CDMO Project Work a.y. 2023/2024

Modelling and solving the multiple couriers problem

Alessio Arcara alessio.arcara@studio.unibo.it
Alessia Crimaldi alessia.crimaldi@studio.unibo.it
Alessio Pittiglio alessio.pittiglio@studio.unibo.it

1 Introduction

Given:

- A set of n items, $N = \{1, 2, \ldots, n\}$.
- A depot, denoted by $n + 1$.

Each item i has a weight $s_i \geq 0$. We have *heterogeneous fleet* of m couriers, $K = \{1, 2, \ldots, m\}$, each with a capacity $l_k \geq 0$. Couriers start at the depot, collect one item, travel from i to j with a cost d_{ij} and return to the depot.

This problem is modeled as a complete digraph $G = (V, A)$ where:

- $V = \{1, 2, \ldots, n, n + 1\}$ are vertices.
- $A = \{(i, j) \in V \times V : i \neq j\}$ are arcs with cost d_{ij}.

An MCP instance is defined by a graph $G = (V, A, d_{ij}, s_i)$, a fleet K, and capacities $l_k \geq 0$ for each courier $k \in K$.

We split the work into three parts: Alessio Arcarca handled the MIP part, Alessia Crimaldi managed the CP part, and Alessio Pittiglio took care of the SAT part. At the end of the project, we decided to attempt implementing an SMT model by leveraging our experience with SAT. The project took us about a month to complete, during which it wasn't always possible to work consistently. The main difficulties we encountered were solving the more complex instances. With additional time and further exploration of the literature, we believe we might have been able to successfully address these challenges.

2 CP Model

After testing our initial model, named "positions", which uses a matrix of decision variables where rows represent couriers and columns represent items (source code provided for reference), we reviewed the literature to identify a more effective model. The model we will present closely follows the formulation found in the literature [3]. In addition to the problem input parameters described in Section 1, in order to proceed with the following modelling, we firstly construct a distance matrix D duplicating the last row and column of the d_{ij} input distances $2 * m$ times: one for the departure depot of each courier and the other one for the arrival depots. Moreover, following the same reasoning, in this case we redefine the set of *vertices* into $V = \{1, \dots, n\} \cup \{1, \dots, 2m\}$ to consider the items and both the departure and arrival depots of each courier.

2.1 Decision variables

To determine which items are taken from each courier and in which order, we use the following decision variables:

- The variable *successors* has domain [0..n+2m] with the meaning that $\textit{successors}_i = j$ iff the item j is the next item taken after i, for $i \in V$.
- The variable *assignments* has domain K with the meaning that $\textit{assignments}_i = j$ iff item i is taken by courier j, for $i \in V$.
- An important variable that allows subtour elimination is *positions*, which has domain [0..n+1] and represent the order in which the node i is visited by a courier, for $i \in V$; e.g. $\textit{positions}_{n+m+m} = 3$ means that the last courier has taken 2 items and then returned to depot.
- The variable *distances* has domain [1..sum(D)] with the meaning that $\textit{distances}_i = j$ iff j is the total travelled distance by courier i.

2.1.1 Auxiliary variables

In order to strengthen the propagation of the constraints, in addition to the decision variables, we also introduced the following auxiliary variables:

- The variable *predecessors* has domain [0..n+2m] with the meaning that $\textit{predecessors}_i = j$ iff the item j is the previous item taken before i, for $i \in V$.
- The variable *cps* has domain [0..max(l)] and represents the weights sum of the items taken by each courier, meaning that $\textit{cps}_i = j$ iff j is the sum between the already taken item sizes and the size of item i, for $i \in V$.
- The variable b has domain N and represent the set of items assigned to the couriers, i.e. $b = [A_1, \dots, A_m]$, where A_i is the set of items taken by courier i, for $i \in K$.
- The variable *dp* has domain [0..max(D)] with the meaning that $\textit{dp}_i = j$ iff j is the distance between node i and its successor node $\textit{successors}_i$, for $i \in$ [1..n+m].

2.2 Objective function

The objective function is to minimize the maximum distance travelled by any courier:

$$\text{minimize } \max_{i \in K} \text{ distances}_i = \sum_{j \in b_i} dp_j + dp_{n+i} \tag{2.1}$$

The bounds of the *distances* variable where previously discussed in Section 2.1.1.

2.3 Constraints

The problem constraints are:

- All couriers must return at depot, so the last m elements of the *successors* array must have successor value 0:
  for all $i \in K, \quad \text{successors}_{n+m+i} = 0$
- All couriers that leave the depot must return to it, so the assigned courier in $n+i$ (departure depots) must be coherent with the assigned courier of $n+m+i$ (arrival depots):
  for all $i \in K, \quad \text{assignments}_{n+i} = i$ AND $\text{assignments}_{n+m+i} = i$
- All couriers are firstly at the departure depot:
  for all $i \in K, \quad \text{positions}_{n+i} = 0$
- All visited nodes should be distinct except 0, that is used to represent no successor in the arrival depots. We impose this via the **global constraint**:
  all_different_except_0(successors)
- The assignment of a courier which take an item i must be the same assignment of the successor of the item i; and the item taken after the i item cannot be the same item i. We impose those with:
  for all $i \in [1..n+m]$,
  $\text{assignments}[\text{successors}_i] = \text{assignments}[i]$ AND $\text{successors}_i \neq i$
- The node visited after another one must have a position increased by one:
  for all $i \in [1..n+m], \quad \text{positions}[\text{successors}_i] = \text{positions}[i] + 1$
- The set of items taken by every courier must be linked with the assignments of the couriers in such a way to create the couriers routes. We impose this via the **global channeling constraint**:
  int_set_channel$([\text{assignments}_i \mid i \in N], b)$
- The sum of the weights of the items taken by a courier must be less or equal to the capacity of that courier: for all $i \in K, \quad \sum_{j \in b_i} s_j \leq l_i,$ which in our model we tried to impose via the ***cumulative*** **global scheduling constraint**, but unfortunately with underperforming results. Due to lack of time, we were unable to explore the reasons.

- The distance between a node i and its successor contained in the distance matrix D must be coherent with the same distance contained in variable dp: for all $i \in [1..n+m]$, $\quad D[i, \text{successors}_i] = dp_i$
- We impose the constraint for 2.1:
  for all $i \in K$, $\quad \text{distances}_i = \sum_{j \in b_i} dp_j + dp_{n+i}$

Implied constraint As shown in the literature [3], our analysis also found that including the following constraints involving the variables *predecessors* and *cps* helps improve model performance, leading to better propagation:

- Constraint that expresses the necessary and sufficient condition that must exist between predecessors and successors:
  for all $i, k \in V$, $\quad \text{predecessors}_k = i \iff \text{successors}_i = k$
- The sum of the item weights is defined as the sum of the weights of previous items increased by the weight of the taken item:
  for all $i \in N$, $\quad cps[\text{successors}_i] = cps_i + s_i$

Symmetry breaking constraint Permuting the couriers could generate equivalent solutions since swapping the roles of two identical couriers, meaning that they have the same load capacity, does not change the validity of the solution. To break this symmetry, we add the following ***leq_lesseq* global lexicographic constraint**:

for all $k_1, k_2 \in K \mid k_1 < k_2$ AND $l[k_1] = l[k_2]$,

$$
\begin{aligned}
&\text{lex} \leq (\\
&\qquad [\texttt{equality}(\text{assignments}_i, k_1) \mid i \in N],\\
&\qquad [\texttt{equality}(\text{assignments}_i, k_2) \mid i \in N]\\
&\qquad)
\end{aligned}
$$

Here `equality` is a function we create to generate two lists of binary variables based on the variable *assignments*: if the input $assignments_i$ and the input courier are equal, it returns 1; otherwise, it returns 0. In this way, we obtain lists of binary variables indicating the items assigned to each courier.
This additional symmetry breaking constraint can reduce the observed symmetry because it enforces a specific lexicographic order among the routes of identical couriers, thereby reducing the number of equivalent permutations to consider and, consequently, the search space.

2.4 Validation

We implemented our models in MiniZinc and ran them using Gecode and Chuffed.

Experimental design

The experiments were conducted on a MacBook Pro with an Apple M1 Pro chip, 16 GB of RAM, running macOS Sequoia. The version of MiniZinc used was 2.8.5. Before developing the current model, we built an alternative model for CP (described in Section 2), which unfortunately was found to be significantly underperforming. We then tested this better model using different search strategies, both for choosing the next variable and for selecting the next value in the domain of a variable to test, until we found the best search strategies for our model, which are *dom_w_deg* and *indomain_min* with *luby restart*, even if more exploration was needed. We tested our model on the two different solvers: each one was tested with and without symmetry breaking and implied constraints, and also with all combined. Moreover, Chuffed was set with a `free_search` parameter to *true*, allowing the solver to switch between its own search and the search parameters set by the user. Additionally, we set a time limit of 275 seconds for each test to find a solution.

Experimental results

	Gecode				Chuffed			
ID	NO_SB	SB	IMP	ALL	NO_SB	SB	IMP	ALL
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**
5	**206**	**206**	**206**	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**	**322**	**322**	**322**	**322**
7	215	217	224	229	**167**	**167**	**167**	**167**
8	–	–	–	–	**186**	**186**	**186**	**186**
9	–	–	–	–	**436**	**436**	**436**	**436**
10	–	–	–	–	**244**	**244**	**244**	**244**
11	–	–	–	–	–	–	–	–
12	–	–	–	–	–	–	–	–
13	1206	1156	1140	1176	–	1432	1454	1342
14	–	–	–	–	–	–	–	–
15	–	–	–	–	–	–	–	–
16	–	–	–	–	–	–	–	518
17	–	–	–	–	–	–	–	–
18	–	–	–	–	–	–	–	–
19	–	–	–	–	–	–	–	–
20	–	–	–	–	–	–	–	–
21	–	–	–	–	–	–	–	–

Table 1: Results using Gecode and Chuffed employing *dom_w_deg*, *indomain_min* and *luby restart* (1) without symmetry breaking or implied constraints, (2) with symmetry breaking constraints, (3) with implied constraints and (4) with all constraints.

3 SAT Model

We need to represent the delivery assignments and routes for each courier using a set of boolean variables. The model is a three-index vehicle flow formulation of the ACVRP [11], utilizing $O(n^2m)$ binary variables y_{ijk} and $O(nm)$ binary variables x_{ij}.

3.1 Decision variables

The variables x_{ij} are defined such that x_{ij} is true if courier i delivers item j, and false otherwise. Additionally, the variables y_{ijk} are defined such that $y_{i,j,k}$ is true if courier i travels directly from location j to location k, and false otherwise. Note that $j = n$ and $k = n$ refer to the depot. We also introduce a set of boolean variables seq_{ij} where seq_{ij} is true if item i is in position j in the delivery sequence, and false otherwise.

3.2 Objective function

We know that SAT solvers are not typically used for optimization purposes. Instead, we use pseudo-boolean (PB) constraints, which extend SAT solvers to handle optimization tasks, [1]. PB constraints involve inequalities of the form $C_0p_0 + C_1p_1 + \ldots + C_{n-1}p_{n-1} \leq C_n$, where each p_i is a literal and C_i is an integer coefficient. The objective function in our model is to achieve a fair division among drivers by minimizing the maximum distance traveled by any courier. To achieve this, we flatten the y variables and the distance matrix D, and set an upper bound $D_{\max}$ on the maximum distance traveled by any courier, defined as the sum of the maximum distances in each row of D. We then introduce constraints to ensure that the total distance traveled by each courier does not exceed $D_{\max}$. Iteratively, we need to tighten this constraint to progressively minimize $D_{\max}$.

3.3 Constraints

To ensure a feasible and optimal solution, we employ specific encoding strategies for our constraints. From now on, we will refer to the "exactly one" constraint as encoded using the sequential encoding method. For pseudo-Boolean (PB) constraints, we experimented with two methods: the *sequential weight counter* (SWC) and the *adder network*. The SWC encoding is a modification of the sequential counter (SEQ) encoding [9], which translates cardinality constraints into SAT. The idea is to encode the PB constraint by a circuit that sequentially sums from left to right the coefficients (also known as weights) whose variable p_i is set to true. The SWC encoding [4] maintains general arc consistency and efficiently handles large instances, requiring $O(nk)$ clauses and auxiliary variables, where n is the number of variables and k is the right-hand side. On the other hand, adder networks translate PB-constraints into clauses through half- and full-adders. This process involves synthesizing a circuit to sum the activated coefficients of the left-hand side into a binary number, which is then compared with the right-hand side binary representation. Although adder networks require significantly fewer clauses, $O(n \log(k))$, they do not maintain general arc consistency. Due to the ability of the SWC encoding to maintain general arc consistency and handle large instances effectively, we preferred using it in our model. Using these encoding assumptions, the constraints for our model are as follows:

1. Each item must be assigned to exactly one courier, so $\forall j \in \{1, \ldots, n\}$:

$$\text{ExactlyOne}(\{x_{ij} \mid i \in \{1, \ldots, m\}\})$$

2. Each courier must respect their load capacity, so $\forall i \in \{1, \ldots, m\}$:

$$\text{PbLe}(\{(x_{ij}, s_j) \mid j \in \{1, \ldots, n\}\}, l_i)$$

   where s_j is the size of item j and l_i is the load capacity of courier i.

3. Each courier's route must start and end at the origin, so $\forall i \in \{1, \ldots, m\}$:

$$\begin{aligned} &\text{ExactlyOne}(\{y_{ink} \mid k \in \{0, \ldots, n\}\}) && \text{(start at the origin)} \\ &\text{ExactlyOne}(\{y_{ikn} \mid k \in \{0, \ldots, n\}\}) && \text{(end at the origin)} \end{aligned}$$

4. For each item j visited by a courier, there must be exactly one outgoing and one incoming route, so $\forall i \in \{1, \ldots, m\}, \forall j \in \{1, \ldots, n\}$:

$$\begin{aligned} &x_{ij} \to \text{ExactlyOne}(\{y_{ijk} \mid k \in \{0, \ldots, n\}\}) && \text{(leave the item)} \\ &x_{ij} \to \text{ExactlyOne}(\{y_{ikj} \mid k \in \{0, \ldots, n\}\}) && \text{(arrive at the item)} \end{aligned}$$

5. To prevent sub-tours within each courier's route, we ensure that each position in the sequence is uniquely occupied using sequential encoding, so $\forall i \in \{1, \ldots, n\}$:

$$\text{ExactlyOne}(\{seq_{ij} \mid j \in \{1, \ldots, n\}\})$$

   Additionally, if a courier travels from j to k, k must be the successor of j, $\forall i \in \{1, \ldots, m\}, \forall j \in \{1, \ldots, n\}$:

$$y_{ijk} \to \text{Successor}(seq_j, seq_k)$$

Implied Constraints To strengthen the model and eliminate redundant solutions, we add constraints ensuring each courier has at least one item:

$$\bigvee_{j=1}^{n} x_{ij}$$

This constraint ensures that each courier is assigned at least one item. Without this constraint, the solver might consider solutions where some couriers have no assignments, leading to inefficient solutions and increased computational effort as the solver attempts to validate or invalidate such configurations. We also prevent single-node loops, $\forall i \in \{1, \dots, m\}, \forall j \in \{0, \dots, n\}$:

$$\neg y_{ijj}$$

This constraint eliminates the possibility of couriers traveling from a location back to the same location without making any deliveries. Such single-node loops are non-productive and would lead to trivial or redundant routes.

Symmetry Breaking Constraints The model suffers from inherent symmetry with respect to the numbering of the couriers, as any solution has $|m|!$ equivalent solutions obtained by permuting the courier indices [10]. To reduce this symmetry, we implement symmetry breaking constraints. We observe that the only distinguishing characteristic of a courier is their load capacity. Consequently, given a partition of items into m sets, if two or more couriers have the capacity to deliver the same two or more sets of items, the same solution can be represented by permuting the assignments of these sets to the couriers. This leads to numerous symmetric solutions with respect to the permutations of these sets . To address this, we first sort the couriers by their load capacities in decreasing order. This sorting ensures that couriers with higher capacities are considered first. To impose a constraint that the loads carried by the couriers must follow the same decreasing order, we introduce new variables w_{ik} to model the courier loads expressed in binary. These variables allow us to enforce that if courier i has a greater or equal capacity compared to courier $i+1$, then the load carried by courier i should be greater or equal to the load carried by courier $i+1$. Even after sorting and imposing the load order, we may still have permutation symmetries between couriers with identical (consecutive after sorting) load capacities who are assigned the same exact loads. To break these remaining symmetries, we impose a lexicographic ordering on the assignments of these couriers. This means that if two couriers i and $i+1$ have the same load capacity, then the assignment vector of courier should be lexicographically less than or equal to that of courier $i+1$.

3.4 Validation

The model has been implemented using Z3. Additionally, the model is written in DIMACS format, a solver-independent language, to allow testing with various SAT solvers. To achieve this, we followed the approach of converting the problem into a system of CNFs and outputting it in the DIMACS format. While Z3 currently supports the DIMACS format for input, it does not inherently reduce every problem to SAT. Some problems are solved using a propositional SAT solver only if the input contains Boolean and/or Bit-vector variables. Even then, there is no guarantee that Z3 will use a pure SAT solver. However, using

Z3's tactic framework, it is possible to control the conversion process [7]. This approach enables us to convert Boolean problems into propositional formulas in CNF format, making it straightforward to export them in DIMACS format.

Experimental design

Before presenting our experimental results, we provide a detailed description of our study. We used three solvers: Glucose, MiniSat, and Z3. The experiments were conducted on a MacBook Pro with an Apple M1 chip and 8 GB of RAM, running macOS Big Sur. The versions of the solvers were as follows: Z3 version 4.13.0, MiniSat version 2.2.0, and Glucose version 4.211. Each solver was allocated a maximum of 5 minutes (300 seconds) to find a solution or prove the problem unsatisfiable. The strategies employed involved different types of encodings for pseudo-Boolean constraints: the sequential weight counter (SWC), a built-in function of Z3, and the adder network. To ensure consistency and reproducibility, all experiments were run multiple times under the same conditions.

Experimental results

The results obtained with and without symmetry breaking and using the sequential weight counter (SWC) encoding for pseudo-Boolean constraints are reported in Tables 2 and 3.

ID	Z3 + SB	Z3 w/out SB
1	**14**	**14**
2	**226**	**226**
3	**12**	**12**
4	**220**	**220**
5	**206**	**206**
6	**322**	**322**
7	**167**	179
8	**186**	**186**
9	**436**	**436**
10	**282**	282

Table 2: Results using Z3 with and without symmetry breaking using the Z3 built-in function for pseudo-Boolean constraints.

4 SMT Model

The SMT model presented here addresses the capacitated vehicle routing problem with multiple couriers, using a three-index vehicle flow formulation as described in [11].

ID	Z3 + BS	Z3 + LS
1	**14**	**14**
2	**226**	**226**
3	**12**	**12**
4	**220**	**220**
5	**206**	**206**
6	**322**	**322**
7	167	183
8	**186**	**186**
9	538	436
10	341	341

Table 3: Comparison between Binary Search (BS) and Linear Search (LS) performance.

4.1 Decision variables

The decision variables, or literals, in this SMT model include x, y, and u. The literal x is a boolean matrix where $x_{ij} = \texttt{true}$ if and only if courier i delivers item j, representing a boolean variable from propositional logic (Bool). The literal y is a three-dimensional boolean matrix where $y_{ijk} = \texttt{true}$ if and only if courier i travels from location j to location k, also derived from propositional logic (Bool). Lastly, u is a two-dimensional integer matrix where u_{ij} represents the position of item j in the tour of courier i, based on integer theory (Int).

4.2 Objective function

The SMT model includes an objective function aimed at optimizing the routes of couriers in the capacitated vehicle routing problem. Specifically, the objective is to minimize the maximum distance traveled by any courier. The objective variable, U, represents the upper limit on the distance a courier can travel. This variable is defined as an integer, reflecting the maximum allowable distance. The total distance traveled by each courier i is calculated as the sum of the distances between locations j and k, weighted by the presence of a courier traveling between those points, represented by the boolean variable y_{ijk} Mathematically, this can be expressed as:

$$\sum_{j=0}^{n}\sum_{k=0}^{n} y_{ijk} D_{jk} \leq U$$

The model includes constraints ensuring that the total distance traveled by each courier does not exceed U.

4.3 Constraints

The primary constraints of our SMT model are as follows:

1. Each courier must respect their load capacity. For each courier $i \in \{1, \ldots, m\}$, the sum of the sizes of the items assigned to them must not exceed their capacity l_i, so $\forall i \in \{1, \ldots, m\}$:

$$\sum_{j=0}^{n} x_{ij} s_j \leq l_i$$

   where s_j is the size of item j.

2. Each item must be assigned to exactly one courier, so $\forall j \in \{1, \ldots, n\}$:

$$\text{ExactlyOne}(\{x_{ij} \mid i \in \{1, \ldots, m\}\})$$

   This ensures that no item is left undelivered or delivered by multiple couriers.

3. Each courier's route must start and end at the origin, so $\forall i \in \{1, \ldots, m\}$:

$$\begin{aligned} &\text{ExactlyOne}(\{y_{ink} \mid k \in \{0, \ldots, n\}\}) && \text{(start at the origin)} \\ &\text{ExactlyOne}(\{y_{ikn} \mid k \in \{0, \ldots, n\}\}) && \text{(end at the origin)} \end{aligned}$$

4. Additionally, if a courier visits an item, they must leave that item, and if they arrive at an item, they must leave from that item, so $\forall i \in \{1, \ldots, m\}, \forall j \in \{1, \ldots, n\}$:

$$\sum_{k=0}^{n} yijk = x_{ij}$$
$$\sum_{k=0}^{n} yjik = x_{ij}$$

5. Using the Miller-Tucker-Zemlin formulation [6], we ensure no subtours exist within any courier's route, so $\forall i \in \{1, \ldots, m\}, \forall j \in \{1, \ldots, n\}$:

$$1 \leq u_{ij} \leq n$$

   Furthermore, if a courier travels from j to k, then k must be the immediate successor of j, $\forall i \in \{1, \ldots, m\}, \forall j \in \{1, \ldots, n\}$:

$$\begin{aligned} y_{ijk} &= \texttt{True} \rightarrow u_{ij} + 1 = u_{ik} \\ y_{ink} &= \texttt{True} \rightarrow u_{ik} = 1 \end{aligned}$$

Implied Constraints To prevent couriers from making single-node loops, for each courier, so $\forall i \in \{1, \ldots, m\}, \forall j \in \{0, \ldots, n\}$:

$$y_{ijj} = \texttt{False}$$

4.4 Validation

The model has been implemented using Z3, leveraging its capabilities as an SMT solver. Z3 was chosen for its familiarity, as it had already been employed in the SAT section of this research, making the transition to using it as an SMT solver straightforward. Although we did not have sufficient time to implement the model using the solver-independent SMT-LIB format, Z3 provided an efficient and effective environment for developing and testing our model. Future work may include translating the model into SMT-LIB to enable experimentation with different SMT solvers.

Experimental design

Before presenting our experimental results, we provide a detailed description of our study to ensure reproducibility. We utilized the Z3 solver for our experiments due to its robust support for SMT problems and its previously demonstrated effectiveness in our SAT experiments. The experiments were conducted on a MacBook Pro with an Apple M1 chip and 8 GB of RAM, running macOS Big Sur. The version of Z3 used was 4.13.0. The search strategies employed included both binary and linear search techniques. Binary search was used to iteratively narrow down the feasible range for the maximum distance constraint, while linear search provided a straightforward approach to explore potential solutions incrementally. To ensure consistency and reproducibility, all experiments were repeated multiple times under identical conditions

Experimental results

The results obtained using linear search and binary search are reported in Table 4

ID	Z3 + LS	Z3 + BS
1	**14**	**14**
2	**226**	**226**
3	**12**	**12**
4	**220**	**220**
5	**206**	**206**
6	**322**	**322**
7	403	360
8	**186**	**186**
9	436	450
10	**244**	278

Table 4: Results using Z3 with linear search (LS) and binary search (BS) techniques.

5 MIP Model

The best model uses the *three-index vehicle-flow* formulation, closely following [11]. Unlike the two-index vehicle-flow formulation, which models the underlying fleet implicitly, the three-index formulation models it explicitly. This explicit modeling leads to a redundant representation that affects performance but is necessary to account for heterogeneous fleet characteristics, such as different load sizes in our case.

5.1 Decision variables

$$x_{ijk} \in \{0,1\} \qquad \forall (i,j) \in A, k \in K \tag{5.1}$$
$$y_{ik} \in \{0,1\} \qquad \forall i \in V, k \in K \tag{5.2}$$
$$1 \le u_{ik} \le n \qquad \forall i \in V \setminus \{n+1\}, k \in K \tag{5.3}$$
$$\text{maxCourDist} \in \mathbb{R}^{+} \tag{5.4}$$

- (5.1): Binary variable indicating whether courier k uses the arc (i,j) to move from vertex i to vertex j.
- (5.2): Binary variable indicating whether item i is collected by courier k.
- (5.3): Continuous variable representing the order in which items are collected by courier k; This is part of the *Miller-Tucker-Zemlin* (MTZ) formulation used to eliminate subtours; we have used this formulation instead of the *Dantzig-Fulkerson-Johnson* formulation because for instances with more than 60 items (like instances between 11 and 21), it performs better [5, 8].
- (5.4): Auxiliary variable to linearize the nonlinear objective function.

5.2 Objective function

As mentioned above, the objective function to minimize

$$\max_{k \in K} \left(\sum_{i \in V} \sum_{j \in V} d_{ij} x_{ijk} \right)$$

is nonlinear. To linerize this, we introduce a new auxiliary variable, maxCourDist, and add the following inequality:

$$\sum_{i \in V} \sum_{j \in V} d_{ij} x_{ijk} \le \text{maxCourDist}$$

This imposes an upper bound of the distance travelled by any courier. By substituting the original nonlinear objective function with

$$\text{minimize maxCourDist} \tag{5.5}$$

The solver will “push this variable down” while ensuring that no courier travels a distance greater than this variable. This new objective function acts equally to the original objective function but is now linear.

5.3 Constraints

$$\sum_{i \in V} \sum_{j \in V} d_{ij} x_{ijk} \leq \text{maxCourDist} \qquad \forall k \in K \tag{5.6}$$

$$\sum_{k \in K} y_{ik} = 1 \qquad \forall i \in V \setminus \{n+1\} \tag{5.7}$$

$$y_{n+1,k} = 1 \qquad \forall k \in K \tag{5.8}$$

$$x_{iik} = 0 \qquad \forall i \in V, k \in K \tag{5.9}$$

$$\sum_{j \in V} x_{ijk} = \sum_{j \in V} x_{jik} = y_{ik} \qquad \forall i \in V, k \in K \tag{5.10}$$

$$\sum_{i \in V \setminus \{n+1\}} s_i y_{ik} \leq l_k \qquad \forall k \in K \tag{5.11}$$

$$u_{ik} - u_{jk} + 1 \leq n(1 - x_{ijk}) \qquad \forall i, j \in V \setminus \{n+1\}, i \neq j, k \in K \tag{5.12}$$

- (5.7): Ensures that each item i is collected exactly once by one of the couriers.
- (5.8): Requires each courier to leave the depot.
- (5.9): Prevents any courier k from moving from an item i to itself, eliminating self-loops.
- (5.10): Ensures that if a courier k collects item i, they must also leave i.
- (5.11): Ensures that the total weight of items carried by courier k does not exceed their capacity l_k.
- (5.12): Eliminates subtours, preventing a courier from forming a loop that doesn’t include the depot. In the original formulation, the Miller-Tucker-Zemlin (MTZ) subtour elimination constraint is expressed as:

  $$x_{ij} = 1 \implies u_j \geq u_i + 1$$

  Since this constraint is nonlinear in its original form, it needs to be linearized using the Big-M trick, resulting in its equivalent linear inequality.

Implied constraints

$$\sum_{i \in V \setminus \{n+1\}} s_i y_{ik_1} \geq \sum_{i \in V \setminus \{n+1\}} s_i y_{ik_2} \qquad \forall k_1, k_2 \in K, k_1 \neq k_2, l_{k_1} > l_{k_2} \tag{5.13}$$

(5.13): This states that if courier k_1 has a higher capacity than courier k_2, then k_1 should carry at least as much weight as k_2. While this constraint might seem redundant, as an optimal solution would likely assign heavier loads to couriers with higher capacities, it is included to prevent scenarios where higher-capacity couriers are partially empty.

Symmetry breaking constraints

$$\sum_{i \in V \setminus \{n+1\}} i y_{ik_1} \geq \sum_{i \in V \setminus \{n+1\}} i y_{ik_2} \qquad \forall k_1, k_2 \in K, k_1 < k_2, l_{k1} = l_{k_2} \tag{5.14}$$

(5.14): Imposes a lexicographic ordering on the assignments of items to couriers with the same capacity, thereby eliminating equivalent solutions that are just permutations of each other.

5.4 Validation

We implemented our models in AMPL using AMPLPy, which enables us to interface AMPL with Python.

Experimental design

The experiments were conducted on a MacBook Pro with an Apple M1 Pro chip, 16 GB of RAM, running macOS Sequoia. We tested various open-source MIP solvers, specifically HiGHS, SCIP, GCG, and CBC. The versions of the solvers were as follows: CBC 2.10.10, SCIP 9.0.0, HiGHS 1.7.0, and GCG 4.0.0. Each solver was allocated a maximum of 275 seconds to find the optimal solution, to account for occasional delays where solvers might slightly exceed the time limit.

HiGHS and SCIP include symmetry detection and breaking algorithms. Therefore, it was interesting to compare the performance of the plain model against our symmetry-breaking constraint using HiGHS. Additionally, HiGHS is the only open-source solver available in AMPL with a Warm Start option. We computed our initial solution using a greedy algorithm with the nearest neighbor heuristic, which, for the TSP problem, on average results in paths that are 25% longer than the shortest path [12] while being very fast.

We also experimented with the *Set Partitioning Problem* formulation as detailed in [2]. This approach involved using a relaxed master problem in conjunction with a sub problem, where the goal was to iteratively find the column with the maximum reduced cost to be added to the master problem. Our idea was to find approximate solutions for the most challenging instances, where our previous *three-index vehicle flow* formulation hits a wall. However, we will not present the results of this attempt, as it did not meet our expectations, though we have left the details in the source code for reference.

Experimental results

The results obtained using the plain formulation with CBC, SCIP, GCG, and HiGHS solvers are reported in Table 5. Their runtimes are depicted in Figure 1. Additionally, the results obtained using HiGHS with different configurations are shown in Table 6, with their respective runtimes depicted in Figure 2. The omitted rows are the ones where none of the configurations/solvers found a solution.

ID	CBC	SCIP	GCG	HiGHS
1	**14**	**14**	**14**	**14**
2	**226**	**226**	-	**226**
3	**12**	**12**	**12**	**12**
4	**220**	**220**	-	**220**
5	**206**	**206**	**206**	**206**
6	**322**	**322**	382	**322**
7	**167**	**167**	-	**167**
8	**186**	**186**	219	**186**
9	**436**	**436**	-	**436**
10	**244**	**244**	273	**244**
13	-	572	-	572
16	-	-	-	347
19	-	-	-	461

Table 5: Results using the plain formulation with CBC, SCIP, GCG, and HiGHS solvers.

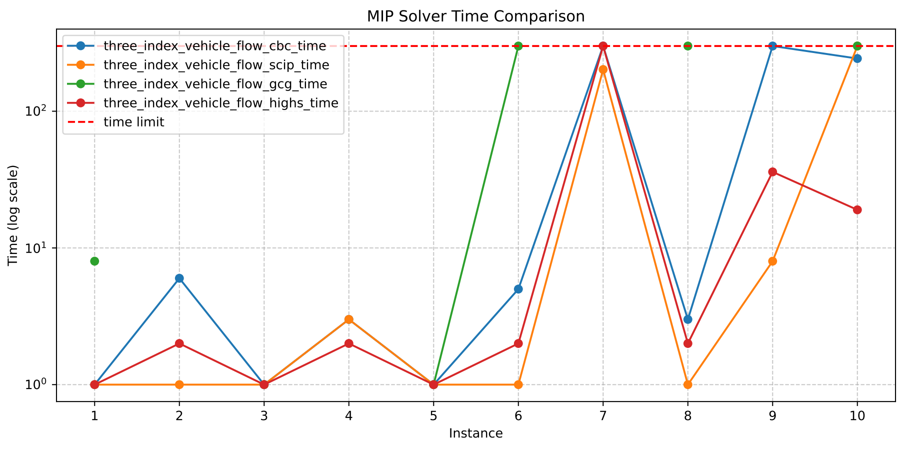

Figure 1: Runtime comparison for the plain formulation using CBC, SCIP, GCG, and HiGHS solvers.

ID	HiGHS	HiGHS+SB	HiGHS+SB+IMPLIED	HiGHS+SB+WM
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	**167**	**167**	**167**	**167**
8	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**
13	572	550	576	550
16	347	295	306	295
19	461	-	-	-

Table 6: Results using HiGHS with various configurations: (1) Plain, (2) with symmetry breaking, (3) with symmetry breaking and implied constraints, and (4) with all constraints plus warm start.

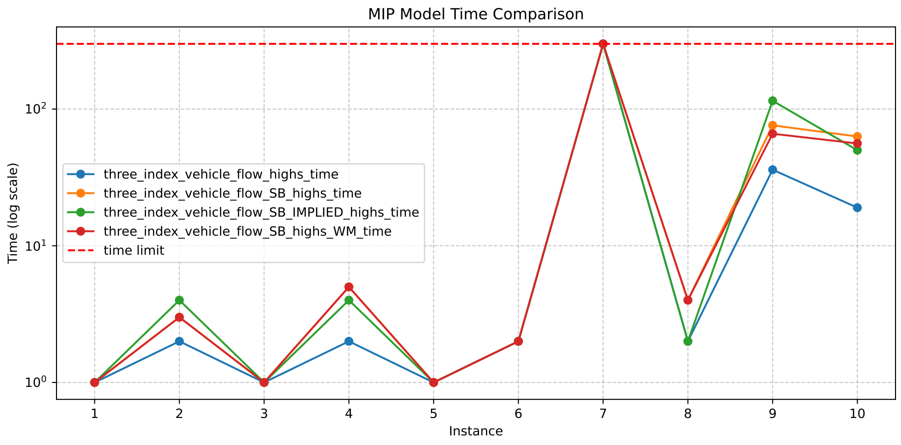

Figure 2: Runtime comparison of various configurations using the HiGHS solver.

6 Conclusions

In our project work, we implemented several formulations based on four paradigms: CP, SAT, SMT, and MIP. For each approach, we started with a model we developed ourselves and then searched the literature for the best formulations within each field. We consistently used the Miller-Tucker-Zemlin (MTZ) subtour elimination method where applicable and imposed a lexicographic ordering constraint to break symmetry, which allowed us to speed up the search and reach feasible solutions for the most challenging instances.

Our results indicated that MIP solvers, particularly HiGHS, found the best solutions. This may be attributed to HiGHS's capability in symmetry detection and algorithms for breaking symmetries, which potentially outperformed our manually imposed lexicographic constraint.

References

[1] Niklas Eén and Niklas Sörensson. “Translating pseudo-boolean constraints into SAT”. In: *Journal on Satisfiability, Boolean Modeling and Computation* 2.1-4 (2006), pp. 1–26.

[2] Dominique Feillet. “A tutorial on column generation and branch-and-price for vehicle routing problems”. In: *4OR* 8.4 (Dec. 2010), pp. 407–424. ISSN: 1614-2411. DOI: `10.1007/s10288-010-0130-z`. URL: `https://doi.org/10.1007/s10288-010-0130-z`.

[3] Matthieu Gondran, Eric Bourreau, and Philippe Lacomme. *Efficient Constraint Programming Approaches for routing problem: a case study for the VRP*. 2019. URL: `https://hal.science/hal-02181527/file/Verolog_2019_Bourreau.pdf#page43`.

[4] Steffen Hölldobler, Norbert Manthey, and Peter Steinke. “A compact encoding of pseudo-Boolean constraints into SAT”. In: *Annual Conference on Artificial Intelligence*. Springer. 2012, pp. 107–118.

[5] AIMMS How-To. *Comparing Formulations*. Accessed: 2024-06-21. 2024. URL: `https://how-to.aimms.com/Articles/332/332-Comparing-Formulations.html`.

[6] Clair E Miller, Albert W Tucker, and Richard A Zemlin. “Integer programming formulation of traveling salesman problems”. In: *Journal of the ACM (JACM)* 7.4 (1960), pp. 326–329.

[7] Leonardo de Moura. *Z3 and DIMACS output*. Accessed: July 6, 2024. 2012. URL: `https://stackoverflow.com/questions/13059096/z3-and-dimacs-output`.

[8] Gabor Pataki. “Teaching Integer Programming Formulations Using the Traveling Salesman Problem”. In: *Siam Review - SIAM REV* 45 (Mar. 2003), pp. 116–123. DOI: `10.1137/S00361445023685`.

[9] Carsten Sinz. “Towards an optimal CNF encoding of boolean cardinality constraints”. In: *International conference on principles and practice of constraint programming*. Springer. 2005, pp. 827–831.

[10] P. Toth and D. Vigo. *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, 2014. ISBN: 9781611973587. URL: `https://books.google.it/books?id=VUzUBQAAQBAJ`.

[11] Paolo Toth and Daniele Vigo, eds. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, 2002. DOI: `10.1137/1.9780898718515`. eprint: `https://epubs.siam.org/doi/pdf/10.1137/1.9780898718515`. URL: `https://epubs.siam.org/doi/abs/10.1137/1.9780898718515`.

[12] "Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP". In: *Discrete Applied Mathematics* 117.1 (2002), pp. 81–86. ISSN: 0166-218X. DOI: https://doi.org/10.1016/S0166-218X(01)00195-0. URL: https://www.sciencedirect.com/science/article/pii/S0166218X01001950.