Multiple Couriers Planning

Edoardo Merli - edoardo.merli@studio.unibo.it
Francesco Pigliapoco - francesco.pigliapoco@studio.unibo.it
Lorenzo Scaioli - lorenzo.scaioli@studio.unibo.it
Paolo De Angelis - paolo.deangelis7@studio.unibo.it

July 2023

1 Introduction

This report describes the different ideas behind the models implemented using the 4 paradigms studied in class, together with the results of their execution on the provided instances.

1.1 The main idea

We have identified two major classes of models depending on the logic behind their decision variables and constraints:

1. **Cluster-first, route-second**
   Ideally, the solver first finds a valid assignment of items to couriers, i.e. an items clustering scheme, and then searches different routes for each courier within its assigned set of items
2. **Route-first, cluster-second**
   Ideally, the solver first finds an order of the items deliveries, i.e. a sort of unique "big-route", and then splits it into segments of contiguous deliveries, one for each courier.

These two approaches can be visualized in figure 1 below, taken from [1]:

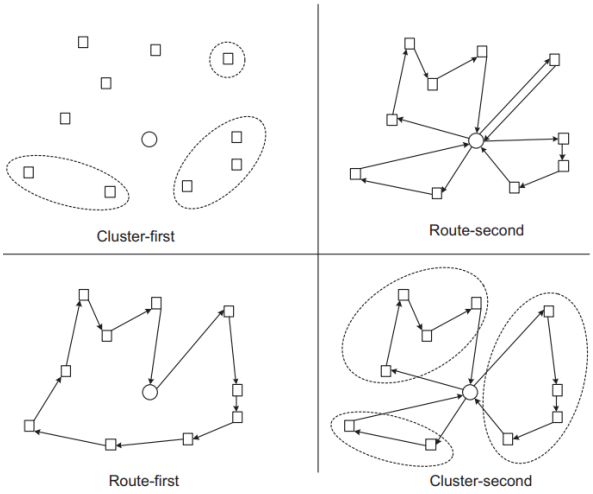

Figure 1: The two approaches to Vehicle Routing Problems. Every row describes an approach.

In our experimentation, we have tested both approaches and have found the first to be more effective. Its use is most evident in the SAT and SMT models, while it's more subtle in CP and MIP models.

1.2 Symmetry breaking constraint and pre-processing

Since the only characteristic identifying a courier is his load capacity, the problem presents a symmetry with respect to clusters assignments, i.e. considering an initial solution and its assignment of objects to couriers, we can obtain symmetric solutions by permuting the set of assigned objects (or clusters) between couriers having sufficient capacity to deliver them, while maintaining the same routes and hence the objective function's value unchanged.

The symmetry breaking constraint to tackle this symmetry has been implemented in two possible ways, since each one performed better or worse than the other depending on the solving method at hand.

1. **Global loads ordering**
   As a pre-processing step, order the couriers' load capacities l decreasingly and then impose $load_i \geq load_{i+1}\ \forall i = 1..m-1$ and a lexicographic ordering (model dependent) between couriers i and $i+1$ if $load_i = load_{i+1}\ \forall i = 1..m-1$ where $load_i$ is the *actual* load carried by courier i.

2. **Pairwise lexicographic ordering**
   For every possible pair of couriers i,j s.t. $i < j$ that can also carry each other's loads, namely $load_i \leq l_j \wedge load_j \leq l_i$, choose only one solution imposing a lexicographic ordering between couriers i and j.

1.3 Implied constraint and objective function bounds

The implied constraint consists in assigning to every courier at least one item. This is possible because of the assumption that $n \geq m$ and the fact that the distance matrix D is a *quasimetric* function (more info at [2]) hence, by the triangular inequality, giving an item to a courier who has none can only reduce our objective function, or more formally: $D_{O,j} + D_{j,O} \leq D_{O,j} + D_{j,k} + D_{k,O}$ and $D_{O,k} + D_{k,O} \leq D_{O,k} + D_{k,j} + D_{j,O}$, $\forall j,k = 1..n$, where O refers to the origin for more compact notation.

The objective function's bounds are:

- $lowerbound = max_{i=1..n}(D_{O,i} + D_{i,O})$ i.e. the maximum between all the round trips to deliver only object i.
- For the upper bound we have to consider an additional object for notation purposes, call it $S = sorted([\max_{k=1..n}(D_{j,k}) \text{ for } j = 1..n])$ i.e. the list of maximum step to leave every delivery point j. Then the upper bound is:

  (i) **without** the implied constraint:

$$upperbound = \sum_{i=2}^{n} S_i + \max_{k=1..n}(D_{O,k}) + \max_{k=1..n}(D_{k,O})$$

(ii) **with** the implied constraint:

$$upperbound = \sum_{i=m+1}^{n} S_i + \max_{k=1..n}(D_{O,k}) + \max_{k=1..n}(D_{k,O})$$

since in the worst case the first $m-1$ couriers deliver the first $m-1$ items, and the remaining objects $m..n$ must be delivered by the last courier m (we start summing from 2 and $m+1$ respectively since we impose that the route must end at the origin point and are adding the distance from the furthest point to O).

Organization of work and time taken

We decided to split the work in two and work in pairs: every couple would be assigned two methods in order to be able to discuss ideas and not have to work individually on a problem. In the end, we switched problems such that to review the other couple's work and share feedback and critics.

Regarding the time taken to complete the project, we started gathering ideas at the beginning of May, and progressively dedicated more and more time to it.

2 CP Model

In order to solve the MPC problem through a CP model we choose the constraint modeling language **MiniZinc**.

2.1 Decision variables

The model is based on the following variables:

- The integer matrix $T \in N^{m \times (n+1)}$, where $T[i,j] = j$ iff the courier i doesn't carry the item j, instead $T[i,j] = m$, with $m \in \{1,..,n+1\}$ if courier i goes from delivery point j to the delivery point m.
- A natural number $obj \in [lowerbound..upperbound]$, that represents the distance traveled by the courier taking the longest route, i.e. takes on the value of the objective function as defined in section 1.3.

2.2 Objective function

The objective function is to minimize $obj = \max_{i=1..m} \sum_{\substack{j=1..n \\ T[i,j] \neq j}} D[j, T[i,j]]$.

2.3 Constraints

2.3.1 *Load_constraint*

- The *Load_constraint* implies that $\sum_{\substack{j=1..n \\ T[i,j]\neq j}} s[j] \leq l[i] \;\; \forall i = 1..m$
  i.e. every courier i can't exceed its load capacity.

2.3.2 *Exactlyone_constraint*

- The *Exactlyone_constraint* implies that the number of $i \in 1, ..., m$ satisfying the condition $T[i, j] \neq j$ is exactly one, $\forall j = 1..n$. I.e. every object is delivered only once.

2.3.3 *T_constraint*

- The *T_constraint* makes use of Minizinc's global constraint *subcircuit*, that imposes that $T[i, ..]$ describes a Hamiltonian sub-cycle (i.e. a Hamiltonian cycle passing through a subset of the nodes) with the same definition of T given in section 2.1.

$$subcircuit(T[i, ..]) \; \forall i = 1..m$$

2.3.4 *Start_from_Origin_constraint*

- The *Start_from_Origin* implies that
  $|\{j \in 1, ..., n | T[i, j] \neq j\}| > 0 \implies T[i, n + 1] \neq n + 1 \forall i = 1..m$,
  namely if a courier delivers an object, the route (Hamiltonian sub-cycle) must pass by the Origin. This constraint is implicit, and hence omitted, if *Implied_constraint* below is active.

2.3.5 *Implied_constraint*

- The *Implied_constraint* implies that $T[i, n + 1] \neq n + 1 \; \forall i = 1..m$
  because each courier must carry at least one item and therefore must depart from the origin.

2.3.6 *Symmetry_breaking_constraint*

- The *Symmetry_breaking_constraint* encodes the "Pairwise lexicographic ordering" discussed in section 1.2, substituting $load_i := \sum_{\substack{j=1..n \\ T[i,j]\neq j}} s[j]$ and applying the lexicographic constraint on the rows of T, namely $lex\_lesseq(T[i, ..], T[j, ..])$.
  This implementation is more favorable in CP since the alternative "Global loads ordered" is wrongly exploited by Minizinc, which initially assigns the majority of items to the courier with highest capacity.

2.4 Validation

Two different solvers were used in the execution of the program: *Gecode*6.3.0 and $Chuffed$.

2.4.1 Experimental design

For the validation part we run the CP model on a *Dell XPS 15 9560, Intel Core i7-7700 CPU @ 2.8GHz, 4 Cores, 16GB RAM* and set the execution time to 300s. The search is used to guide the exploration of solutions and to further constrain variables. MiniZinc offers several search heuristics to guide the process of exploration and for this problem it is used $int\_search()$, which has three arguments:

- $Variables$: is an array composed by integer elements. In this position is placed T
- $Variable\ choice\ annotation$: This argument specifies the search heuristic or strategy for selecting the next variable to assign a value to during the search. It determines the order in which the domain values of the variables are explored. For this problem dom_w_deg is chosen, which considers the variable with the smallest value of domain size divided by weighted degree, which is the number of times it has been in a constraint that caused failure earlier in the search.
- $Value\ choice\ annotation$: is a choice of how to constrain a variable. It is used $indomain\_min$ to assign the variable its smallest domain value.
- $Large\ Neighbourhood\ Search$: this search strategy, together Luby restart ($restart\_luby(100)$), has been tested only on "difficult problems", which have been defined as problems not solved without it in less than 30 seconds. We applied LNS with 15% change percentage on T, namely $relax\_and\_reconstruct(array1d(T), 85)$.

2.4.2 Experimental results

ID	Gecode no_LNS	Gc no_LNS_no_SB	Gc no_LNS_no_impl	Chuffed
1	**14**	**14**	**14**	**14**
2	**244**	**244**	**244**	**244**
3	**12**	**12**	**12**	**12**
4	**220**	**220**	**220**	**220**
5	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**
8	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**
10	**244**	**244**	283	**244**

ID	Gecode no_LNS	Gc LNS	Gc LNS_no_SB	Gc LNS_no_impl	Chuffed
7	**167**	201	172	185	**167**
11	N/A	N/A	525	N/A	N/A
12	N/A	N/A	**346**	N/A	N/A
13	N/A	562	590	584	1350
14	N/A	N/A	711	N/A	N/A
15	N/A	N/A	668	N/A	N/A
16	N/A	N/A	**286**	N/A	N/A
17	N/A	N/A	986	N/A	N/A
18	N/A	N/A	564	N/A	N/A
19	N/A	N/A	**334**	N/A	N/A
21	N/A	N/A	575	N/A	N/A

Table 1: Results using Gecode (Gc) with ("difficult problems") and without ("easy problems") restart and Large Neighbourhood Search, with and without Symmetry Breaking, with and without implied constraint and using Chuffed.

3 SAT Model

In order to solve the MPC problem through a SAT model we choose **Z3 solver**. We developed two models, that share decision variables, constraints and objective function, but differ in the search procedure. For a more natural discussion, those are presented later in 3.4. We encoded natural numbers in binary in order to do Math on them, so every mathematical operation below has to be interpreted in the natural space of binary numbers.

3.1 Decision variables

The model is based on the following variables:

- A boolean matrix $a \in \{0,1\}^{m \times n}$, where $a[i,j] = 1$ iff the *i-th* courier delivers the item *j*.
- A boolean tensor $r \in \{0,1\}^{m \times (n+1) \times (n+1)}$, where $r[i,j,k] = 1$ iff the courier i goes from delivery point j to delivery point k. Delivery point $n+1$ is the origin O.
- A boolean matrix $t \in \{0,1\}^{n \times n}$ that encodes the visit sequence of the various delivery points reached by the couriers: $t[j,k] = 1$ iff the item j is delivered as *(k+1)-th* in its courier's route.
- A boolean matrix $courier\_loads \in \{0,1\}^{m \times n\_b1}$ where n_b1 is the number of bits necessary to encode the maximum possible load of a courier. The array $courier\_loads[i]$ $\forall i = 0..m-1$ is the binary representation of the actual load carried by the *(i+1)-th* courier.
- A boolean matrix $distances \in \{0,1\}^{m \times n\_b2}$ where n_b2 is the number of bits necessary to encode the maximum possible distance traveled by

a courier (i.e. the bits of *upper_bound*). The array $distances[i]$ $\forall i = 0..m-1$ is the binary representation of the distance traveled by the *(i+1)-th* courier. It has the same upper bound *upper_bound* of the objective function and $lower\_bound = \min_{j=0..n-1} D[n,j] + D[j,n]$ if the implied constraint is active, $lower\_bound = 0$ otherwise (although in general the lower bound on the distances is not imposed, they are simply a binary representation).

3.2 Objective function

The objective function is to minimize $distances[i]$ over $i = 0..m-1$, with the same bounds discussed in section 1.3.
Depending on the model, we implemented Linear search and Binary search, by iteratively querying the solver and applying the appropriate bounds to the objective function.

3.3 Constraints

3.3.1 *a*

- The $assignment\_j$ $\forall j = 0..n-1$ constraints implies that $\forall j = 0..n-1$ $\exists! i \in \{0..m-1\}$ s.t. $a[i,j] = 1$, so that every item is delivered by just one courier.

3.3.2 *courier_loads*

- The $compute\_courier\_load\_i$ $\forall i = 0..m-1$ constraints implies that $courier\_loads[i] = \sum_{j=0}^{n} a[i,j] * s\_bin[j]$ where $s\_bin[j]$ is the size of the *(j+1)-th* item, in this way we define the *courier_loads* variable. Then we impose $leq(courier\_loads[i], l\_bin[i])$ $\forall i = 0..m-1$ so that every courier can't exceed its load capacity.

3.3.3 *r*

- The *zero_diagonal* constraint implies that $\forall i = 0..m-1$ $\forall j = 0..n-1$ $r[i,j,j] = 0$, so that a courier can't leave from *j-th* delivery point to go to the same point.
- The $if\_a[i,j]\_then\_r[i,j,:]$ $\forall i = 0..m-1$ $\forall j = 0..n-1$ constraints implies that $\forall i = 0..m-1$ $\forall j = 0..n-1$ if $a[i,j] = 1$ then $\exists! k$ s.t. $r[i,j,k] = 1$ else $r[i,j,k] = 0$ $\forall k = 0..n-1$. In this way the *(i+1)-th* courier transports the *(j+1)-th* item in a iff the courier arrives at the *(j+1)-th* delivery point during its route.
  More over the *courier_i_leaves_origin* constraint implies that every courier has to leave from the origin.
- The $if\_a[i,j]\_then\_r[i,:,j]$ $\forall i = 0..m-1$ $\forall j = 0..n-1$ constraints implies that $\forall i = 0..m-1$ $\forall j = 0..n-1$ if $a[i,j] = 1$ then $\exists! k$ s.t. $r[i,k,j] = 1$ else

$r[i,k,j] = 0\ \forall k = 0..n-1$. In this way the *(i+1)-th* courier transports the *(j+1)-th* item in a iff the courier departs from the *j-th* delivery point during its route.
Moreover the *courier_i_returns_to_origin* constraint implies that every courier has to return to the origin.

- The *maintain_t_order* constraint implies that $\forall i = 0..m-1\ \forall j,k = 0..n-1$ if $r[i,j,k] = 1$ then $t[j]$ and $t[k]$ must be successive. In this way we maintain the order of the deliveries encode in t even in the r tensor. Moreover $\forall i = 0..m-1\ \forall j = 0..n-1$ if $r[i,n,j] = 1$ than $t[j,0] = 1$, so that the first element delivered by the *(i+1)-th* courier from the origin is the same in t and r.

3.3.4 *t*

- The *time_of_j* $\forall j = 0..m-1$ constraints implies that $\forall j = 0..n-1$ $\exists! k \in \{0..n-1\}$ s.t. $t[j,k] = 1$, so that every item is delivered only once in a defined order in its courier's route.

3.3.5 *distances*

- The *def_distances* constraint implies that, considering the set M such that every couple $(j,k) \in M$ satisfy the condition $r[i,j,k] = 1$, then $distances[i] = \sum_{(j,k)\in M} D[j,k]\ \forall i = 1..m$.

3.3.6 Implied constraints

- The *a_implied_constraint* imposes that $\forall i = 1..m-1\ \exists\ j \in 0..n-1$ s.t. $a[i,j] = 1$, in this way every courier has to deliver at least one item.
- The *r_implied_constraint* imposes that $r[i,n+1,n+1] = 0$, so that every courier has to reach at least one delivery point and can't just satisfy *courier_i_leaves_origin* and *courier_i_returns_to_origin* constraints by doing a self-loop on the origin point.

3.3.7 Symmetry breaking constraints

- The *symmetry_breaking* constraint encodes the "Global loads ordering" discussed in section 1.2, substituting $load_i := couriers\_loads[i]$ *couriers_loads* and applying the lexicographic constraint on couriers delivering same loads on the respective rows of a, namely $lex\_lesseq(A[i,..], A[i+1,..])$.
This implementation is more favorable in SAT since it uses fewer constraints.

3.4 The models

Here we discuss the search strategy employed by the second model, which is essentially a *sequential search*. The second model, in fact, makes use of two

solvers in order to follow a more informed search strategy:

1. The first solver's task is to find an assignment matrix a that satisfies its respective constraints.
2. Once the first solver has found a suitable a, its values are injected into the second solver, whose task is to find an (optimal) routing scheme for the couriers.

Obviously, the first solver's variables are then assigned to take a different configuration of value from the one just found in this iteration.

This is essentially the direct implementation of a cluster-first, route-second approach, as described in section 1.

We are aware that such a behavior can be obtained also with only one solver using incremental solving procedures; however, our tests have shown that such an approach was much slower. Our hypothesis is that keeping the same variables present in each sub-solver enables it to make use of the structure learned during the previous iterations, instead of losing such information every time.

3.5 Validation

The first model has been implemented supporting both linear and binary search, while the second one supports only linear search. This is because the injection phase requires iterating over all the matrix a, which makes backtracking too expensive.

3.5.1 Experimental design

For the validation part we run the two SAT models on a *Dell XPS 15 9560, Intel Core i7-7700 CPU @ 2.8GHz, 4 Cores, 16GB RAM*. We run the two model with binary search with/without implied constraint and with/without symmetry breaking, moreover we run the one solver model with linear search. For every solver we set the execution time to 300s.

3.5.2 Experimental results

ID	base	b_no_SB	b_no_impl	b_linear	sequential	seq_no_SB	seq_no_impl
1	**14**	**14**	**14**	**14**	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**	**226**	**226**	**226**
3	**12**	**12**	**12**	**12**	**12**	**12**	**12**
4	**220**	**220**	**220**	**220**	**220**	**220**	**220**
5	**206**	**206**	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**	**322**	**322**	**322**
7	185	234	196	192	341	284	173
8	**186**	**186**	**186**	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**	**436**	**436**	**436**
10	**244**	**244**	251	**244**	**244**	**244**	**244**
13	N/A	N/A	N/A	N/A	1228	1616	1202

Table 2: Results using a unique solver (base) and two sequential solvers (seq), with and without Symmetry Breaking, with and without implied constraint, and using a base linear solver

4 SMT Model

In order to solve the MPC problem through a SMT model we choose **Z3 solver**.

4.1 Decision variables

The model is based on the following variables, where boolean variables are from the Propositional Logic Theory, and integer variables are from LIA Theory.

- A boolean matrix $A \in \{0,1\}^{m \times n}$ where $A[i,j] = 1$ iff courier i delivers item j.
- An integer matrix $O \in N^{m \times n}$, where $O[i,j] = k$ iff courier i delivers item j as the k-th object in his route. $O[i,j] = 0$ iff courier i doesn't deliver item j.
- A set of integers $load \in N^m$ were $load[i]$ represents the total load carried by the courier i.
- A set of integers $count \in N^m$ were $count[i]$ represents the total number of items carried by courier i.
- An array $dist \in N^m$ where $dist[i]$ represents the total distance traveled by courier i.
- A natural number $obj \in N$, representing the objective function's value.

4.2 Objective function

The objective function is to minimize $obj = \max_{i=1..m}(dist[i])$, with the same bounds discussed in section 1.3.

4.3 Constraints

4.3.1 A

- $PbLe(\{(A[i,j], s[j])|j = 1..n\}, l[i])\forall i = 1..m$ encodes the load capacities constraint. It has been favored over $load[i] <= l[i]$ due to its efficiency.
- $\sum_{i=1}^{m} A[i,j] = 1 \ \forall j = 1..n$ encodes "exactly one" over the columns of A, encoding that each item is delivered by exactly one courier.

4.3.2 O

- The *O_correspondences_constraint* implies $O[i,j] > 0 \leftrightarrow A[i,j] = 1$, i.e. item j is in the route of courier i iff courier i delivers it.
- The second and third constraints exploit the variables *order_items* and *non_zero_items*.

  The *route_O_distinct_constraint* implies that $Distinct(\{O[i,j] \text{ if } O[i,j] > 0 \text{ else } -j | j = 1..m\})$, i.e. that the positive entries of O are all distinct. The $-j$ trick enables us to encode this as an "all different" constraint.

  The *value_O_constraint* implies that $O[i,j] <= count[i] \ \forall i = 1..m, j = 1..n$. This ensures that O describes a path and that no numbering is skipped.
- The *Dist_constraint* implies that $\forall i = 1..m \ dist[i] = \sum_{\substack{j1,j2=1..n \\ O[i,j1]\neq 0 \\ O[i,j2]=O[i,j1]+1}} D[j1,j2] + \sum_{j| \ O[i,j]=1} D[n+1,j] + \sum_{\substack{j| \\ O[i,j]=count[i]}} D[j,n+1]$
  encoding the distance of travelled by courier i.

4.3.3 *Load_constraint*

- The *Load_constraint* implies that $load[i] = \sum_{\substack{j=1..n \\ A[i,j]=1}} s[j] \quad \forall i = 1..m.$

4.3.4 *Counts_constraint*

- The *Counts_constraint* implies $count[i] = \sum_{\substack{j=1..n \\ A[i,j]=1}} 1 \quad \forall i = 1..m.$

4.3.5 Implied constraint

- $Or(A[i])\ \forall i = 1..m$ in order to impose the delivery of at least one object by each courier.

4.3.6 Symmetry breaking constraint

- See *Symmetry_breaking* constraint of SAT, section 3.3.7.

4.4 Validation

4.4.1 Experimental design

For the validation part we run the SMT model on a *Dell XPS 15 9560, Intel Core i7-7700 CPU @ 2.8GHz, 4 Cores, 16GB RAM* and set the execution time to 300s. After the first attempts made using a single solver, there was a lack of efficiency in finding solutions. For this reason, given the need to determine the arrays A, O, and the array *dist* according to the order in which they are written, it was essential to consider first two solvers (like in SAT) and then three solvers: *Solver_A*, *Solver_O*, *Solver*

The last two solvers contain the constraint of the first, and the last contains the constraint of the previous. The first step is to determine the matrix A using *Solver_A*. Similarly, we obtain O using *Solver_O* and finally *dist*, which is stored as *result_objective*. The search is iterated considering that $obj < result_objective$ and if $result\_objective = lower\_bound$ the process is stopped.

4.4.2 Experimental results

ID	base	sq2_sol	sq2_no_SB	sq2_no_impl	sq3_sol	sq3_no_SB	sq3_no_impl
1	**14**	**14**	**14**	**14**	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**	**226**	**226**	**226**
3	**12**	**12**	**12**	**12**	**12**	**12**	**12**
4	**220**	**220**	**220**	**220**	**220**	**220**	530
5	**206**	**206**	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**	**322**	**322**	**322**
7	229	172	185	226	247	210	214
8	**186**	**186**	**186**	**186**	**186**	**186**	627
9	**436**	**436**	**436**	**436**	**436**	**436**	1020
10	**244**	**244**	**244**	245	**244**	**244**	935
13	N/A	1642	2270	1340	1332	1368	1310
16	N/A	N/A	N/A	N/A	1036	536	2755

Table 3: Results using a unique solver (base), two sequential solvers (sq2) and three sequential solvers (sq3), with and without Symmetry Breaking, with and without implied constraint

5 MIP Model

For the MIP model we choose the solver-independent language AMPL (https://ampl.com/). The principal idea of this model is to encode in a binary tensor the possible cycles traveled by each courier and impose that each courier has to travel through its delivery points through just one Hamiltonian cycle. We find a suitable idea to encode the upper idea in the article *Using AMPL/CPLEX to model and solve the electric vehicle routing problem (EVRP) with heterogeneous mixed fleet*[3].

5.1 Decision variables

The model is based on the following variables:

- A binary tensor $X \in \{0,1\}^{m \times (n+1) \times (n+1)}$, where $X[i,j,k] = 1$ iff the courier i departs from the *j-th* delivery point to arrive at the *k-th* delivery point. The *n+1* rows and columns refer to the origin.
- An array $T \in [1..n]^n$ that encodes the visit sequence of the various delivery points reached by the couriers. For example, if $T = [1,3,2,2,1]$ the items 1 and 5 are the first to be delivered by the couriers, items 3 and 4 are the seconds to be delivered by their couriers and item 2 is the last to be delivered by its courier.
- A natural number $Obj \in \mathbb{N}$ that represents the longest distance travelled by any courier. For this variable we set the upper bound $dist\_upper\_bound = \sum_{i=1}^{n+1} \max_{j=1..n+1} D[i,j]$, and the lower bound $obj\_lower\_bound = \max_{i=1..n} D[n+1,i] + D[i,n+1]$, the distance traveled by a courier to deliver the furthest item and to get back to the origin.

5.2 Objective function

In the model $Obj\_function$ is the objective function that the solver has to minimize. The function return the variable Obj.

5.3 Constraints

5.3.1 X

- The *one_arrival_per_node* constraint implies $\sum_{i=1}^{m} \sum_{j=1}^{n+1} X[i,j,k] = 1$ $\forall k = 1..n$, so that just one courier arrives at the *k-th* delivery point.
- The *one_departure_per_node* constraint implies $\sum_{i=1}^{m} \sum_{k=1}^{n+1} X[i,j,k] = 1$ $\forall j = 1..n$, so that just one courier departs from the *j-th* delivery point.
- The *origin_arrival* constraint implies that $\sum_{j=1}^{n+1} X[i,j,n+1] = 1$ $\forall i = 1..m$, so that every courier returns at the origin.

- The *origin_departure* constraint implies that $\sum_{k=1}^{n+1} X[i, n+1, k] = 1 \; \forall i = 1..m$, so that every courier starts from the origin.
- The *no_self_loop* constraint implies that $X[i, j, j] = 0 \; \forall i = 1..m \forall j = 1..n$, so that each courier can't leave and arrive at the same delivery point.
- The *balanced_flow* constraint implies that $\sum_{k=1}^{n+1} X[i, k, j] = \sum_{k=1}^{n+1} X[i, j, k]$ $\forall i = 1..m \forall j = 1..n$, so that if the *i-th* courier arrives at the *j-th* delivery point it has to depart from it.
- *load_capacity* constraint implies $\sum_{j=1}^{n+1} \sum_{k=1}^{n} X[i, j, k] * size[k] \leq capacity[i]$ $\forall i = 1..m$ where $size[k]$ represents the size of the *k-th* item and $capacity[i]$ represents the load of the *i-th* courier. In this way every courier must respects its load capacity.

5.3.2 *T*

For the next constraints we use the Big-M strategy in order to define *if-then-else* conditions in the AMPL language, we set $M = 2 * n$.

- The *first_visit* constraint implies that $T[k] \leq 1 + M * (1 - X[i, n+1, k])$ $\forall i = 1..m \forall k = 1..n$, so that for every courier the first element delivered, call it k, gets T[k]=1.
- The *successive_visit_1* and *successive_visit_2* constraints imply that $T[j] - T[k] \geq 1 - M * (1 - X[i, k, j]) \; \forall i = 1..m \forall k = 1..n \forall j = 1..n$ and $T[j] - T[k] \leq 1 + M * (1 - X[i, k, j]) \; \forall i = 1..m \forall k = 1..n \forall j = 1..n$. In this way if the *i-th* courier leaves the *k-th* delivery point and arrives at the *j-th* one, $X[i, j, k] = 1$, then $T[j] - T[i] = 1$, the *j-th* delivery point is visited exactly after the *k-th* one and can't return to an already visited point. Therefore for every courier loops are avoided in the path of the delivery.

5.3.3 *Obj*

- The *def_Obj* constraint implies $Obj \geq \sum_{j=1}^{n+1} \sum_{k=1}^{n+1} X[i, j, k] * D[j, k]$ $\forall i = 1..m$, so that *Obj* is equal or higher than the longest distance travelled by any courier.

5.3.4 Implied constraints

- The *implied_constraint* imposes that $X[i, n+1, n+1] = 0 \; \forall i = 1..m$, so that every courier transports at least one item.

5.3.5 Symmetry breaking constraint

- The *symmetry_breaking* constraint encodes the "Global loads ordering" discussed in section 1.2, substituting $load_i := \sum_{j=1}^{n} \sum_{k=1}^{n} X[i, j, k] * size[k]$. and applying the lexicographic constraint on couriers delivering same loads

on the respective last rows of X, namely $lex\_lesseq(X[i, n+1, ..], X[i+1, n+1, ..])$.

5.4 Validation

The model has been implemented in AMPL and run using several solvers: *HiGHS, COIN Branch and Cut (CBC), Gurobi, CPLEX*. For every solver we set the execution time to 300s.

Experimental design

For the validation part we run the MIP model on a *Dell XPS 15 9560, Intel Core i7-7700 CPU @ 2.8GHz, 4 Cores, 16GB RAM*, we used *HiGHS, COIN Branch and Cut (CBC), Gurobi, CPLEX* as solver. Afterword we run the MIP model with *HiGHS* and *Gurobi* with/without the implied constraint and with/without the symmetry breaking in order to evaluate their contribution.

Experimental results

ID	HiGHS	H_no_SB	H_no_impl	CBC	Gurobi	G_no_SB	G_no_impl	CPLEX
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	**167**	**167**	168	N/A	**167**	**167**	**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**
12	N/A	N/A	N/A	N/A	N/A	653	N/A	N/A
13	N/A	N/A	N/A	N/A	540	444	526	732
16	N/A	N/A	N/A	N/A	343	**286**	N/A	N/A
19	N/A	N/A	N/A	N/A	1040	334	N/A	N/A

Table 4: Results using HiGHS (H) and Gurobi (G), with and without Symmetry Breaking, with and without implied constraint; CBC and CPLEX.

6 Conclusion

In this project we implemented several models based on the 4 paradigms: CP, SAT, SMT and MIP. During the implementation of the models, we noticed that for the larger instances, removing the symmetry breaking constraint allowed the solver to find solutions when the complete model couldn't. This is easy to explain theoretically since the symmetry breaking constraint reduces the feasible region while maintaining unchanged the search space.

On the other hand, the removal of the implied constraint often caused a significant performance decrease.
We would like to emphasize how, for SAT and SMT, the 2/3 solvers approach yielded results where the base model clearly failed. Moreover, we concluded that the CP model was the best at solving the problem.

References

[1] C. Prins, P. Lacomme, and C. Prodhon, "Order-first split-second methods for vehicle routing problems: A review," *Transportation Research Part C: Emerging Technologies*, vol. 40, pp. 179–200, 2014.

[2] "Definition of quasimetric." `https://proofwiki.org/wiki/Definition:Quasimetric`.

[3] X. Zuo, C. Zhu, C. Huang, and Y. Xiao, "Using ampl/cplex to model and solve the electric vehicle routing problem (evrp) with heterogeneous mixed fleet," in *2017 29th Chinese Control And Decision Conference (CCDC)*, pp. 4666–4670, 2017.