Combinatorial Decision Making and Optimization (CDMO) Project Report

Hesam Sheikh Hassani*
Safoura Banihashemi†
Habib Kazemi‡

December 2024

1 Introduction

This report addresses the Multiple Couriers Planning (MCP) problem similar to the VRP problem [1][2], focusing on three approaches: Constraint Programming (CP), Satisfiability Modulo Theories (SMT), and Mixed-Integer Programming (MIP). The models share several core parameters:

- **m**: Number of couriers.
- **n**: Number of items.
- **l**: List of courier capacities.
- **s**: List of item sizes.
- **D**: Distance matrix representing travel distances between nodes.

The project was completed in 35 days. Hesam Sheikh Hassani developed the CP model, Habib Kazemi worked on SMT and Docker integration, and Safoura Banihashemi focused on the MIP model.

AI tools were used to assist with generating Python function documentation, creating GitHub Actions configurations to automate running experiments, and improving the clarity and readability of the report and readme.

*hesam.sheikhhassani@studio.unibo.it
†safoura.banihashemi@studio.unibo.it
‡habib.kazemi2@studio.unibo.it

2 CP Model

The CP model for the MCP problem uses a two-dimensional route matrix where rows represent couriers and columns represent positions in their routes (first item to deliver, second, and suchlike). This structure allows us to effectively capture both the assignment and routing aspects of the problem in a single representation. Moreover, the choice of matrix representation allows efficient and simple modeling in MiniZinc without unnecessary complexities and limitations.

2.1 Decision Variables

The model employs the following decision variables:

$$
\begin{aligned}
route_{k,p} &\in \{0, 1, ..., n+1\} && \forall k \in \{1, ..., m\}, p \in \{1, ..., p_{max}\} \\
route\_length_k &\in \{2, ..., p_{max}\} && \forall k \in \{1, ..., m\} \\
dist_k &\in Z^+ && \forall k \in \{1, ..., m\} \\
courier\_load_k &\in \{0, ..., \sum_{j=1}^{n} s_j\} && \forall k \in \{1, ..., m\} \\
max\_dist &\in Z^+
\end{aligned}
$$

Where $route_{k,p}$ represents the location visited by the courier k at position p in their route. A value of 0 indicates an unused position, values 1 to n represent delivery locations, and $n + 1$ represents the depot. $route\_length_k$ the actual length of courier k's route, including depot visits at the start and end. $dist_k$ is the total distance traveled by courier k along their assigned route. $courier\_load_k$ is the total load (sum of item sizes) assigned to the courier k, and max_dist, the maximum distance traveled by any courier (objective function).

2.2 Objective Function

The objective is to minimize the maximum distance traveled by any courier, encouraging a more fair item distribution among couriers:

$$\text{minimize} \quad \textbf{max\_dist} = \max_{k \in \{1,...,m\}} (\textbf{dist}[k])$$

2.3 Constraints

We encourage a minimal and simplistic set of constraints, allowing us to focus on the overall modeling and effective optimizations. The model includes the following constraints:

2.3.1 Route Structure Constraints

Each route must start and end at the depot:

$$\begin{aligned} route_{k,1} &= n+1 && \forall k \in \{1,...,m\} \\ route_{k,route\_length_k} &= n+1 && \forall k \in \{1,...,m\} \\ route\_length_k &\leq p_{max} && \forall k \in \{1,...,m\} \end{aligned}$$

Positions after the route length must be padded with zeros:

$$route_{k,p} = 0 \quad \forall k \in \{1,...,m\}, p \in \{route\_length_k + 1, ..., p_{max}\}$$

2.3.2 Assignment Constraints

Each delivery location must be visited exactly once across all routes:

$$\sum_{k=1}^{m} \sum_{p=2}^{p_{max}-1} [route_{k,p} = i] = 1 \quad \forall i \in \{1,...,n\}$$

We find that a choice of *count* constraint works much more efficiently than the $all\_different$ constraint. The *count* constraint ensures that each item is picked up only once, by each courier, and across all of them.

2.3.3 Capacity Constraints

The total load for each courier must not exceed their capacity:

$$\begin{aligned} courier\_load_k &= \sum_{p=2}^{route\_length_k-1} s_{route_{k,p}} && \forall k \in \{1,...,m\} \\ courier\_load_k &\leq l_k && \forall k \in \{1,...,m\} \end{aligned}$$

2.3.4 Distance Constraints

The total distance for each courier is calculated as:

$$dist_k = \sum_{p=1}^{route\_length_k-1} D_{route_{k,p},route_{k,p+1}} \quad \forall k \in \{1,...,m\}$$

2.3.5 Lower Bounds and Upper Bounds

We observe that in MiniZinc, adding lower and upper bounds does not have as much effect as it has its risks. The efficiency gains are not noticeable, and wrong bounds could prevent the solver from reaching solutions at all. On the back of this observation, our CP solver avoids the use of Upper Bounds as we reach satisfactory results without it. However, a lower bound constraint is set on the traveled distance of each courier, to ensure all couriers take at least one

item. This ensures that no courier remains stale, and the items are distributed to all of the couriers:

$$dist_k \geq \min_{i \in \{1,...,n\}} (D_{n+1,i}) \cdot 2 \quad \forall k \in \{1, ..., m\}$$

2.3.6 Symmetry Breaking Constraints

For couriers with equal capacity, we impose a lexicographic ordering on their routes to eliminate symmetric solutions:

$$route_i \leq_{lex} route_j \quad \forall i, j \in \{1, ..., m\} : i < j \wedge l_i = l_j$$

This constraint significantly reduces the search space by eliminating equivalent solutions that differ only in the assignment of identical couriers, while also respecting the maximum capacity of each courier.

2.3.7 Additional Techniques

To assign items efficiently and solve as many instances as possible, we leverage two additional techniques.

Route Matrix Limiting (RML): The size of the **route** matrix significantly influences the solver's search space and solving time. A trivial shape for the **route** matrix is $m \times (n+2)$, where $n+2$ represents all items and two depot nodes. However, in an optimal solution, no courier would visit all possible destinations. To optimize the model, we impose a limit on the maximum length of each courier's route, calculated as:

$$\text{max\_route\_length} = \lceil \frac{n}{m} \rceil + 2$$

This formula considers the average distribution of items among the couriers and adds 2 for the mandatory starting and ending at the depot. This drastically reduces the computation and time required to solve an instance.

Search Strategy with Multiple Phases: To efficiently solve the MCP problem, we use a heuristic-based search strategy divided into two phases:

1. **Route Construction:** The **route**$[k, p]$ variables are assigned using `first_fail`, prioritizing variables with smaller domains, and `indomain_split` which divides domains into halves for efficient exploration.

2. **Route Length Optimization:** The **route_length** variables are refined using `indomain_random`, enabling diverse exploration for optimal route lengths.

This strategy is implemented within a `seq_search` framework, progressively refining the solution with each phase.

2.4 Validation

Experimental design: To evaluate the solver's performance, we execute the MiniZinc model through a Python script, limiting the runtime to 300 seconds per instance. We test three configurations with varying levels of constraint relaxation:

- **CP_SYM_LB_RML_HRSTIC:** This configuration includes all proposed techniques and constraints: symmetry-breaking, lower bound, Route Matrix Limiting (RML), and search heuristics. We have two versions of this solver for `gecode` and `chuffed`, but they only differ in the value selection strategy of the *second phase* of the solver, as `chuffed` does not support `indomain_random`.
- **CP_SYM_LB:** Includes only symmetry-breaking and lower bound constraints.
- **CP:** The baseline configuration, incorporating only the essential problem constraints.

Each configuration is tested using both the `gecode` and `chuffed` solvers to provide a broader evaluation of the impact of the techniques. Our experimental results are as follows:

ID	Gecode			Chuffed		
	Base	**SYM+LB**	**OPTIMIZED**	**Base**	**SYM+LB**	**OPTIMIZED**
1	**14**	**14**	**14**	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**	**226**	**226**
3	**12**	**12**	**12**	**12**	**12**	**12**
4	220	**220**	**220**	**220**	**220**	**220**
5	**206**	**206**	**206**	**206**	**206**	**206**
6	322	**322**	**322**	**322**	**322**	**322**
7	167	167	167	167	167	**167**
8	186	**186**	**186**	**186**	**186**	**186**
9	436	N/A	436	N/A	N/A	**436**
10	244	N/A	244	244	N/A	**244**
12	N/A	N/A	346	N/A	N/A	N/A
13	1932	N/A	N/A	N/A	N/A	N/A
16	N/A	N/A	286	N/A	N/A	N/A
19	N/A	N/A	334	N/A	N/A	N/A

Table 1: Evaluation Results. **Note:** Base = Basic CP model; SYM+LB = CP_SYM_LB; OPTIMIZED = CP_SYM_LB_RML_HRSTIC. Instances where none of the solvers achieved any results have been excluded. N/A is for instances where the solver has returned no answer within the time limit.

We can observe that all of the solvers that reach a solution, even those that do not terminate before the time limit, yield the same optimal values.

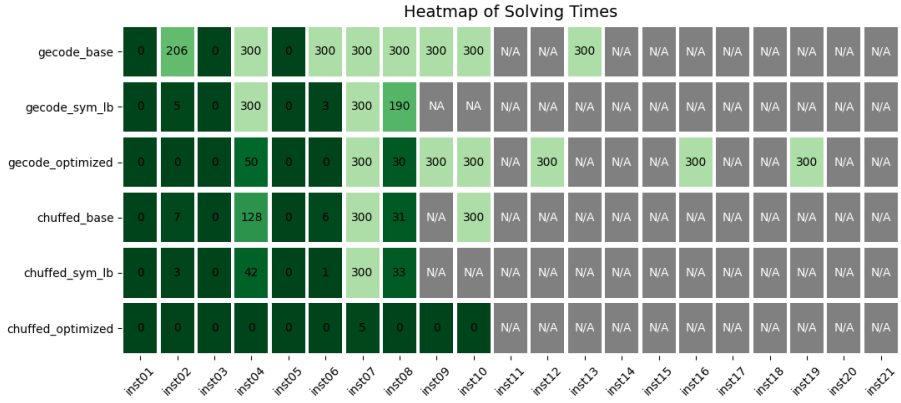

Figure 1: Heatmap of solving times with the six solver configurations. Darker cells indicate shorter solving times, while gray cells denote instances where the solver did not yield any answers within the time limit. Note: base = Basic CP model; sym_lb = CP_SYM_LB; optimized = CP_SYM_LB_RML_HRSTIC.

The heatmap in Figure 1 illustrates the solving times (in seconds) for various configurations of the solver in all instances. The optimized configurations (`gecode_optimized` and `chuffed_optimized`) generally perform better, achieving the shortest solving times in many instances. For harder instances, the base configurations often require the maximum allowed solving time (300 seconds). This visualization highlights the effectiveness of optimizations, particularly in reducing solving times for satisfiable instances.

Hardware: The experiment has been run on a local device with *Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz* processor, 6 CPU cores, 2 threads per core (not all cores utilized by solvers) and a memory of 15.9 GB.

3 SMT Model

The SMT model is similar to the CP approach described in Section 2.

3.1 Decision Variables

We used the decision variables $route_{k,p}$, $dist_k$, and max_dist, as detailed in Section 2.1.

3.2 Objective Function

The objective function remains the same as described in our CP-based approach in Section 2.2 and the goal is to `minimize` the `max_dist`. To establish a lower bound for `max_dist`, we define it as the maximum round-trip distance from the

depot to any item

$$\texttt{max\_dist} \geq \max_{i \in \{1,\ldots,n\}} \left(D_{n+1,i} + D_{i,n+1}\right)$$

3.3 Constraints

3.3.1 Assignment Constraints

Each item must be carried by exactly one courier, and all items must be delivered:

$$\bigwedge_{i=1}^{n} \left(\sum_{k=1}^{m} \sum_{p=2}^{p_{\max}-1} 1(route_{k,p} = i) = 1 \right)$$

where $1(\text{condition}) = \begin{cases} 1, & \text{if condition is true,} \\ 0, & \text{otherwise.} \end{cases}$

To improve computational efficiency, instead of directly implementing the above formula, we utilized the conjunction of the global constraints `AtMost(*bools, 1)` and `AtLeast(*bools, 1)` from Z3 creating an "exactly-one" constraint.

Each courier must deliver at least one item:

$$\bigwedge_{k=1}^{\mathrm{m}} (route_{k,2} \neq \mathrm{n}{+}1)$$

3.3.2 Route Structure Constraints

Each route must start and end at the depot:

$$\bigwedge_{k=1}^{m} (\mathrm{route}_{k,1} = n+1) \qquad \bigwedge_{k=1}^{m} (\mathrm{route}_{k,p_{\max}} = n+1)$$

All other positions must be either depot or an item:

$$\bigwedge_{k=1}^{m} \bigwedge_{p=2}^{p_{\max}-1} (1 \leq \mathrm{route}_{k,p} \leq n+1)$$

If a Courier reaches the Depot, it should stay there:

$$\bigwedge_{k=1}^{m} \bigwedge_{p=3}^{p_{\max}-2} (\mathrm{route}_{k,p} = \mathrm{n}+1 \Rightarrow \mathrm{route}_{k,p+1} = n+1)$$

3.3.3 Capacity Constraints

The total load for each courier must not exceed their capacity:

$$\bigwedge_{k=1}^{m} \left(\sum_{p=2}^{p_{\max}-1} s_{route_{k,p}} \leq l_k \right) \quad \text{where } s_{n+1} = 0.$$

3.3.4 Distance Constraints

The total distance for each courier is less than or equal to max_dist:

$$dist_k = \sum_{p=1}^{p_{\max}-1} D_{\text{route}_{k,p},\text{route}_{k,p+1}} \quad \forall k \in \{1,\ldots,m\}$$

$$\bigwedge_{k=1}^{m} (\texttt{max\_dist} \geq dist_k)$$

Symmetry Breaking Constraints: Please refer to Section 2.3.6.

$$\forall k_1, k_2, \ldots, k_m \left(\text{if } l_{k_1} = l_{k_2} = \cdots = l_{k_m}\right) \implies route_{k_1,2} < route_{k_2,2} < \cdots < route_{k_m,2}$$

3.4 Validation

The model was run in Z3 with and without symmetry breaking. While adding symmetry breaking reduced runtime for some instances, it did not enable the model to solve any additional instances beyond those solvable without symmetry breaking.

Experimental Results: GitHub Actions [4] were used to automate and standardize the execution of SMT model experiments in a reproducible Dockerized environment.

Hardware: GitHub Actions provides virtualized environments for isolated runs. Hardware details were inspected using `lscpu` and `free -h`. The hardware used includes *AMD EPYC 7763* as the processor, running on 4 vCPU cores with 2 threads per core, and 15GB of memory.

ID	Z3 with SB	Z3 without SB
1	**14**	**14**
2	**226**	**226**
3	**12**	**12**
4	**220**	**220**
5	**206**	**206**
6	**322**	**322**
7	**167**	**167**
8	**186**	**186**
9	**436**	**436**
10	**244**	**244**
11-15	N/A	N/A
16	**286**	**286**
17-21	N/A	N/A

Table 2: Results using Z3 with and without symmetry breaking. Optimal values are in bold.

4 MIP Model

Our Mixed-Integer Programming (MIP) model for the MCP problem uses a three-dimensional binary decision variable structure to represent routes, where the dimensions correspond to the departure city, destination city, and assigned courier. This setup combines city assignment and routing in a single framework. The model efficiently handles key constraints like capacity limits, sub-tour elimination, and distance minimization. By using PuLP, it provides a flexible and scalable way to balance the workload among couriers while minimizing the maximum distance traveled. It provides an optimal or near-optimal solution within the defined time limits.

4.1 Decision Variables

- x[i][j][c]: Binary variable representing a 3D matrix of size $(n+1) \times (n+1) \times m$, where $i, j \in \{1, \ldots, n+1\}$, and $c \in \{1, \ldots, m\}$, indicating whether courier c travels from city i to city j.
- u[i][c]: Integer variable representing a 2D matrix of size $n \times m$, where $i \in \{1, \ldots, n\}$, and $c \in \{1, \ldots, m\}$, indicating the position of city i in the route of courier c. Smaller values correspond to earlier visits in the sequence, meaning city i is visited earlier in courier c's route. This variable is part of the Miller-Tucker-Zemlin (MTZ) [3] formulation used to eliminate sub-tours by ensuring a valid order of city visits.
- courier_distance[c]: Integer array of size $1 \times m$ indicating the total travel distance for courier c.
- courier_weights[c]: Integer array of size $1 \times m$ indicating the total weight carried by courier c.
- max_distance: Integer variable representing the maximum travel distance among all couriers. It is bounded between a lower bound (lower_bound) and an upper bound (upper_bound).

4.2 Objective Function

Minimize the maximum travel distance (max_distance) among couriers:

$$\text{Minimize max\_distance}$$

$$\text{max\_distance} \geq \sum_{i=1}^{n+1} \sum_{j=1}^{n+1} \text{distance}(i,j) \cdot x_{i,j,c} \quad \forall c \in \{1, \ldots, m\}$$

4.3 Constraints

4.3.1 Route Structure Constraints

Each courier starts and ends their route at the depot:

$$\sum_{j} x_{\text{depot},j,c} = 1, \quad \sum_{j} x_{j,\text{depot},c} = 1, \quad \forall c$$

Prevent invalid assignments:

$$\sum_{c} x_{i,i,c} = 0, \quad \forall i$$

Ensures that for every courier c and city i, the number of times a courier enters the city equals the number of times it leaves:

$$\sum_{j \neq i} x_{i,j,c} = \sum_{j \neq i'} x_{j,i',c}, \quad \forall i, c$$

4.3.2 Assignment Constraints

Each city j must be visited exactly once by any courier:

$$\sum_{c} \sum_{i} x_{ijc} = 1, \quad \forall j$$

4.3.3 Capacity Constraints

The total weight carried by each courier must not exceed its capacity. The weight w_c is calculated as:

$$w_c = \sum_{i \in \text{cities}} \text{s}[i] \cdot x_{ijc}, \quad w_c \leq \text{l}[c]$$

4.3.4 Sub-Tour Elimination (MTZ Formulation) [3]

The Miller-Tucker-Zemlin (MTZ) formulation prevents sub-tours by enforcing the visiting order of cities:

$$\bigwedge_{c=1}^{m} \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} (i \neq j) \implies (u_{ic} - u_{jc} + n \cdot x_{ijc} \leq n - 1)$$

where u_i is the visiting order of city i, and n is the total number of cities.

4.3.5 Lower Bounds and Upper Bounds

These bounds are used to estimate the solution space for the MIP solver, ensuring that the algorithm explores feasible solutions while minimizing the maximum distance that any courier must travel.

- **Lower bound:** The lower bound is calculated as the longest round-trip distance between the depot and any city, ensuring that each courier can handle at least one trip:

$$\text{Lower Bound} = \max(\text{round\_trip\_distances})$$

  Where round_trip_distances is an array of distances for round trips from the depot to each city.

- **Upper bound:** The upper bound estimates the maximum travel distance a courier might need to cover by distributing the workload evenly among the couriers:

$$\text{Upper Bound} = \left(\frac{\text{n}}{\text{m}}\right) \cdot \max(\text{round\_trip\_distances}) + \text{Lower Bound}$$

4.4 Solvers Utilized

The model employs three solvers to maximize computational efficiency and solution quality:

1. **CBC Solver:** An open-source solver embedded in the PuLP library. Reliable for smaller instances with limited resources.
2. **HiGHS Solver:** Known for performance in large-scale linear programming. Offers improved speed for sparse matrices.
3. **Gurobi Solver:** A state-of-the-art commercial optimization solver. Features advanced heuristics and branch-and-bound techniques for rapid convergence.

Solver Execution:
Each solver is assigned a 300-second time limit. Results, including solution feasibility, optimality, and objective values, are stored in JSON format for further analysis.

4.5 Validation

Experimental Results: For the experimental setup and hardware specifications, please refer to Section 3.4

ID	CBC Solver	HiGHS Solver	Gurobi Solver
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**	**N/A**	**286**
17-21	**N/A**	**N/A**	**N/A**

Table 3: Results from the MIP Performance Comparison Across Different Solvers

The comparison of solvers (CBC, HiGHS, Gurobi) showed identical results for most instances, but Gurobi outperformed others on more complex cases, successfully solving instance 16 within the 300-second time limit.

Conclusion

In this project, we explored three methods—CP, SMT, and MIP—to solve the MCP problem. Among these, CP provided the best results across various instances. While we applied symmetry-breaking techniques and experimented with different lower and upper bounds to improve efficiency, these optimizations had limited impact on solver performance, except for MIP, which showed noticeable improvements on more complex instances. Our findings highlight CP's effectiveness for this problem, offering a solid foundation for addressing it.

References

[1] Bruno Scalia C. F. Leite (2023). The Vehicle Routing Problem: Exact and Heuristic Solutions. https://towardsdatascience.com/the-vehicle-routing-problem-exact-and-heuristic-solutions-c411c0f4d734

[2] Wikipedia contributors. (2024, October 3). Vehicle routing problem. In *Wikipedia, The Free Encyclopedia*.

[3] Aimms. (2020). *Miller-Tucker-Zemlin Formulation*. Retrieved from `https://how-to.aimms.com/Articles/332/332-Miller-Tucker-Zemlin-formulation.html`

[4] GitHub. (n.d.). GitHub Actions. Retrieved December 5, 2024, from *https://github.com/features/actions*.