Multiple Courier Planning Problem

Anthea Silvia Sasdelli anthea.sasdelli@studio.unibo.it
Gabriele Fossi gabriele.fossi@studio.unibo.it
Xiaofeng Zhang xiaofeng.zhang@studio.unibo.it

July 7, 2024

1 Introduction

In recent years, especially following the COVID-19 pandemic, the frequency of online transactions for goods, food, clothes, and documents has surged. Consequently, companies now face the critical challenge of efficiently scheduling the dispatching of items through multiple couriers to minimize transportation costs. This report addresses the Multiple Couriers Planning (MCP) problem, which involves assigning items and routes to couriers in an optimal manner.

The MCP problem that we had to solve can be formally defined as follows: Given m couriers and $n \geq m$ items to be delivered to various customer locations, each courier i has a maximum load capacity l_i, and each item j has a specific size s_j and distribution point j. The objective is to assign items to couriers and determine a sequence of delivery points (a tour) for each courier, starting and ending at a designated origin point o. The assignment must respect the load capacity l_i of each courier and aim to minimize the maximum distance traveled by any courier to ensure a fair division of labor.

1.1 Formalization of the problem

In order to tackle this task, different approaches has been considered. They are based on CP, SMT and MIP. The models have been formulated in different ways, each with its own variables and constraints. However, all the models share the same parameters:

- **couriers** = 1..m: set of integers representing available couriers.
- **items** = 1..n: set of integers representing items to be delivered.
- **nodes** = 1..n+1: set of integers representing nodes in the graph, including origin and destination points.

Moreover, there is also a variable in common: X.

- $X[k, i, j]$: It is a boolean variable that is equal to 1 if courier k goes from location i to location j. The locations i and j corresponds to the object locations $(1, ..., n)$ plus the origin (indicated as $n + 1$). Otherwise it is equal to 0. This formulation facilitates the application of constraints related to capacity and route continuity.

Some constraints in the different models have the same rationale, however they will be discussed in their respective sections since their formulation differs according to the methodology employed.

The objective function to minimize is the maximum distance traveled by any courier. In order to define it, first we need to define the courier distance:

$$courier\_distance[k] = \sum_{i,j} D[i, j] \cdot X[k, i, j] \quad k = 1, ..., m \quad i, j = 1, ..., n + 1$$

Where D is the distance matrix.
Then the objective function is defined as the maximum among the couriers' distances:

$$max(courier\_distance[k]) \quad k = 1, ..., m$$

Regarding the upper and lower bounds, different ones were considered and tried. and they will be explained in the following sections.

1.2 Subdivision of the work

Since our group is formed by three members, we decided to split the three different methods among us. After implementing base models we discussed them together, in order to share and reason about the constraints and ideas that each one had implemented, as well as possible suggestions on what to try next. After that, we revised the work together and help each other fix possible errors.

In order to complete the project it took around one month. The difficulties were mostly related to the scarceness of available documentation in the internet on how to code in some languages, such as AMPL.

2 CP Model

This chapter introduces the first phase of solving the MCP problem using Constraint Programming (CP). MiniZinc was employed as the modeling language for this approach.

2.1 Decision variables

To address the Multiple Couriers Planning (MCP) problem effectively, we define the following decision variables:

- The $X[k, i, j]$ variable defined in Section 1.1.
- **visited[k,i]**: a binary two-dimensional array that indicates whether courier k visits point i. Formally:

$$\text{visited}[k,i] = \begin{cases} 1 & \text{if courier } k \text{ visits point } i \\ 0 & \text{otherwise} \end{cases}$$

  This variable is used to keep track of the nodes visited by each courier. It helps to ensure that all required nodes are visited and that no node is visited by more couriers than necessary. Furthermore, it helps us in implementing constraints that ensure each node is visited exactly once and in calculating the total distances traveled by each courier.

2.2 Objective Function

The objective of the Multiple Couriers Planning problem is to minimize the maximum distance traveled by any courier and is defined in Section 1.1. In the CP model we chose the following lower and bounds:

- Lower bound (*lower_bound*): In order to define the lower bound we assumed that each courier carries at least one object. As a consequence to choose the lower bound we considered all the paths consisting of only one visited node and we took as lower bound the maximum of the corresponding paths:

$$lower\_bound = max(D[n+1,i] + D[i,n+1]) \; i = 1, ..., n$$

- Upper bound (*upper_bound*): On the other hand, the upper bound is the length of the path starting at the origin, visiting every node in the index order $(1, 2, ...n)$ and ending at the origin. Indeed there cannot be a reasonable longer path than this one:

$$upper\_bound = D[n+1,1] + \sum_{i=1}^{n-1} D[i,i+1] + D[n,n+1]$$

2.3 Constraints

- **Each item must be visited exactly once:**

$$\text{forall}(i \in 1..n) \left(\sum_{k \in \text{couriers}, j \in \text{nodes}} (X[k,i,j]) = 1 \right)$$

  This constraint ensures that each item node (excluding the warehouse) is visited exactly once by one courier.

- **Flow consistency for each deliveryman:**

$$\text{forall}(k \in \text{couriers}, i \in \text{nodes}) \left(\sum_{j \in \text{nodes}} (X[k,i,j]) = \text{visited}[k,i] \wedge \sum_{j \in \text{nodes}} (X[k,j,i]) = \text{visited}[k,i] \right)$$

  This constraint ensures that each courier's movements respect the visited status of nodes, maintaining flow consistency in both outgoing and incoming deliveries.

- **Deliverymen must start and end at the warehouse:**

$$\text{forall}(k \in \text{couriers}) \left(\sum_{j \in 1..n} (X[k,n+1,j]) = 1 \wedge \sum_{i \in 1..n} (X[k,i,n+1]) = 1 \right)$$

  Each courier must start from and return to the warehouse node (node 1), ensuring complete delivery cycles.

- **Load constraints for each deliveryman:**

$$\text{forall}(k \in \text{couriers}) \left(\sum_{i \in 1..n} (\text{visited}[k,i] \cdot s_j[i]) \leq l_i[k] \right)$$

  This constraint ensures that the total size of items carried by each courier does not exceed their respective load capacities l_i.

- **Symmetry breaking constraint $\rightarrow$ Subtour elimination :**

$$\text{forall}(k \in \text{couriers}, i \in 1..n, j \in 1..n \text{ where } i \neq j)\, (u[k,i] - u[k,j] + n \cdot X[k,i,j] \leq n-1)$$

  These constraints prevent the formation of subtours by ensuring that each courier's route forms a single closed loop without any unnecessary detours.

2.4 Validation

This section outlines the validation process conducted to evaluate the effectiveness of solving the Multiple Couriers Planning (MCP) problem using MiniZinc and various solvers.

2.4.1 Experimental design

The experiments for CP were structured as follows:

- **Folders and Files Used:** The `Instances` folder contains problem instances, while solutions are saved in `res/CP/`.
- **MiniZinc Models:** The main model used for solving MCP is `MCP_final.mzn` and it's variation without the upper and lower bounds.
- **Solvers Used:** MiniZinc solvers *gecode* and *chuffed* were employed with a solver time limit of 300 seconds (5 minutes) per instance.

2.4.2 Experiment Results

Table 1 shows the results obtained for each instance. The results show that there is not a significant difference between the use of objective function bounds. On the other hand, the use of Gecode solver outperforms the model with the Chuffed solver. Overall, the first ten instances are relatively easy to solve, while the second half of instances is very hard to solve.

3 SMT Model

In order to use SMT to solve the MPC problem, we tried different solutions by using Z3 library. Although the results were roughly the same, we still got some different results by changing the Constrains and implementing them in different methods.

ID	Final		No Bounds	
	Gecode	Chuffed	Gecode	Chuffed
1	**14**	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**
3	**12**	**12**	16	16
4	**220**	N/A	**220**	**220**
5	**206**	**206**	**206**	**206**
6	**322**	358	**322**	**322**
7	**167**	N/A	**167**	N/A
8	**186**	N/A	**186**	N/A
9	**436**	585	**436**	N/A
10	250	**244**	**244**	N/A
11	N/A	N/A	N/A	N/A
12	N/A	N/A	N/A	N/A
13	N/A	N/A	N/A	N/A
14	N/A	N/A	N/A	N/A
15	N/A	N/A	N/A	N/A
16	N/A	N/A	N/A	N/A
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	N/A
20	N/A	N/A	N/A	N/A
21	N/A	N/A	N/A	N/A

Table 1: Objective function values for each instance and CP model. The values in bold correspond to optimal values. When the solver is unable to find a solution within a time limit of 300 seconds it stops and the correspoding value is N/A.

3.1 Decision variables

The model we use is mainly based on the following variables, some of which are Boolean variables, which are defined based on Propositional Logic Theory. Some of them are Integer variables, which are defined based on Linear Integer Arithmetic Theory. The following are the Decision Variables used in the SMT solution.

- The X variable defined in Section 1.1.
- $\mathbf{Y[k, i]}$: A binary variable that indicates whether item i is assigned to courier k. Formally:

$$Y[k,i] = \begin{cases} 1 & \text{if item } i \text{ is assigned to courier } k \\ 0 & \text{otherwise} \end{cases}$$

- $\mathbf{U[i]}$: An integer variable that represents the position of point i in the route (used for subtour elimination). Formally:

$$U[i] \in \mathbb{Z}^+, \quad 0 < U_i \leq n$$

- $\mathbf{C[k]}$: An integer variable that represents the total distance traveled by courier k. Formally:

$$C[k] \in \mathbb{Z}^+$$

- **MaxCost**: An integer variable that represents the maximum distance traveled by any courier. Formally:

$$MaxCost = \max_k \{C[k]\}, \quad MaxCost \in \mathbb{Z}^+$$

3.2 Objective Function

The objective of the Multiple Couriers Planning problem is to minimize the maximum distance traveled by any courier. The method aims to ensure that the travel distance is evenly distributed among the couriers, minimizing the longest route any courier has to take.

$$\text{Minimize} \quad \text{MaxCost}$$

where

$$\text{MaxCost} = \max_k \{C[k]\}$$

At the same time, in order to use binary search to find the solution faster, the experiment needs to define two additional variables, **LB** and **UB**, namely the lower and upper bounds.

- The lower bound is the same one which is defined in Section 2.2 and is defined as the maximum round-trip distance from the origin to any distribution point and back to the origin.
- The upper bound, on the other hand, is a bit more loose with respect to the one in Section 2.2 and defined as the sum of all distances in the distance matrix:

$$\text{UB} = \sum_{i=0}^{n} \sum_{j=0}^{n} D[i,j]$$

  Where:

  - D is the distance matrix.
  - $D[i,j]$ represents the distance from distribution point i to distribution point j.

3.3 Constraints

- **Domains for U**: Ensure that the position variables U are within valid bounds.

$$0 < U_i \leq n \quad \forall i \in \{0,\dots,n-1\}$$

  This is a linear inequality constraint. The purpose of this constraint is to ensure that the position variablesU are within valid bounds, which helps in eliminating subtours.

- **Each item must be delivered by exactly one courier**:

$$\bigvee_{k=0}^{m-1} \bigvee_{j=0, j\neq i}^{n} X[k,i,j] \quad \forall i \in \{0,\dots,n-1\}$$

$$\bigvee_{k=0}^{m-1} \bigvee_{j=0, j\neq i}^{n} X[k,j,i] \quad \forall i \in \{0,\dots,n-1\}$$

  This is a set of disjunction constraints. The purpose of these constraints is to ensure that each item is picked up and delivered by exactly one courier.

- **At most one outgoing and incoming arc for each item and courier**:

$$\sum_{j=0, j\neq i}^{n} X[k,i,j] \leq 1 \quad \forall k \in \{0,\dots,m-1\}, \forall i \in \{0,\dots,n\}$$

$$\sum_{j=0, j\neq i}^{n} X[k,j,i] \leq 1 \quad \forall k \in \{0,\dots,m-1\}, \forall i \in \{0,\dots,n\}$$

  These are linear inequality constraints. The purpose of these constraints is to ensure that each courier has at most one outgoing and one incoming route for each item.

- **Item assignment constraints**:

$$Y[k,i] = \bigvee_{j=0,j\neq i}^{n} X[k,i,j] \quad \forall k \in \{0,\dots,m-1\}, \forall i \in \{0,\dots,n-1\}$$

$$Y[k,i] = \bigvee_{j=0,j\neq i}^{n} X[k,j,i] \quad \forall k \in \{0,\dots,m-1\}, \forall i \in \{0,\dots,n-1\}$$

$$\sum_{k=0}^{m-1} Y[k,i] = 1 \quad \forall i \in \{0,\dots,n-1\}$$

  These are a combination of disjunction and linear equality constraints. The purpose of these constraints is to ensure that items are assigned correctly to couriers based on the routes, and that each item is assigned to exactly one courier.

- **Load constraints**:

$$\sum_{i=0}^{n-1} Y[k,i] \cdot s[i] \leq l[k] \quad \forall k \in \{0,\dots,m-1\}$$

  This is a linear inequality constraint. The purpose of this constraint is to ensure that the total load assigned to each courier does not exceed their capacity.

- **Ensure couriers start and end at the origin**:

$$\sum_{j=0}^{n-1} X[k,n,j] = 1 \quad \forall k \in \{0,\dots,m-1\}$$

$$\sum_{j=0}^{n-1} X[k,j,n] = 1 \quad \forall k \in \{0,\dots,m-1\}$$

  These are linear equality constraints. The purpose of these constraints is to ensure that each courier's route starts and ends at the origin.

- **Subtour elimination constraints**:

$$U[j] > U[i] \quad \forall k \in \{0,\dots,m-1\}, \forall (i,j) \in \{0,\dots,n-1\} \text{ if } X[k,i,j] = 1$$

  This is a linear inequality constraint. The purpose of this constraint is to ensure that there are no subtours within a courier's route.

- **Symmetry-breaking constraints(This constrain is used for optimization)**:

$$\sum_{i=0}^{n-1} Y[k,i] \geq \sum_{i=0}^{n-1} Y[k+1,i] \implies C[k] \leq C[k+1] \quad \forall k \in \{0,\dots,m-2\}$$

  This is a conditional constraint. The purpose of this constraint is to break symmetries and reduce the search space by ordering the cost of couriers.

- **Cost constraints**:

$$C[k] = \sum_{i=0}^{n} \sum_{j=0}^{n} X[k,i,j] \cdot D[i,j] \quad \forall k \in \{0,\dots,m-1\}$$

$$MaxCost \geq C[k] \quad \forall k \in \{0,\dots,m-1\}$$

$$\bigvee_{k=0}^{m-1} MaxCost = C[k]$$

  These are linear equality and inequality constraints. The purpose of these constraints is to ensure the cost is calculated correctly and to set the objective function to minimize the maximum distance traveled by any courier.

3.4 Validation

3.4.1 Experimental design

In order to implement the SMT solution to the MCP problem, the experiment used the z3 library as the basic environment for building the model. After using simple constraints, the experiment found that as the constraints increase, the number of available solutions will decrease, but after using a better search solution, the number of solutions will increase. Therefore, in this SMT experiment, we used binary search to speed up the search for solutions, but after adding SB, the solutions that could have been obtained could not be obtained. This is because the number of constraints increased. Even though the purpose of adding SB is to reduce the search space, some solutions were not successfully obtained due to the increase in constraints.

- **Folders and Files Used:** The `Instances` folder contains problem instances, while solutions are saved in `res/SMT/`.
- **SMT:** The main models used for solving SMT are `Optimize_BINARY` and `Optimize_WITH_SB`.
- **Solvers Used:** SMT solvers *Optimize with symmetry breaking* and *Optimize with Binary Search* were employed with a solver time limit of 300 seconds (5 minutes) per instance.

3.4.2 Experimental results

Table 3 provides the results obtained from the experiments conducted.

ID	Optimize Without SB	Optimize
1	**14**	**14**
2	**226**	**226**
3	**12**	**12**
4	**220**	**220**
5	**206**	**206**
6	**322**	**322**
7	**167**	N/A
8	**186**	**186**
9	**436**	N/A
10	**244**	N/A
11	N/A	N/A
12	N/A	N/A
13	N/A	N/A
14	N/A	N/A
15	N/A	N/A
16	N/A	N/A
17	N/A	N/A
18	N/A	N/A
19	N/A	N/A
20	N/A	N/A
21	N/A	N/A

Table 3: Objective function values for each instance and SMT model. The values in bold correspond to optimal values. When the solver is unable to find a solution within a time limit of 300 seconds it stops and the corresponding value is N/A.

In the first ten solutions of all examples, we used two schemes, namely the results obtained by Optimize using SB constraints and without SB constraints. From the results, it can be seen that after adding SB constraints, although it can effectively reduce the search range, for some relatively large-scale problems, the required answers are still not obtained within the specified time.

4 MIP Model

The MIP model was implemented using AMPL, a solver-independent langauge that allows to solve MIP problems. In this way we are able to try different solvers without the need to rewrite the model.

4.1 Decision variables

The MIP model is based on the folloing variables:

- The X variable defined in Section 1.1.
- $succ[i]$: It is an integer variable that represents the sequence of the visited location. So for example if object i location is the third one in the route, $succ[i]$ will have a value of 3. This introduced variable is needed to determine constraints about the routes.
- $courier\_distance[k]$: It is an integer variable that represents the distance of the path traveled by courier k
- $max\_distance$: It is an integer variable that represents the maximum distance traveled by any courier: $max(courier\_distance[k])$ $k = 1, ..., m$
- $num\_objects[i]$: It is an integer variable that represents the number of objects that courier i needs to deliver. This variable is introduced in order to define the symmetry breaking constraint.

4.2 Objective Function

The objective function is the same defined in Section 1.1, while the lower and upper bounds are the same one which were employed in the CP model (Section 2.2).

4.3 Constraints

Some of the constraints that we imposed in our MIP model are already present in the other models, while other ones are specific to it. However, also the constraints that we defined in the other models are presented since in MIP we will have to redefine them such that they are linear.

- **One courier departs from one object location**: For each object location i there must be only one courier that departs from that location (to ensure that only one courier delivers the object).

  $$\sum_{k,j} X[k, i, j] = 1 \quad j = 1, ...n$$

  Where i can assume values between 1 and n (the objects) while j can assume values between 1 and $n + 1$ (the objects plus the origin).

- **One courier arrives to one object location**: Similarly, for each object location i there must be only one courier that arrives to that location (to ensure that only one courier delivers the object).

  $$\sum_{k,i} X[k, i, j] = 1 \quad k = 1, ..., n$$

  Where j can assume values between 1 and n (the objects) while i can assume values between 1 and $n + 1$ (the objects plus the origin).

- **Couriers start at origin**: Each courier path must start at the origin:

  $$\sum_{j} X[k, n + 1, j] = 1 \quad k = 1, ..., m \quad j = 1, ..., n$$

- **Couriers end at origin**: Each courier path must end at the origin:

  $$\sum_{j} X[k, i, n + 1] = 1 \quad k = 1, ..., m \quad i = 1, ..., n$$

- **Couriers cannot go from one location to itself**:

$$X[k,i,i] = 0 \quad k = 1,...,m \quad i = 1,...n$$

- **Couriers must carry at least one item**:

$$X[k,n+1,n+1] = 0 \quad k = 1,...m$$

Similar to the previous one but defined in another constraint for clarity.

- **Path consistency**: This constraint ensures that if a courier travels from location i to location j there must be a corresponding path from j back to i

$$\sum_j X[k,j,i] = \sum_j X[k,i,j] \quad k = 1,...,m \quad i = 1,...,n \quad j = 1,...n+1$$

- **Load constraint**: The load of each courier must not exceed their possible capacity.

$$\sum_{i,j} X[k,i,j] \cdot s\_values[i] \leq l\_values[k] \quad k = 1,...m \quad i = 1,...n \quad j = 1,...,n+1$$

where $s\_values[i]$ represents the size of object i while $l\_values[k]$ represents the maximum load capacity for courier k.

Then we proceed to define the constrain related to *succ*, to ensure the correct sequence of the visited locations.

- **First location**: It ensures that if a courier's route starts at the origin and travels to object k, then k must be the first object visited by the courier.

$$succ[j] \leq 1 + 2n \cdot (1 - X[k,n+1,j]) \quad k = 1,...,m \quad j = 1,...n$$

- **Successive location**: It guarantees that if a courier travels from i to j, then $succ[j]$ must be greater than $succ[i]$

$$succ[i] - succ[j] \geq 1 - 2n \cdot (1 - X[k,j,i]) \quad k = 1,...m \quad i,j = 1,...n$$

Then we have the symmetry breaking constraint:

- **Number of objects per courier**: First we need to ensure the correct computation of the number of objects carried by the courier:

$$num\_objects[k] = \sum_{i,j} X[k,i,j] \quad k = 1,...,m \quad i = 1,...n \quad j = 1,...,n+1$$

- **Symmetry Breaking Constraint**: A symmetry-breaking constraint that orders the couriers by their distance, similar to the one in section 3.3.

$$courier\_distance[k] \leq courier\_distance[k+1] + M \cdot (num\_objects[k+1] - num\_objects[k]) \quad k = 1,...,m-1$$

Where M is a sufficiently large constant to ensure that the constraint holds when there are more objects for courier $k+1$

Finally the last constraint about the maximum courier distance:

- **Maximum courier distance**: The maximum courier distance must be greater or equal to each courier distance.

$$max\_distance \geq courier\_distance[k] \; k = 1,...,m$$

4.4 Validation

4.4.1 Experimental design

The experiments were structured as follows:

- **Folders and Files Used:** The `Instances` folder contains problem instances while solutions are saved in `res/MIP`.
- **MIP models:** The base model used to solve MCP is `model.mod` while its variation with the symmetry breaking constraint is `model_SB.mod`

The MIP model was implemented using AMPL. AMPL allows the use of different solvers, however in the experiments we only considered two of them due to limited resources, namely:

- HiGHS: a high performance serial and parallel solver for large scale sparse linear optimization problems. [HH18]
- SCIP: one of the fastest academically developed solvers for MIP.

The aim is to compare the performance of the two different solvers, considering both the scenarios with and without symmetry breaking constraints. The time limit to find a solution is set to 300 seconds.

4.4.2 Experimental results

Table 4 contains the results for each instance, i.e. the value of the objective function in the 4 different cases that we tried. The results among the different combinations of solvers and symmetry breaking constraints do not vary significantly. The first ten instances are solved to optimality by all the models, except for the third instance which is not solved correctly by the solvers with the symmetry breaking constraint. On the other hand, subsequent instances are very hard to solve. The solvers are not able to find solutions, except for instance 13 which is solved by the SCIP solver and the HiGH solver with the symmetry breaking constraint and instance 16 which is solved simply by the HiGHS solver.

ID	No SB		SB	
	HiGHS	SCIP	HiGHS	SCIP
1	**14**	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**
3	**12**	**12**	N/A	N/A
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	N/A	N/A	N/A	N/A
12	N/A	N/A	N/A	N/A
13	N/A	984	904	N/A
14	N/A	N/A	N/A	N/A
15	N/A	N/A	N/A	N/A
16	579	N/A	N/A	N/A
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	N/A
20	N/A	N/A	N/A	N/A
21	N/A	N/A	N/A	N/A

Table 4: Objective function values for each instance and MIP model. The values in bold correspond to optimal values. When the solver is unable to find a solution within a time limit of 300 seconds it stops and the correspoding value is N/A.

5 Conclusion

In conclusion, while we were proceeding with the implementation of CP, SMT and MIP solutions, we encountered challenges as the problem instances grew in size and complexity. For initial instances, solutions were feasible across all three languages, showcasing their respective strengths in different contexts. However, scaling issues and computational limitations prevented us from finding solutions for the more complex instances using all three methods.
Despite the performance being similar among all the models, MIP slightly outperformed the other ones. Indeed it is able to find optimal solutions for all the first 10 instances, and some combinations of solvers together with the symmetry breaking constraint manage to solve instances 13 and 16 (even if not optimally).

References

[HH18] Q. Huangfu and J. A. J. Hall. Parallelizing the dual revised simplex method. 2018.