Multiple Couriers Planning (MCP) Project Report

Ehsan Ramezani ehsan.ramezani@studio.unibo.it
Mehregan Nazarmohsenifakori mehrega.nazarmohseni@studio.unibo.it
Fatemehzahra Ghafari Ghomi fatemezahra.ghafari@studio.unibo.it
Shafagh Rastegari shafagh.rastegari@studio.unibo.it

January 2025

1 Introduction

The Multiple Couriers Planning (MCP) problem involves assigning items to multiple couriers, each with limited capacity, while planning their routes to minimize the maximum distance traveled by any courier. This problem shares similarities with the Capacitated Vehicle Routing Problem (CVRP). the MCP focuses on achieving a fair distribution of work among couriers based on a defined objective.

Parameters: **m** (number of couriers), **n** (number of items), **l** (courier capacities), **s** (item sizes), **D** (distance matrix for travel distances).

Our project took us around two months to finish, and as a team, we discussed our ideas while each individual was responsible for a specific part: Ehsan Ramezani developed the CP modeling, Mehregan Nazarmohsenifakori performed the SAT part, Fatemehzahra Ghafari Ghomi worked on the SMT approach, and Shafagh Rastegari concentrated on the MIP part. the Docker integration and defining a general solver to work with different modules have been done by all team members.

AI-assisted tools were used for drafting parts of the report and generating utility functions for handling JSON data, managing directory paths, and ensuring consistency between modules.

1.1 Lower and Upper Bounds

Establishing tight bounds on the objective function is crucial for improving computational efficiency and pruning infeasible solutions. We calculate them as follows:

Lower Bound: The item farthest from the depot must be delivered, so there are no possibilities to avoid the distance cost.

$$\mathrm{LB} = \max_{1 \le i \le n} \left(D_{n+1,i} + D_{i,n+1} \right)$$

Upper Bound: We use a hybrid approach, selecting the minimum of two approximations:

1. **First Upper Bound:** Sort items by descending distance from the depot and assign them greedily to couriers until capacity is met. Each courier delivers its assigned items one at a time, returning to the depot after each. By the triangle inequality, this approach is an overestimate compared to a possible optimal route.

The upper bound per courier:

$$UB_i = \sum_{j \in S_i} \left[D(o, j) + D(j, o) \right]$$

The final upper bound:

$$UB = \max(UB_1, \dots, UB_m)$$

2. **Second Upper Bound:** When couriers deliver nearly equal numbers of items, the largest route length is close to optimal. The maximum travel distance is estimated as:

$$UB_2 = \sum_{k=1}^{\lceil \frac{n}{m} \rceil} \mathrm{top\_k} \left(\max(D[i]) \mid i \in \{1, \dots, n\} \right) + \mathrm{LB}$$

2 CP Model

The idea of the model that has been implemented in the CP approach to solve the MCP problem originated from the official repository of MiniZinc-benchmark [3]. The chosen model, which we refined, can be referred to as the giant tour representation. In the following sections, we will discuss our model along with the modifications that made it proper for the MCP problem and its constraints.

2.1 Decision Variables

The giant tour representation introduces artificial depots, where each courier starts at a predefined point, completes deliveries, and returns to an endpoint, with routes linked sequentially (Gokalp et al. [2]). To formalize the problem, we define:

Sets:

- NODES: Giant tour locations ($n + 2m$).

- START_NODES / END_NODES: Couriers' start ($n+1$ to $n+m$) and end nodes ($n+m+1$ to $n+2m$).
- demand: An array ($n+2m$), where $S[i]$ at delivery nodes, otherwise 0.

Decision Variables: Each is indexed over $n+2m$, defining the giant tour:

- `successor`[i]: Immediate successor of node i.
- `predecessor`[i]: Immediate predecessor of node i.
- `route_m`[i]: Courier k assigned to node i.
- `load`[i]: Total load carried before node i.
- `final_dist`[i]: Distance traveled by the courier up to node i.

2.2 Objective Function

The objective function is to minimize *obj* where *obj* = max(final_dist), as we aim to minimize the maximum distance traveled by any courier to ensure workload fairness. The bounds of the *obj* variable (lower_bound $\leq$ *obj* $\leq$ upper_bound) were previously discussed in Section 1.

2.3 Constraints

The model enforces the following essential constraints to ensure valid solutions:

- **Circuit Constraint**:

  ```
  circuit(successor);
  ```

  This global constraint ensures the `successor` array forms a Hamiltonian circuit, meaning each node is visited exactly once in a single cycle.

- **Connecting END_NODES to START_NODES**:
  - $\forall i \in \{n+m+1, \ldots, n+2m-1\}, \quad \text{successor}[i] = i - m + 1$
  - $\forall i \in \{n+2, \ldots, n+m\}, \quad \text{predecessor}[i] = i + m - 1$
  - $\text{successor}[n+2m] = n+1, \quad \text{predecessor}[n+1] = n+2m$
  - $\forall i \in \text{START\_NODES}, \quad \text{successor}[i] \notin \text{END\_NODES}$

  The first four constraints work together to connect each courier's route seamlessly: the endpoint of one courier's path directly links to the starting point of the next courier's route. This structured connection continues down the line until the final courier's endpoint loops back to the very first starting point, forming one continuous loop that ties all routes together.

  The fifth and final constraint which is an implied one, in fact, introduces a key distinction between the baseline model and our CPF model: it blocks start nodes from ever being immediately followed by end nodes in the sequence.

- **Assign each pair start-end nodes with a courier**
  - $\forall i \in \text{START\_NODES}, \quad \text{route\_m}[i] = i - n$
  - $\forall i \in \text{END\_NODES}, \quad \text{route\_m}[i] = i - n - m$

  Here, with these constraints, we assign a courier to each pair of starting and ending nodes. For example, the k-th courier from m couriers will be assigned to the $n + k$ starting node and the $n + m + k$ ending node. It should be noted that the second constraint is an implied constraint, which helps in more complex instances by pruning search space.

- **Consistency Between Successor/Predecessor and Courier Assignments**:
  - $\forall i \in \text{NODES}, \quad \text{successor}[\text{predecessor}[i]] = i$
  - $\forall i \in \text{NODES}, \quad \text{predecessor}[\text{successor}[i]] = i$
  - $\forall i \in \text{ITEMS}, \quad \text{route\_m}[\text{successor}[i]] = \text{route\_m}[i]$
  - $\forall i \in \text{ITEMS}, \quad \text{route\_m}[\text{predecessor}[i]] = \text{route\_m}[i]$

  The first two constraints maintain consistency between successors and predecessors, ensuring a valid tour structure. The last two constraints enforce that a courier's route remains unchanged throughout the delivery process; if a location falls between a start and end node, the same courier must be responsible for its predecessor and successor. This prevents invalid assignments and ensures continuity in each courier's route. Additionally, both second and last constraints are implied ones.

- **Distance Accumulation Constraints**:
  - $\forall d \in \text{START\_NODES}, \quad \text{final\_dist}[d] = 0$
  - If an item's successor is also an item:
  $$\text{final\_dist}[\text{successor}[d]] = \text{final\_dist}[d] + D[d, \text{successor}[d]]$$
  - If an item's successor is an end node:
  $$\text{final\_dist}[\text{successor}[d]] = \text{final\_dist}[d] + D[d, n + 1]$$
  - If a start node's successor is an item:
  $$\text{final\_dist}[\text{successor}[d]] = D[n + 1, \text{successor}[d]]$$
  - If a start node's successor is an end node:
  $$\text{final\_dist}[\text{successor}[d]] = 0$$

  The first constraint initializes the distance at starting nodes to 0. Subsequent constraints extend the cumulative distance along routes. This ensures the total distance traveled is tracked from start to end nodes, including transitions to/from the depot.

- **Load Management Constraints**:
  - $\forall i \in \text{START\_NODES}, \quad \text{load}[i] = 0$
  - $\forall i \in \text{ITEMS}, \quad \text{load}[i] + \text{demand}[i] = \text{load}[\text{successor}[i]]$
  - $\forall i \in \text{START\_NODES}, \quad \text{load}[i] = \text{load}[\text{successor}[i]]$
  - $\forall i \in \text{ITEMS}, \quad \text{load}[i] \leq L[\text{route\_m}[i]]$
  - $\forall i \in \text{COURIERS}, \quad \text{load}[i + n + m] \leq L[i]$

  The first constraint sets the load at the starting nodes to zero. Later constraints make sure that loads are always added up before a node along a certain route by the courier that is responsible for it, and they also make sure that courier capacity limits are respected (L). The final constraint checks the load at end nodes.

- **Symmetry-Breaking Constraint**:
  - $\forall i, j \in \text{COURIERS}, i < j, \quad (L[i] = L[j]) \Rightarrow \text{successor}[n + i] < \text{successor}[n + j]$

  Orders couriers with identical capacities ($L[i] = L[j]$) by enforcing a sequence on their successor nodes, eliminating redundant permutations and reducing the solution space.

2.4 Validation

Experiment Design and Result: The experiment was conducted using both the baseline CP and CPF models, with and without symmetry breaking, across two solvers: Gecode and Chuffed. MiniZinc models were executed by a Python script.

For search strategies, we employed `int_search` combined with `first_fail` and `indomain_split`, as this search strategy yielded the best results for our model.

Additionally, preprocessing steps (converting data to DZN format and calculating bounds) are considered within the time limit.

The computational resource of the Docker container for the CP approach is as follows: M3-Pro 12-core CPU and 7.9 GB allocated memory out 18 GB total.

Inst.	**Models**				
	CPF-NS-C	**CPF-NS-G**	**CPF-S-C**	**CPF-S-G**	**CP-S-G**
1	**14**	**14**	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**	**226**
3	**12**	**12**	**12**	**12**	**12**
4	**220**	**220**	**220**	**220**	**220**
5	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**	**322**
7	**167**	**167**	**167**	**167**	**167**
8	**186**	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**	**244**
11	799	783	816	783	974
12	**346**	381	**346**	381	907
13	1806	1076	1838	1100	1598
14	1926	1073	1570	1165	1641
15	1332	1019	1090	1019	1565
16	**286**	**286**	**286**	**286**	**286**
17	1603	1413	1511	1456	1771
18	1134	819	1185	819	1347
19	**334**	390	**334**	390	**334**
20	1664	1599	1810	1641	1837
21	995	892	1046	892	1215

Table 1: Bold values indicate optimal solutions. Model abbreviations: CPF-NS-C = CPF_no_sym_chuffed, CPF-NS-G = CPF_no_sym_gecode, CPF-S-C = CPF_sym_chuffed, CPF-S-G = CPF_sym_gecode, CP-S-G = CP_sym_gecode. Due to space constraints, only the best results are shown. **Execution time is set for 300 seconds per instance. Other models' solutions are available in JSON files.**

3 SAT Model

To solve the MPC problem using a SAT approach, we utilized the Z3 solver. We applied different search strategies to our model, which has the same decision variables, constraints, and objective function. Natural numbers were encoded in binary to perform mathematical operations, which are interpreted within the binary representation.

3.1 Decision Variables

The structure of SAT model is based on the following variables:

1. **Assignment Matrix (a)**: A boolean matrix $a \in \{0,1\}^{m\times n}$. $a[i,j] = 1$ if and only if the i-th courier delivers item j.

2. **Path Assignment Tensor (x)**: A boolean tensor $x \in \{0,1\}^{m\times(n+1)\times(n+1)}$. $x[i,j,k] = 1$ if and only if the i-th courier travels from delivery point j to delivery point k. Note: $n+1$ refers to the origin point(O).

3. **Item Delivery Order Matrix (t)**: A boolean matrix $t \in \{0,1\}^{n\times n}$. $t[j,k] = 1$ if and only if item j is delivered as the k-th stop in a courier's route.

4. **Courier Load Matrix** (w): Boolean matrix $\in \{0,1\}^{m \times n_{bin\_load}}$ where $w[i]$ = binary encoding of courier i's actual load (n_{bin_load}: bit-length for max load)
5. **Distance Matrix** ($distances$): Boolean matrix $\in \{0,1\}^{m \times n_{bin\_dist}}$ where $distances[i]$ = binary encoding of courier i's total distance (n_{bin_dist}: bit-length for distance upper bound)

3.2 Objective Function

At each step of searching through the solution space to satisfy the constraints, the maximum distance is stored as the objective function. Then reduce it and search for the next answer to minimize it.

3.3 Constraints

The constraints necessary for the implementation of the model are the following:

- Ensure that each item is assigned to just one courier:

$$\bigwedge_{j=1}^{n} \text{exactly\_one}(a_{i,j} \mid i \in \{1,\ldots,m\})$$

- For each courier $i \in \{1,\ldots,m\}$, the total load is computed as:

$$w[i] = \sum_{j=1}^{n} a_{i,j} \cdot s\_bin[j],$$

  where $s\_bin[j]$ represents the weight of the j-th item. This defines the w variable. To ensure no courier exceeds its load capacity, impose the constraint:

$$\bigwedge_{i=1}^{m} \text{less}(w[i], l\_bin[i])$$

  l_bin: Binary encoding of courier capacity values.

- Ensure each item delivered exactly once and at a specific stop in its assigned courier's route:

$$\bigwedge_{i=1}^{n} \text{exactly\_one}(t_{i,j} \mid j \in \{1,\ldots,n\})$$

- Ensure that a courier can't leave from j-th delivery point to go to the same point (Don't have self loop):

$$\bigwedge_{i=1}^{m} \bigwedge_{j=1}^{n+1} \text{Not}(x_{i,j,j})$$

- Ensure if courier i carry item j, then this courier should leave item j:

$$\bigwedge_{i=1}^{m} \bigwedge_{j=1}^{n} (a_{i,j}, \implies \text{exactly\_one}(x_{i,j,k} \mid k \in \{1,\ldots,n+1\}))$$

$$\bigwedge_{i=1}^{m} \bigwedge_{j=1}^{n} (\text{Not}(a_{i,j}) \implies \text{allfalse}(x_{i,j,k} \mid k \in \{1,\ldots,n+1\}))$$

- Ensure if courier i carry item k, then this courier should reached item k:

$$\bigwedge_{i=1}^{m}\bigwedge_{k=1}^{n}(a_{i,k} \implies \text{exactly\_one}(x_{i,j,k} \mid j \in \{1,\dots,n+1\}))$$

$$\bigwedge_{i=1}^{m}\bigwedge_{k=1}^{n}(\text{Not}(a_{i,k}) \implies \text{allfalse}(x_{i,j,k} \mid j \in \{1,\dots,n+1\}))$$

- Ensure courier i leaves from origin and ensure courier i returns to origin:

$$\bigwedge_{i=1}^{m} \text{exactly\_one}(x_{i,n,j} \mid j \in \{1,\dots,n+1\}) \quad , \quad \bigwedge_{i=1}^{m} \text{exactly\_one}(x_{i,j,n} \mid j \in \{1,\dots,n+1\})$$

- Ensure if the i-th courier travels from delivery point j to delivery point k, then t_j and t_k should be consecutive:

$$\bigwedge_{i=1}^{m}\bigwedge_{j=1}^{n}\bigwedge_{k=1}^{n}(x_{i,j,k} \implies \text{consecutive}(t_j, t_k))$$

- If courier i travels from origin to delivery point j, then $\mathbf{t}_{j,1} = 1$:

$$\bigwedge_{i=1}^{m}\bigwedge_{j=1}^{n}(x_{i,n,j} \implies t_{j,1})$$

- The *distances* constraint specifies that for each courier i, the total distance traveled is calculated by considering all pairs $(j,k) \in M$, where M is the set of pairs satisfying $x[i,j,k] = 1$. The total distance for the i-th courier is then given by:

$$distances[i] = \sum_{(j,k)\in M} D[j,k], \quad \forall i \in \{1,\dots,m\}.$$

- **Implied constraints:**

  The implied constraints impose that each courier can't carry more than a certain number of items. For this constraint, the definition of the variable t must be changed. In the new definition, t is a boolean matrix that $t \in \{0,1\}^{n\times\lceil \frac{n}{m} \rceil}$. In the new definition, $t[j,k] = 1$ if and only if item j is delivered as the k-th stop in a courier's route but k has limitation and less than $\lceil \frac{n}{m} \rceil$. In other words, this definition means that the number of items in all couriers is less than $\lceil \frac{n}{m} \rceil$.

- **Symmetry breaking constraints:**

  For this constraints, at first we need to sort the courier capacity. If two couriers have the same load capacity, there is no possibility to distinguish them; this means that their paths could always be swapped. To break such symmetry, we impose a lexicographic ordering between them:

$$\forall i \in \{1,\dots,m-1\} \quad (l_i = l_{i+1}) \implies \text{less}(a_i, a_{i+1})$$

  Else, if the couriers don't have the same load capacity, it can happen that their path can be swapped, if their capacity is large enough. To disambiguate them, we force the first couriers (those whose capacity is higher) to carry a higher weight.

$$\forall i \in \{1,\dots,m-1\} \quad (l_i > l_{i+1}) \implies \text{less}(w[i+1], w[I])$$

3.4 Search Strategies

Due to the Z3 solver's lack of built-in optimization capabilities, we implemented a custom optimizer. This optimizer employs two key strategies:

- **Linear Search**: Decreases the upper bound step by step until the problem is unsatisfiable; the last valid solution is optimal.
- **Binary Search**: Iteratively halves the search space, updating lower and upper bounds until they converge to the optimal value.

3.5 Validation

The SAT model was implemented using the Z3 Python library. Table 2 shows the results of our SAT model, evaluated with both search methods, with and without symmetry breaking, and implied constraint. We found solutions for a few difficult cases (7th, 13th, and 16th) and the optimal solution for the 7th instance by the implied constraint, which can be seen in the table. The experiment was executed using the same hardware as the CP approach mentioned in Section 2.4.

Inst.	**LS**	**LS+SB**	**LS+IMP**	LS+SB+IMP	**BS**	**BS+SB**	**BS+IMP**	BS+SB+IMP
1	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**	**226**	**226**	**226**	**226**
3	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
4	**220**	**220**	**220**	**220**	**220**	**220**	**220**	**220**
5	**206**	**206**	**206**	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**	**322**	**322**	**322**	**322**
7	239	202	**167**	168	220	183	**167**	**167**
8	**186**	**186**	**186**	**186**	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**	**244**	**244**	**244**	**244**
13	N/A	N/A	N/A	N/A	N/A	N/A	1604	N/A
16	N/A	N/A	385	N/A	N/A	N/A	438	460
etc.	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Table 2: Results obtained using the SAT model with the defined strategies. LS, BS, SB, and IMP represent linear and binary search, symmetry breaking, and implied constraints. Some instances are N/A due to encoding failures or not finding a solution.

4 SMT Model

4.1 Decision Variables

The decision variable is defined as follows:

- **X** is a two-dimensional array where X[1..m,1..n] = 0..time. This indicates that the i^{th} courier collects the j^{th} item at time k, where k >0.
  If X[i,j] = 0, it implies that the i^{th} ourier does not collect the j^{th} item at all.

4.1.1 Auxiliary Variable

- **dist** is a one-dimensional array where dist[1..m] represents the total distance traveled by each courier, starting and ending at the origin.
- **load** is a one-dimensional array where load[1..m] represents the total weight of the items collected by each courier.
- **count** is a one-dimensional array where count[1..m] represents the number of items collected by each courier.

4.2 Objective Function

While solver is searching for solutions between the considered lower and upper bounds, we save the maximum of dist[i] as obj and keep the process of searching and after finding a solution, set it as the upper bound for finding the next solution till time terminates or become unsatisfiable.

4.3 Constraints

In all the constraints, the function $1(Condition)$ is assigned as shown below:

$$1(X_{i,j} > 0) = \begin{cases} 1, & \text{if } X_{i,j} > 0, \\ 0, & \text{otherwise.} \end{cases}$$

The main constraints of the model are:

- **$X_{i,j}$ value Constraint**

$$\bigwedge_{i \in \text{COURIERS}} \bigwedge_{j \in \text{ITEMS}} (0 \leq X_{i,j} \leq \text{count}_i)$$

$$count_i = \left(\sum_{j \in \text{ITEMS}} 1(X_{i,j} > 0) \right), \quad i \in \text{COURIERS}$$

This constraint ensures that each element in the X array is assigned a value between 0 and count[i]. If $X[i][j] = 0$, it means the i-th courier does not collect the j-th item. If $X[i][j] > 0$, it indicates that the i-th courier collects the j-th item at time k. The maximum valid value for k is equal to the total number of items collected by the i-th courier.

- **Item Collection Constraint**

$$\bigwedge_{j \in \text{ITEMS}} \left(\sum_{i \in \text{COURIERS}} 1(X_{i,j} > 0) = 1 \right)$$

This constraint ensures that each item is assigned to exactly one courier, guaranteeing that no item is left uncollected or assigned to multiple couriers.

- **Capacity Constraint**

$$\bigwedge_{i \in \text{COURIERS}} \left(\sum_{j \in \text{ITEMS}} 1(X_{i,j} > 0) \cdot s_j \leq l_i \right)$$

  This constraint ensures that the total load assigned to each courier does not exceed their maximum capacity

- **Unique Delivery Times for Each Item by a Courier**

$$Items_i = \{a_j \mid a_j = \begin{cases} X_{i,j}, & \text{if } X_{i,j} > 0, \\ -j, & \text{otherwise.} \end{cases} \quad \forall j \in \text{ITEMS}\} \quad , \bigwedge_{i \in \text{COURIERS}} \text{Distinct}(Items_i)$$

  This rule ensures that each item delivered by a courier has a unique delivery time. For every courier, we create a list of delivery times for all items. If an item is assigned to the courier, its delivery time is added to the list. If an item isn't assigned to the courier, we use a placeholder value `-j` to skip it. The `Distinct` constraint then guarantees that no two items delivered by the same courier can have the same delivery time. This keeps the schedule organized and avoids overlaps, making the delivery process more efficient.

4.3.1 Implied Constraints

- **Minimum Item Collection by Couriers Constraint**

$$\bigwedge_{i \in \text{COURIERS}} \left(\sum_{j \in \text{ITEMS}} 1(X_{i,j} > 0) \geq 1 \right)$$

  This rule ensures that every courier is assigned at least one item to deliver. Without it, the solver might waste time considering solutions where some couriers have no items at all. By adding this constraint, the search space becomes smaller and more focused, helping the solver find better and more practical solutions faster.

- **Balanced Item Distribution Constraint**

$$\bigwedge_{i \in \text{COURIERS}} \left(\sum_{j \in \text{ITEMS}} 1(X_{i,j} > 0) \leq \text{max\_item} \right) \quad , \quad \text{where max\_item} = \left\lfloor \frac{n}{m} \right\rfloor + 1$$

  This rule ensures that no courier ends up with too many items by setting a fair limit: each courier can carry at most $\frac{\text{number of items}}{\text{number of couriers}} + 1$ items. It keeps the workload balanced, so no one is stuck with an unfair share, and helps the solver find better solutions faster. By preventing overloading, it also reduces the maximum distance any courier has to travel, making the delivery process more efficient.

4.3.2 Symmetry Breaking Constraints

- **Load Assignment Constraint**
  To simplify the search process and avoid redundant combinations, a symmetry-breaking constraint is added. This rule organizes couriers by their assigned loads: if two couriers can carry the same amount, the one listed earlier must be given a heavier load.

4.4 Validation

We built an SMT model using Z3 and tested it with two techniques: symmetry-breaking and implied constraints. Results showed that implied constraints greatly improved efficiency, allowing the model to solve the 7th instance optimally. The results for all instances are shown in Table 3. Tests were run on an Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, with a 300-second time limit per instance

ID	**SMT**	**SMT_SYM**	**SMT_IMP**	**SMT_SYM_IMP**
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	207	195	**167**	**167**
8	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**
10	**224**	**224**	**224**	**224**
11..12	N/A	N/A	N/A	N/A
13	1926	1454	1758	1288
14..21	N/A	N/A	N/A	N/A

Table 3: Results from the SMT Approach Comparison Across Different Models.

5 MIP Model

In this section, we present our model for solving the Multiple Courier Planning problem with mixed integer programming. We employed the Python library PuLP to do this. In the following sections, we will discuss our model formulation.

5.1 Decision Variables

The model is based on the following variables:

- **x**[**k**][**i**][**j**]: Binary variable represents a 3D matrix of size $(m) \times (n+1) \times (n+1)$, where $k \in \{1, \dots, m\}$ and $i, j \in \{1, \dots, n+1\}$, where $x[k][i][j] = 1$ indicates that the courier k travel form i to j.

- **a**[**k**][**j**]: Binary variable represent a 2D matrix of size $(m) \times (n)$, where $k \in \{1, \dots, m\}$ and $j \in \{1, \dots, n\}$, which indicates whether courier k delivers item j. If $a[k][j] = 1$ it means that the courier k has item j.

- **t**[**k**][**j**]: Integer variable represents a 2D matrix of size $(m) \times (n)$, where $k \in \{1, \dots, m\}$ and $j \in \{1, \dots, n\}$, indicate the position of city i in the route of courier k. It is used in the Miller-Tucker-Zemlin (MTZ) subtour elimination constraints to prevent subtours.
- **max_distance**: Integer variable represents the maximum distance traveled by any courier. It is bounded between an upper bound and a lower bound that is used in the project, which is explained at the beginning of the document.
- **courier_weights**[**k**]: Integer array of size $\{1, \dots, m\}$, represents the total weight of items delivered by the courier k. It is bounded between an upper bound which is the maximum weight that courier k can carry, and a lower bound is the minimum weight, which is 0, meaning that a courier does not necessarily have to carry any items.
- **courier_distance**[**k**]: Integer array of size $\{1, \dots, m\}$, represents the total distance traveled by the courier k. It is bounded between an upper bound and a lower bound. The upper bound is the one that is used in the project, which is explained at the beginning of the document, and the lower bound is 0.

5.2 Objective Function

The objective function minimizes the variable `max_distance`, defined as the maximum distance traveled by any courier. To enforce this, the constraint

$$\texttt{max\_distance} \geq \texttt{courier\_distance}[k] \quad \forall k \in \{1, \dots, m\},$$

ensures `max_distance` serves as an upper bound for all individual courier distances. The solver iteratively reduces `max_distance` while satisfying routing, capacity, and subtour elimination constraints, thereby identifying the minimal possible value for the longest route.

5.3 Constraints

- **Item Delivery Constraint**

$$\sum_{k=1}^{m} a_{k,j} = 1 \quad \forall j \in \{1, 2, \dots, n\} \quad , \quad \sum_{k=1}^{m} \sum_{i=0}^{n+1} x_{k,i,j} = 1 \quad \forall j \in \{1, 2, \dots, n\}$$

  Ensures that every item is delivered by exactly one courier and that every city (except the depot) is entered exactly once.

- **Courier Capacity Constraint**

$$\sum_{j=1}^{n} S_j \cdot a_{k,j} = \text{courier\_weights[k]} \quad \forall k \in \{1, 2, \dots, m\}$$

  Ensures that the total weight of items delivered by each courier does not exceed their capacity.

- **No self-loop**

$$\sum_{i=1}^{n} x_{k,i,i} = 0 \quad \forall k \in \{1, 2, \dots, m\}$$

Ensures that no courier travels from a city to itself.

- **Start and End Constraint**

$$\sum_{j=1}^{n} x_{k,depot,j} = 1 \quad \forall k \in \{1, 2, \dots, m\} \quad , \quad \sum_{i=1}^{n+1} x_{k,i,depot} = 1 \quad \forall k \in \{1, 2, \dots, m\}$$

Ensures that every courier starts and ends their route at the depot.

- **Courier Leaves Item Constraint**

$$\sum_{i=1}^{n+1} x_{k,j,i} = a_{k,j} \quad \forall k \in \{1, 2, \dots, m\}, \forall j \in \{1, 2, \dots, n\}$$

Ensures that if a courier delivers an item, they must leave the city where the item is delivered. Means that if courier k delivers item j ($a_{k,j,i} = 1$), they must leave city j ($\sum_{i=1}^{n+1} x_{kij} = 1$).

- **Flow Conservation Constraint**

$$\sum_{i=1}^{n+1} x_{k,i,j} = \sum_{i=1}^{n+1} x_{k,j,i} \quad \forall k \in \{1, 2, \dots, m\}, \forall j \in \{1, 2, \dots, n\}$$

The number of times courier k enters city j must equal the number of times they leave city j.

- **No Loop Between Cities**

$$x_{k,i,j} + x_{k,j,i} \leq 1 \quad \forall k \in \{1, 2, \dots, m\}, \forall i, j \in \{1, 2, \dots, n\}, i \neq j$$

A courier cannot travel directly from city i to city j and then immediately back from city j to city i.

- **MTZ Subtour Elimination Constraint**

$$t_{k,i} - t_{k,j} \leq n \cdot (1 - x_{k,i,j}) - 1 \quad \forall k \in \{1, 2, \dots, m\}, \forall i, j \in \{1, 2, \dots, n\}, i \neq j$$

We are using MTZ (Miller-Tucker-Zemlin) subtour elimination [1] to prevents subtours by ensuring that the position of city i is less than the position of city j if courier k travels from i to j. Means that if courier k travels from city i to city j ($x_{k,i,j} = 1$), then city i must appear before city j in the route ($t_{k,i} < t_{k,j}$).

- **Courier Distance Calculation**

$$\sum_{i=1}^{n+1} \sum_{j=1}^{n+1} D_{ij} \cdot x_{kij} = \text{courier\_distance}_k \quad \forall k \in \{1, 2, \dots, m\}$$

Calculates the total distance traveled by each courier K which is the sum of the distances of all routes they take.

- **Max Distance Constraint**

$$\text{max\_distance} \geq \text{courier\_distance[k]} \quad \forall k \in \{1, 2, \dots, m\}$$

It ensure that the maximum distance traveled by any courier greater than or equal to the distance traveled by each courier.

5.4 Validation

We employed three solvers(CBC, Groubi, and HiGHS) for this section, and the results for all instances are shown in Table 4. Tests were run on an Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, with a 300-second time limit per instance

Instance	CBC	Gurobi	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	**167**	**167**	**167**
8	**186**	**186**	**186**
9	**436**	**436**	**436**
10	**244**	**244**	**244**
11..15	N/A	N/A	N/A
16	N/A	**286**	N/A
17, 18	N/A	N/A	N/A
19	N/A	**334**	N/A
20, 21	N/A	N/A	N/A

Table 4: Comparison performances between all models. Bold values indicate optimal solutions.

6 Conclusions

In conclusion, our project explored four optimization approaches—Constraint Programming (CP), SAT, SMT, and Mixed Integer Programming (MIP)—to solve the Multiple Couriers Planning problem, focusing on minimizing the maximum distance traveled while ensuring fair workload distribution. We applied both implied constraints and symmetry-breaking techniques across different approaches to reduce the search space and achieve optimality for some complex instances. The CP model, particularly the CPF variant with implied constraints, demonstrated superior performance, solving most instances optimally within time limits, while SAT and SMT approaches achieved success in specific cases but faced scalability challenges. The MIP model proved effective for smaller instances but struggled with larger ones due to computational complexity.

References

[1] AIMMS. Miller-tucker-zemlin formulation. https://how-to.aimms.com/Articles/332/332-Miller-Tucker-Zemlin-formulation.html, n.d.

[2] Osman Gokalp and Aybars Ugur. A multi-start ils–rvnd algorithm with adaptive solution acceptance for the cvrp. *Soft Computing*, 24(4):2941–2953, 2020.

[3] Andrea Rendl. Minizinc benchmarks - capacitated vehicle routing problem (cvrp), 2015. Available at: https://github.com/MiniZinc/minizinc-benchmarks/tree/master/cvrp.