower_bound* and *upper_bound*.

2.4 Experiment and results

We evaluate the solving capabilities of two CP models, one without any search strategy and another with a search strategy.We apply *int_search()* to the *succ* variable, using the *dom_w_deg* heuristic for variable selection and *indomain_median* for value selection, that chooses the median value from the domain instead of the minimum. Additionally, we use **Luby restart** strategy triggered after 70 failures, along with **Large Neighborhood Search (LNS)** with a 30% relaxation, allowing for better exploration of the search space, especially in larger instances. Each model is run with two solvers: Gecode and Chuffed. A timeout condition is set to 300 seconds.

2.4.1 Results

The table 2.4.1 shows the results of all three variants evaluated on the 21 instances.

Instance	gecode	chuffed	gecode (with LNS)
1	14	14	14
2	226	226	226
3	12	12	18
4	220	220	220
5	206	206	206
6	322	322	322
7	183	167	167
8	186	186	186
9	436	436	436
10	244	244	-
11	743	-	394
12	481	-	346
13	-	-	492
14	1085	-	597
15	1113	-	598
16	308	-	286
17	1510	-	-
18	1016	-	503
19	455	-	334
20	-	-	925
21	908	-	407

Table 1: CP results

3 SMT

In this section, we use SMT models to solve MCP. SMT extends propositional logic (SAT) with additional theories such as arithmetic, arrays, and others, allowing the modeling of complex problems that involve both logical and numeric reasoning. For the MCP, we adopt a mixed approach by combining SAT with Linear Integer Arithmetic (LIA), using boolean variables for item assignment and integer variables to model route order and compute distances. We implement two SMT models to solve the MCP:

- Boolean Model: uses boolean variable to represent whether a courier delivers an item, combined with integer variables for delivery order and distances.
- Integer Model: uses only integer variables to directly assign items to couriers and also for routing and distances.

3.1 Boolean model

3.1.1 Decision variables

- $A \in \{0,1\}^{m \times n}$, boolean matrix where $A_{k,i} = 1$ means that a courier k delivers item i(otherwise 0);
- $pos \in N^{m \times n}$, integer matrix representing the delivery position of an item in the courier's route;
- $num\_assigned \in N^m$, integer array that stores the number of items assigned to each courier;
- $distance \in N^m$, integer array that stores the distance traveled by each courier.
- $objective \in N$, integer variable to store the maximum distance traveled by any courier.

3.1.2 Constraints

- $\sum_{k=1}^{m} A_{k,i} = 1 \; \forall i \in 1..n$, each item must be delivered by exactly one courier;
- $\{A_{k,1} \vee ... \vee A_{k,n}\} = True \; \forall k \in 1..m$, each courier must deliver at least one item;
- $\sum_{i=1}^{n} A_{k,i} \times size_i \leq capacity_k \; \forall k \in 1..m$, capacity constraint;
- $num\_assigned_k = \sum_{i=1}^{n} A_{k,i} \; \forall k \in 1..m$, counts the delivered items by the courier k, then these variables are used to constrain the upper_bound of the pos_k variable.

- $A_{k,i} \implies 1 <= pos_{k,i} <= num\_assigned_k \, \forall i \in 1..n, k \in 1..m$, if an item i is assigned to courier k, then the position of i in the route of k should be between a value between 1 and the total number of assigned items to k;
- $\neg A_{k,i} \implies pos_{k,i} < 0 \, \forall i \in 1..n, k \in 1..m$, if an item i is not in the route of courier k, then $pos_{k,i}$ should be negative;
- $Distinct(\{pos_{k,i} | i \in 1..n\}) \, \forall k \in 1..m$, this constraint ensures that for each courier k two delivered items cannot occupy the same position in the courier's route. To exploit this constraint, the position values of undelivered items should be either negative or greater than the number of items assigned to that courier.
- The distance is computed using the *pos* variable, since it contains the delivery route for each courier. It's composed of three parts: distance from depot to the first delivered item, sum of distances from the first to the last items and the distance from the last delivered item to the depot.

  $\forall k \in 1..m$,

$$depot\_first\_dist_k = \sum_{i=1}^{n} (A_{k,i} \wedge (pos_{k,i} = 1)) \times dist_{n,i}$$

$$between\_dist_k = \sum_{i=1}^{n} A_{k,i} \times \sum_{\substack{j=1 \\ j \neq i}}^{n} (A_{k,j} \wedge (pos_{k,j} = pos_{k,i} + 1)) \times dist_{i,j}$$

$$last\_depot\_dist_k = \sum_{i=1}^{n} (A_{k,i} \wedge (pos_{k,i} = num\_assigned_k)) \times dist_{i,n}$$

$$distance_k = depot\_first\_dist_k \; + between\_dist_k + last\_depot\_dist_k$$

3.1.3 Objective

The objective function is to minimize the maximum distance traveled by any courier, minimize $obj \geq distance_k \quad \forall k \in 1..m$. The objective is constrained between *lower_bound* and *upper_bound*.

3.2 Integer model

This model differs from the previous one only in the item assignment decision variable, where the boolean matrix is replaced by an integer array of size n, where each element takes a value in the range 1..m.

- $assign \in N^n$, if $assign_i = k$ means that the item i is delivered by the courier k.
- This variable already encodes that each item must be delivered by exactly one courier.
- All the other variables remain the same as in the previous model. The constraints are essentially the same, with the main difference being that we use the boolean expression $assign_i = k$ instead of directly accessing boolean matrix, to capture the same logic.
- Also the distance computation and the objective function remain defined in the same way as the boolean model.

3.3 Experiment and results

The SMT models were implemented using the Z3 solver in Python. A timeout condition of 300 seconds is passed to the library, though for complex instances sometimes the solver continues to run for a while even after the timeout is reached. To prevent this we configured the SIGALARM system signal to fire it after 300 seconds and handled it interrupting computation. The choice of using a second model was essentially to compare if there could be any searching improvement using a different type of decision variable. The choice to implement a second model was essentially to investigate whether using a different type of decision variable could lead to any improvements in the solution search.

3.3.1 Results

The table 3.3.1 shows the results of both models evaluated on the 21 instances.

4 MIP

In this section, we use the MIP approach to solve the MCP. The solution is inspired from this paper [1].

4.1 Decision variables

- $x \in \{0,1\}^{m\times(n+1)\times(n+1)}$, it's a 3-dimensional binary variable, if $x_{k,i,j} = 1$ means that a courier k travels directly from location i to location j, 0 otherwise. This variable keeps track of the movements of a courier through all the locations, including the depot;
- $o \in N^{m\times n}$, this variable is used to prevent the sub-tours using Miller-Tucker-Zemlin formulation(MTZ). Specifically, if a $x_{k,i,j} = 1$, $i \neq j$, then the courier k travels from location i to location j, so enforcing $o_{k,j} = o_{k,i} + 1$, this ensures that the courier cannot travel back from j to i, preventing

Instance	Boolean	Integer
1	14	14
2	226	226
3	12	12
4	220	220
5	206	206
6	322	322
7	185	168
8	186	186
9	436	436
10	244	244
11	-	-
12	-	-
13	1924	1726
14	-	-
15	-	-
16	965	1096
17	-	-
18	-	-
19	1867	-
20	-	-
21	-	-

Table 2: SMT results

subtours and unconnected routes. It is important to note that the depot is not included in this variable because the couriers must start and return to depot;

- $distance \in N^m$, array that stores the distance traveled by each courier;
- $obj \in N$: integer variable to store the maximum distance traveled by any courier.

4.2 Constraints

- $\sum_{k=1}^{m} \sum_{\substack{i=1 \\ i \neq j}}^{n+1} x_{k,i,j} = 1 \quad \forall j \in 1..n$, ensures that each item j is delivered by exactly one courier k, and i refers to all the locations (including the depot);
- $\sum_{j=1}^{n} x_{k,depot,j} = 1 \quad \forall k \in 1..m$, ensures that each courier leaves the depot exactly once;
- $\sum_{i=1}^{n} x_{k,i,depot} = 1 \quad \forall k \in 1..m$, ensures that each courier returns to depot exactly once;

- $x_{k,i,i} = 0 \quad \forall k \in 1..m, \forall i \in 1..n+1$, ensures a courier k doesn't loop on the same node, it is not strictly necessary but it can help avoiding not necessary assignments.
- $\sum_{i=1}^{n} \sum_{\substack{j=1 \\ i \neq j}}^{n+1} x_{k,i,j} \times size_i \leq capacity_k \quad \forall k \in 1..m$, each courier k can't load over the maximum capacity;
- $\sum_{j=1}^{n+1} x_{k,i,j} = sum_{j=1}^{n+1} x_{k,j,i} \quad \forall k \in 1..m$, this is balanced_flow constraint, which ensures that the number of times a courier exits the node i must be equal to the number of times the courier enters the node i;
- $o_{k,j} - o_{k,i} \geq 1 - (1 - x_{k,i,j}) \times M \quad \forall k \in 1..m, \ \forall i, j \in 1..n, \ i \neq j$, this constraint prevents the formation of sub-tours. It's implemented in MIP using the Big-M notation, where M is a large number(we set $M = 2 \times n$). The idea is that when $x_{kij} = 1$, the courier k travels from location i to location j, then the constraint $o_{k,j} - o_{k,j} \geq 1$ ensures the formation of a connected and a valid tour. Otherwise the constraint is relaxed due to the large value of M.
- $distance_k = \sum_{i=1}^{n+1} \sum_{\substack{j=1 \\ j \neq i}}^{n+1} x_{kij} \times dist_{ij} \quad \forall k \in 1..m$, stores the distance traveled by each courier k.

4.3 Objective

The objective function is to minimize the maximum distance traveled by any courier, minimize $obj \geq distance_k \quad \forall k \in 1..m$. The objective is constrained between lower bound and upper bound.

4.4 Experiment and results

We used the Pyomo optimization framework in Python to implement the model with the variables and constraints defined above. Pyomo is a solver-independent modeling language, that allows to solve the model with different solvers without changing the model. We evaluated the model for GLPK solver, CBC solver and Highs solver. For all the solvers, the timeout is fixed to 300 seconds.

4.4.1 Results

The table 4.4.1 shows the results of all three solvers evaluated on the 21 instances.

References

[1] Yoav Kaempfer and Lior Wolf. Learning the multiple traveling salesmen problem with permutation invariant pooling networks, 2019.

Instance	GLPK	CBC	HiGHS
1	14	14	14
2	226	226	226
3	12	12	12
4	220	220	220
5	206	206	206
6	322	322	322
7	234	-	167
8	186	186	186
9	436	436	436
10	244	244	244
11	-	-	-
12	-	-	-
13	-	-	646
14	-	-	-
15	-	-	-
16	-	-	362
17	-	-	-
18	-	-	-
19	-	-	-
20	-	-	-
21	-	-	-

Table 3: MIP results