SATurn

Alessandro Gentili, alessandro.gentili3@studio.unibo.it
Lorenzo D'Ascenzo, lorenzo.dascenzo@studio.unibo.it

July 2024

1 Introduction

The Multi-Courier Problem (MCP) seeks to optimize the assignment of items to couriers and the corresponding routes, aiming to reduce the maximum distance covered by any couriers. Four distinct models were implemented to tackle this problem, utilizing different approaches:

- Constraint Programming (CP)
- Boolean Satisfiability (SAT)
- Satisfiability Modulo Theories (SMT)
- Mixed-Integer Programming (MIP)

1.1 Standard practices and procedure

The input parameters are:

- m: the number of couriers
- n: the number of items
- l: the list, of size m, containing the load capacity for each courier
- s: the list, of size n, containing the size of each item
- D: the distance matrix, of size $(n+1) \times (n+1)$, containing the distances between the distribution spots. The last row and column refer to the depot.

We have to take a couple of things into consideration in order to properly model the problem:

- $n \geq m$, i.e., the number of items is greater than or equal to the number of couriers.

- The distance matrix D satisfies the triangular inequality, so

$$D_{ij} \leq D_{ik} + D_{kj} \quad \forall i, j, k \in \{1, \ldots, n+1\}$$

From this we can deduce:

- Each courier must deliver at least one item.
- Each courier must visit all and only the distribution points corresponding to the items she is carrying.

Each courier starts from the depot, that we consider as node $n+1$, and return to the depot after she delivered all the items. The variable we want to minimize is the maximum distance travelled by any courier, that is a non negative integer number.

2 CP Model

2.1 Decision variables

Our model uses the following decision variables:

- $x_{ij} \in \{0, 1\}$ for $i = 1, \ldots, m$ and $j = 1, \ldots, n$, where $x_{ij} = 1$ if and only if courier i is assigned item j.
- $y_{it} \in \{1, \ldots, n+1\}$ for $i = 1, \ldots, m$ and $t = 1, \ldots, n+2$, where $y_{it} = j$ means courier i visits node j at time t.
- $distance_i \in \mathbb{Z}$ for $i = 1, \ldots, m$, representing the total distance travelled by courier i.
- $u_{ij} \in \{0, \ldots, n\}$ for $i = 1, \ldots, m$ and $j = 1, \ldots, n+1$, where u_{ij} represents the position of node j in the tour of courier i. This variable is used for subtour elimination.

2.2 Objective function

The objective is to minimize the maximum distance travelled by any courier. This is represented by the variable $max\_distance$, which is defined as:

$$max\_distance = \max_{i=1,\ldots,m} distance_i$$

This objective aligns with the goal stated in Section 1 of achieving a fair division among drivers by minimizing the maximum distance travelled by any courier.

2.3 Constraints

The main constraints of our model are:

1. Load capacity: For each courier i, the total size of assigned items must not exceed the courier's capacity:

$$\sum_{j=1}^{n} x_{ij} \cdot s_j \leq l_i \quad \forall i = 1, \ldots, m$$

2. Each courier must deliver at least one item:

$$\sum_{j=1}^{n} x_{ij} > 0 \quad \forall i = 1, \ldots, m$$

3. Each item must be assigned to exactly one courier:

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \forall j = 1, \ldots, n$$

4. Order the loading of items with respect to the load capacity of each courier:

$$l_{k_1} \leq l_{k_2} \rightarrow \sum_{i=1}^{n} x_{k_1,i} \cdot s_i \leq \sum_{i=1}^{n} x_{k_2,i} \cdot s_i \quad \forall k_1, k_2 = 1, \ldots, m, k_1 \neq k_2$$

5. Each courier's tour must start and end at the origin:

$$y_{i1} = n + 1 \wedge y_{i,n+2} = n + 1 \quad \forall i = 1, \ldots, m$$

6. A courier visits a node if and only if she is carrying the corresponding item (here we use the assumption on triangular inequality):

$$x_{ij} = 1 \leftrightarrow \exists t \in \{2, \ldots, n + 1\} : y_{it} = j \quad \forall i = 1, \ldots, m, \forall j = 1, \ldots, n$$

7. Once a courier returns to the origin, she stays there:

$$y_{it} = n + 1 \rightarrow y_{i,t+1} = n + 1 \quad \forall i = 1, \ldots, m, \forall t = 2, \ldots, n + 1$$

8. Each node is visited by at most one courier:

$$\sum_{i=1}^{m} \sum_{t=2}^{n+1} (y_{it} = j) \leq 1 \quad \forall j = 1, \ldots, n$$

9. Distance calculation:

$$distance_i = \sum_{t=1}^{n+1} D_{y_{it}, y_{i,t+1}} \quad \forall i = 1, \ldots, m$$

10. Sub-tour elimination constraints:

    (a) The origin is always the first node in each tour:

    $$u_{i,o} = 0 \quad \forall i = 1, \ldots, m$$

    (b) If an item is assigned to a courier, its position in the tour is between 1 and n:

    $$x_{ij} = 1 \rightarrow 1 \leq u_{ij} \leq n \quad \forall i = 1, \ldots, m, \forall j = 1, \ldots, n$$

    (c) Different items assigned to the same courier must be delivered at different positions:

    $$x_{ij} = 1 \land x_{ik} = 1 \rightarrow u_{ij} \neq u_{ik} \quad \forall i = 1, \ldots, m, \forall j, k = 1, \ldots, n, j \neq k$$

    (d) Ordering of deliveries (Miller-Tucker-Zemlin constraint):

    $$x_{ij} = 1 \land x_{ik} = 1 \rightarrow u_{ij} - u_{ik} + n \cdot (y_{i,u_{ij}+1} \neq k) \geq 1$$

    $$\forall i = 1, \ldots, m, \forall j, k = 1, \ldots, n, j \neq k$$

These sub-tour elimination constraints ensure that the solution contains valid tours for each courier. The last constraint, in particular, is a variant of the Miller-Tucker-Zemlin formulation, which enforces the correct ordering of nodes in each tour and eliminates sub-tours.

Implied constraints From constraint number 8 we can deduce this constraint, stating that each courier cannot visit an intermediate node more than once:

$$\sum_{t=2}^{n+1} (y_{it} = j) \leq 1 \quad \forall m = 1, \ldots, m \quad \forall j = 1, \ldots, n$$

By including this constraint, we're providing additional structure to the problem that can help the solver find good solutions more quickly.

Symmetry breaking constraints We observe symmetry when multiple couriers have the same capacity. To break this symmetry, we impose a lexicographic ordering on the assignment variables for couriers with equal capacity:

$$l_i = l_{i_{eq}} \rightarrow [x_{i1}, \ldots, x_{in}] \geq_{lex} [x_{i_{eq},1}, \ldots, x_{i_{eq},n}]$$

$$\forall i = 1, \ldots, m-1, \forall i_{eq} = i+1, \ldots, m$$

This constraint reduces the search space by eliminating equivalent solutions that differ only in the order of couriers with the same capacity.

2.4 Validation

Experimental design Our experimental setup was as follows:

- Solvers: We used Gecode only.
- Hardware: All experiments were run on a machine with Intel core i3, 8 GB RAM, running Windows 11.
- Software: MiniZinc 2.8.3 was used for all experiments.
- Time limit: A time limit of 300 seconds (5 minutes) was set for each instance.
- models: We employed four different models, presented in order of complexity:
  1. model0: without symmetry breaking constraint
  2. model1: with symmetry breaking constraint
  3. model2: with sub-tour elimination constraint
  4. model3: with search strategy seq_search([int_search(y, first_fail, indomain_random), restart_linear(n*n)])
- Instances: We tested our model on 21 instances of varying sizes and complexities.

Experimental results Table 1 presents the best objective values obtained for each instance using different solvers and model variants.

The results show a clear trend of improvement from model0 to model3. Specifically:

- there is an important improvement from model0 to model1, especially if we look to the runtime.
- there is a slight improvement from model1 to model2, actually only on instance 7. The runtime for model2 grows because of the complexity of the model, but can't find better solutions in general. This means that the sub-tour elimination constraints do not reduce the search space.
- the search strategy implemented in model3 seems not so effective.
- Instances above 10 are hard for all model variants.

It's worth noting that while model2, and its variants with the search strategy, generally performs best, the improvements come at the cost of increased complexity and potentially longer solving times for some instances.

Table 1: Best objective values for different model variants, optimal values in bold.

	model0		model1		model2		model3	
Instance	obj	runtime	obj	runtime	obj	runtime	obj	runtime
1	**14**	0	**14**	0	**14**	0	**14**	0
2	**226**	158	**226**	0	**226**	1	**226**	3
3	**12**	0	**12**	0	**12**	0	**12**	0
4	**220**	285	**220**	2	**220**	5	**220**	300
5	**206**	0	**206**	0	**206**	0	**206**	0
6	**322**	1	**322**	0	**322**	0	**322**	28
7	182	300	220	300	200	300	**167**	300
8	**186**	300	**186**	6	**186**	29	**186**	162
9	436	300	436	300	436	300	-	300
10	244	300	244	300	244	300	-	300
11	-	300	-	300	-	300	-	300
12	-	300	-	300	-	300	-	300
13	-	300	-	300	-	300	1578	300
14	-	300	-	300	-	300	-	300
15	-	300	-	300	-	300	-	300
16	-	300	-	300	-	300	-	300
17	-	300	-	300	-	300	-	300
18	-	300	-	300	-	300	-	300
19	-	300	-	300	-	300	-	300
20	-	300	-	300	-	300	-	300
21	-	300	-	300	-	300	-	300

3 SAT Model

The model uses three different encodings for the at-least-one, at-most-one, and exactly-one cardinality constraints:

- Sequential encoding: Uses additional variables to enforce cardinality constraints. It uses a chain of implication to enforce cardinality constraints.
- Heule encoding: Uses additional variables to enforce cardinality constraints. It commonly uses less clauses than sequential encoding, it is tipically more efficient on larger datasets.
- Bitwise encoding: Uses a binary representation to enforce cardinality constraints.

3.1 Decision variables

The SAT model uses the following decision variables:

- $y_{k,i,j}$: Boolean variable indicating whether courier k travels from node i to node j

- $u_{i,b}$: Boolean variable used for sub-tour elimination, representing the binary encoding of the position of node i. This is shared among all couriers because the respective tours do not overlap (see section 3.3).

3.2 Objective function

The objective is to minimize the maximum distance traveled by any courier. In SAT, optimization is typically achieved through iterative solving and adding constraints. The code uses a loop that continues solving the problem while adding constraints to block previously found solutions. It keeps track of the best solution found so far (minimizing the maximum distance travelled by any couriers) and stops when no improvement is found after a certain number of tries (optimal solution) or when a time limit is reached (not optimal solution).

3.3 Constraints

The SAT model includes the following constraints:

1. For each courier k, the sum of sizes of items delivered must not exceed the courier's capacity:

$$\sum_{i=0}^{n} \sum_{j=0}^{n-1} y_{k,i,j} \cdot s_j \leq l_k \quad \forall k \in \{0, \ldots, m-1\}$$

2. Each courier must start and end their route at the depot (node n):

$$\sum_{j=0}^{n-1} y_{k,n,j} = 1 \quad \forall k \in \{0, \ldots, m-1\}$$
$$\sum_{i=0}^{n-1} y_{k,i,n} = 1 \quad \forall k \in \{0, \ldots, m-1\}$$

3. Each item must be delivered exactly once:

$$\sum_{k=0}^{m-1} \sum_{j=0}^{n} y_{k,j,i} = 1 \quad \forall i \in \{0, \ldots, n-1\}$$
$$\sum_{k=0}^{m-1} \sum_{j=0}^{n} y_{k,i,j} = 1 \quad \forall i \in \{0, \ldots, n-1\}$$

4. For each courier and each node (except the depot), the number of incoming edges must equal the number of outgoing edges:

$$(\sum_{j=0}^{n} y_{k,i,j} = 1) \rightarrow (\sum_{j=0}^{n} y_{k,j,i} = 1) \quad \forall k \in \{0, \ldots, m-1\}, \forall i \in \{0, \ldots, n-1\}$$

5. A courier cannot stay in the same node:

$$y_{k,i,i} = 0 \quad \forall k \in \{0, \ldots, m-1\}, \forall i \in \{0, \ldots, n\}$$

6. Subtour elimination: The model uses a binary encoding of node positions to eliminate subtours:

   - The depot (node n) is assigned to position 0.
   - For non-depot nodes, positions are constrained to be between 1 and n.
   - If courier k travels from node i to node j, the position of j must be greater than the position of i.

   This is implemented using boolean logic constraints on the $u_{k,i,b}$ variables.

Symmetry breaking While not implemented in the provided code, symmetry breaking constraints could be added to reduce the search space. For example, lexicographic ordering could be used to break symmetries between couriers with the same capacity.

Solution blocking After finding a solution, a constraint is added to block this solution, forcing the solver to find a different solution in the next iteration:

$$\bigvee_{k,i,j} (y_{k,i,j} \neq y^*_{k,i,j})$$

where $y^*_{k,i,j}$ represent the values in the current solution. This iterative process continues until an optimal solution is found or the time limit is reached or the maximum number of tries is reached.

3.4 Validation

Unfortunately, we were not able to generate results for SAT model due to some problems with code.

4 SMT Model

For the SMT model we used the same structure of the CP model, the difference here is the implementation: we used the z3 library and the theory of arrays, meaning that all the decision variables are array of integers. A basic theory of arrays is characterized by the select-store axioms. The expression select(a, i) returns the value stored at position i of the array a; and store(a, i, v) returns a new array identical to a, but at position i it contains the value v.

4.1 Decision variables

See section 2.1.

4.2 Objective function

See Section 2.2.

4.3 Constraints

See section 2.3.

4.4 Validation

Experimental design Our experimental setup was as follows:

- Solvers: We used the standard Optimizer provided by z3 library.
- Hardware: All experiments were run on a machine with Intel core i3, 8 GB RAM, running Windows 11.
- Software: z3 4.13.0 was used for all experiments.
- Time limit: A time limit of 300 seconds (5 minutes) was set for each instance.
- models: We employed three different models, presented in order of complexity:
  1. model0: without symmetry breaking constraint
  2. model1: with symmetry breaking constraint
  3. model2: with sub-tour elimination constraint
- Instances: We tested our model on 21 instances of varying sizes and complexities.

experimental results

The results show a clear trend of improvement from model0 to model3. Specifically:

- there is an important improvement from model0 to model1, especially if we look to the runtime.
- there is a slight improvement from model1 to model2. The runtime for model2 grows because of the complexity of the model, but in general it can't find better solutions.
- Instances above 9 are hard for all model variants.

It's worth noting that the improvements come at the cost of increased complexity and potentially longer solving times for some instances.

Table 2: Best objective values for different model variants, optimal values in bold.

	model0		model1		model2	
Instance	obj	runtime	obj	runtime	obj	runtime
1	**14**	0	**14**	0	**14**	0
2	**226**	42	**226**	9	**226**	15
3	**12**	0	**12**	0	**12**	0
4	**220**	48	**220**	7	**220**	14
5	**206**	0	**206**	0	**206**	0
6	**322**	7	**322**	1	**322**	1
7	182	300	220	300	200	300
8	**186**	72	**186**	13	**186**	46
9	-	300	**436**	166	**436**	267
10	-	300	-	300	-	300
11	-	300	-	300	-	300
12	-	300	-	300	-	300
13	-	300	-	300	-	300
14	-	300	-	300	-	300
15	-	300	-	300	-	300
16	-	300	-	300	-	300
17	-	300	-	300	-	300
18	-	300	-	300	-	300
19	-	300	-	300	-	300
20	-	300	-	300	-	300
21	-	300	-	300	-	300

5 MIP Model

In this chapter, we solve the Multi-Courier Problem (MCP) using the Gurobi optimizer. The model is constructed with several decision variables that represent the actions and states of the couriers and the routes they take.

5.1 Decision Variables

- **Binary Variables:**
  - x_{ijk}:Indicates whether courier k travels directly from node i to node$j$.

$$x_{ijk} = \begin{cases} 1 & \text{if courier } k \text{ travels from node } i \text{ to node } j \\ 0 & \text{otherwise} \end{cases}$$

  - y_{ik}: Indicates whether courier k visits customer i.

$$y_{ik} = \begin{cases} 1 & \text{if courier } k \text{ visits customer } i \\ 0 & \text{otherwise} \end{cases}$$

- **Continuous Variables:**
  - u_{ik}: Used to eliminate subtours, representing the position of customer i in the route of courier k.
- **Objective Variable:**
  - *max_distance*: Represents the maximum distance traveled by any single courier. This is a key variable as the objective of the model is to minimize this value.

5.2 Objective Function

The primary goal of the MCP model in this experiment is to minimize the maximum distance traveled by any courier. This is represented by the variable *max_distance*.

$$\text{Objective:} \quad \min \max_k \left(\sum_{i \in \text{Deposit}} \sum_{j \in \text{Deposit}} x_{ijk} \cdot D_{ij} \right)$$

Where:

- D_{ij} denotes the distance between nodes i and j.
- x_{ijk} indicates whether a route between nodes i and j is traveled by courier k.

By minimizing the maximum distance, the objective function ensures a balanced load among couriers and promotes efficiency in the overall routing system.

5.3 Constraints

Several constraints are imposed to ensure that the model adheres to the practical requirements of the MCP. These constraints are critical to obtaining a feasible and optimal solution:

- Maximum Distance Constraint

$$\sum_{i \in \text{Deposit}} \sum_{j \in \text{Deposit}} x_{ijk} \cdot D_{ij} \leq \text{max\_distance} \quad \forall k$$

  Ensures that the distance traveled by each courier does not exceed the maximum allowable distance.

- Customer Visit Constraint

$$\sum_{k \in \text{ListCouriers}} y_{ik} = 1 \quad \forall i \in \text{ListCustomers}$$

$$\sum_{j \in \text{Deposit}} x_{ijk} = 1 \quad \forall i \in \text{ListCustomers}, \forall k \in \text{ListCouriers}$$

Guarantees that each customer is visited exactly once by one courier.

- Depot Constraints

$$\sum_{j \in \text{ListCustomers}} x_{0jk} = 1 \quad \forall k \in \text{ListCouriers}$$

$$\sum_{i \in \text{ListCustomers}} x_{i0k} = 1 \quad \forall k \in \text{ListCouriers}$$

Ensures that each courier starts from and returns to the depot (node 0).

- Capacity Constraints

$$\sum_{i \in \text{ListCustomers}} y_{ik} \cdot s_i \leq l_k \quad \forall k$$

Ensures that the load carried by each courier does not exceed its capacity.

- Flow Conservation Constraint

$$\sum_{j \in \text{Deposit}} x_{ijk} = \sum_{j \in \text{Deposit}} x_{jik} \quad \forall i \in \text{ListCustomers}, \forall k$$

Ensures that the number of vehicles entering and leaving each customer node is balanced, which maintains the continuity of the route.

- Subtour Elimination Constraints

$$u_{ik} - u_{jk} + n \cdot x_{ijk} \leq n - 1 \quad \forall i, j \in \text{ListCustomers}, i \neq j$$

Prevents the formation of subtours which are not feasible solutions in the MCP context.

5.4 Validation

The validation of the model was carried out through computational experiments on a series of predefined instances.

1. **Instance Setup**: Each instance is set up with its specific parameters.
2. **Model Solving**: The model is solved using Gurobi with a time limit set to ensure practical feasibility.

3. **Solution Evaluation**: The solutions are evaluated for optimality and feasibility, focusing on the maximum distance traveled by couriers and adherence to constraints.

Result Analysis:

- **Optimal Solutions**: The model successfully found optimal solutions for the majority of instances, with optimal routes and load distributions.(See Table 3)
- **JSON Output**: For each instance, a JSON file was created to store the results, including the time taken, the objective value, and whether the solution was optimal.

The successful validation across various instances demonstrates the robustness and efficiency of the model in solving the MCP, adhering to practical constraints, and optimizing courier routes.

Table 3: Best objective values for different model variants, optimal values in bold.

Instance	obj	runtime
1	**14**	0
2	**226**	0
3	**12**	0
4	**220**	0
5	**206**	0
6	**322**	0
7	**167**	11
8	**186**	0
9	**436**	1
10	**244**	30

6 Conclusions

The Multi-Courier Problem (MCP) is a complex optimization problem aiming to minimize the maximum distance traveled by any courier while ensuring all items are delivered efficiently. This report explored four distinct modeling approaches to solve MCP: Constraint Programming (CP), Boolean Satisfiability (SAT), Satisfiability Modulo Theories (SMT), and Mixed-Integer Programming (MIP). Each method leveraged different mathematical and computational techniques to address the problem's constraints and objectives.

- **CP Model**: The CP models, built using MiniZinc and runned on the python interface, are able to solve the MCP using different constraints. The results we found are dependent on such constraints and tell us something interesting, sometimes unexpected, on them.

- **SAT Model**:
- **SMT Model**: The SMT models are builded on the way of the CP ones. Here the difficulties was the coding part. We defined those models using z3 library and the theory of arrays only. The results generates are similar to the CP's results, as expected.
- **MIP Model**: Utilizing the Gurobi optimizer, the MIP model effectively minimized the maximum distance traveled by any courier. The model's structure ensured adherence to all MCP constraints, including load capacity, depot constraints, and sub-tour elimination.

  The MIP model exhibited robust performance, successfully finding optimal solutions for the instances from 1 to 10