Capacitated Vehicle Routing Problem (CVRP)

Francesco Baiocchi, Christian Di Buò, Simone Reale
{francesco.baiocchi2, christian.dibuo, simone.reale3}@studio.unibo.it

1 Introduction

This project delves into the *Capacitated Vehicle Routing Problem (CVRP)*, known in the literature to be NP-hard. The objective of this problem is to minimize the maximum distance traveled by any courier. For all the definitions provided in this work, we use the following notation:

- $COURIERS = 1, ..., m$
- $ITEMS = 1, ..., n$
- $NODES = 1, ..., n + 1$

After an initial study of the literature about the problem, heavily inspired by the book *"Vehicle Routing"* by P. Toth and D. Vigo [5], we found out some common features, that we identified as key points of the development of our solutions.

Boundaries Providing boundaries for the optimization process is crucial to address the search properly and speed up the solution extraction. In particular, the upper bounds and the lower bounds of the maximum distance are based on the distance matrix D and are the following:

$$lower\_bound = \max_{i \in ITEMS} (\ D_{n+1,\ i}\ +\ D_{i,\ n+1}\ ) \tag{1}$$

$$upper\_bound = \sum_{i \in NODES} \max_{j \in NODES} D_{i,\ j} \tag{2}$$

Couriers departure Given the objective function and the triangle inequality among the distances associated to nodes, if an optimal solution in which at least one courier stays in the depot exists, then there exists an equivalent solution in which each courier departs, as confirmed also by empirical results. Therefore, we developed our models on the assumption that each courier leaves the depot.

Miller-Tucker-Zemlin subtour elimination To gurantee that each courier follows an Hamiltonian cycle through the delivered items, we used the MTZ formulation, as stated in the paper "Integer Programming Formulation of Traveling Salesman Problems" [2].

We developed four different approaches, CP, SAT, SMT and MIP, proposing solver-independent solutions. First, we started collaborating to understand the problem with the CP formulation. Then, each member independently delved into a single approach: Simone Reale finished CP and worked on SAT, Francesco Baiocchi did SMT and SMT-LIB, while Christian Di Buò did MIP and managed the software structure. Almost all the models are set in an *incremental way*, starting from a base structure with the fundamental constraints of the problem and then adding implied and symmetry breaking ones.

We took approximately one month to complete the project, where the main difficulties found have been related to lack of documentation and optimizing the models for bigger instances.

2 CP

2.1 Decision variables

The CP models are based on the following decision variables:

- $x_{c,i} \in NODES$, with $c \in COURIERS$ and $i \in NODES$, which defines the construction of the paths for each courier, meaning that $x_{c,i} = j$ if the courier c goes from the node i to j, where $n + 1$ is the *depot*.
- $bins_i \in COURIERS$, with $i \in ITEMS$, where setting $bins_i = c$ means that the item i is assigned to the courier c.
- $load_c \in [min\_load \; .. \; max\_load]$, with $c \in COURIERS$, that is an *auxiliary variable* used to assign the load carried by each courier in the solution, where *min_load* and *max_load* are respectively the *lower bound* and the *upper bound* of the load values:

$$min\_load = min(s),\; max\_load = max(l).$$

2.2 Objective function

The *objective function* is defined through the distance matrix D given as input and the paths x and aims to minimize the following quantity:

$$\max_{c \in COURIERS} \left(\sum_{i \in NODES:\; x_{c,i} \neq i} D_{i,x_{c,i}} \right) \tag{3}$$

2.3 Constraints

The main problem structure is defined through the following constraints.

- `bin_packing_capa`

  This global constraint taken from the MiniZinc library ensures that the actual load carried by each courier, given by the sum of its item sizes, is lower than its overall capacity. Mathematically speaking, it is defined by:

  $$\sum_{\substack{i \in ITEMS: \\ bins_i = c}} s_i \leq l_c \quad \forall c \in COURIERS \tag{4}$$

  The global version has been preferred to the explicit one, due to a more optimized implementation provided by MiniZinc for performances.

- **Couriers assignment and Channelling constraint**

  It defines how the items must be assigned to the couriers and links the *bins* and x matrices in the following way:

  $$\begin{cases} x_{c,i} = i & \text{if } bins_i \neq c \\ x_{c,i} \neq i & \text{if } bins_i = c \end{cases} \quad \forall c \in COURIERS, \ \forall i \in ITEMS \tag{5}$$

- `subcircuit`

  Again, it is a MiniZinc global constraint, that models the fact that x is the *successors matrix of each node*, guaranteeing that each courier forms an Hamiltonian cycle through its nodes assignments. Therefore, this contraint is applied on each row of the x matrix.

- **Couriers departure**

  It states explicitly that each courier must take at least one pack and not stay in the depot, due to performance optimization:

  $$x_{c,n+1} \neq n+1 \quad \forall c \in COURIERS \tag{6}$$

2.3.1 Implied constraints

The implied constraint `bin_packing_load`, added to the model to improve the performance required the introduction of the variable *load*. It assigns the load of each courier as the sum of the items' sizes s, depending on the assignments in *bins* and is defined mathematically as:

$$\sum_{i \in ITEMS: \ bins_i = c} s_i = load_c \quad \forall c \in COURIERS \tag{7}$$

2.3.2 Symmetry breaking constraints

Experimental trials showed that too much symmetry breaking constraints cause the performance to worsen, so only one symmetry breaking about the load is left in the related model. In particular, it is the `lex_lesseq`, which imposes the lexicographic ordering on the couriers assignments of two items if the items have the same size and the couriers have the same load carried:

$$\begin{aligned} (s_i = s_j \land load[bins_i] = load[bins_j]) \implies lex\_lesseq([bins_i, \ bins_j]) \\ \forall i, j \in ITEMS \end{aligned} \tag{8}$$

2.4 Validation

2.4.1 Experimental design

The models are evaluated using as solvers both *Gecode* and *Chuffed*, with ad-hoc search strategies, in order to fully exploit the potential of each solver. After making different experiments on the variable assignation, it came up that x is the variable matrix that most influences the exploration of the search tree. However, different choices have been taken for Gecode and Chuffed, depending on the their availability of *variable selection* and *domain selection* strategies.

- **Gecode:**

  The exploration of the search tree is managed by the `int_default_search`, which is the type of integer search that is properly suited on the Gecode solver. After experimenting also `dom_w_deg` and `first_fail`, it turned out that the best annotation choices for this strategy are `afc_size_max` for variables, together with a random choice for the value assignment (`indomain_random`).

  In addition, a restart strategy has been added, based on the *Luby sequence*, in combination with the *Large Neighborhood Search (LNS)*. From previous experiments, we noticed that low values of the retain percentage did not influence much the exploration. Therefore, it has been set to the 80%, referring to the x matrix.

- **Chuffed:**

  The search strategy is split in two parts. In the beginning, the last column of x, containing the routes *depot-item*, is assigned in order to address the exploration starting from the first items to deliver. Then, the rest of the path is assigned consequently.

  The best variable selection strategy found is `first_fail` on both the steps explained above, combined with a restart policy that follows the *Luby sequence.*

The experiments were conducted on a MSI Prestige with an Intel i7-1280p chip, 16 GB of RAM, running Windows 11.

2.4.2 Experimental results

The results of our experiments are presented in Table 1. Overall, CP has achieved the best performances among all the approaches explored, succeeding in solving all the 21 instances of the problem with both solvers.

The results table shows that Gecode allows to reach better performances on average with respect to Chuffed, due to different solving engines. We can evince that adding implied constraints to the base model without setting properly the search strategy has not a significant effect on the outcomes. The main reason is that, independently on the constraints, the exploration of the search tree is more inclined to remain blocked in branches that do not lead to a valuable solution. Therefore, the restart policy represent a crucial improvement in this sense, especially in combination with LNS and a source of randomness. Although it worsen the performance in terms of time for basic problem instances, a high retain probability for the LNS is fundamental for bigger ones with the relaxation of a portion of the constraints, which enhance the exploration. This is confirmed by both the `implied_lns` and `symm` models, even if symmetry breaking constraints seem to not bring an improvement, probably causing to cut the exploration of valuable branches in the exploration.

On the other hand, Chuffed reaches similar performances with all the models, meaning that the restart policy in this case has a smaller impact. As in Gecode, also the additional implied and symmetry breaking constraints do not address the exploration towards better results, causing even to worsen the outcome in some cases, with respect to the base model.

Table 1: CP results. Values in **bold** are solved to optimality.

	Gecode				Chuffed		
Id	base	implied	implied_lns	symm	base	implied_rs	symm
1	**14**	**14**	14	14	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**	**226**	**226**	**226**
3	**12**	**12**	12	18	**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	**167**	**167**	**167**	**167**	**167**	**167**	**167**
8	**186**	**186**	**186**	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**	**244**	**244**	**244**
11	847	847	391	426	913	913	934
12	457	402	**346**	**346**	354	372	373
13	1316	1316	496	524	1102	1102	1102
14	1232	1224	736	782	1300	1298	1302
15	1088	1088	646	–	1016	1016	1018
16	496	496	**286**	**286**	**286**	**286**	**286**
17	1435	1467	932	–	1388	1397	1412
18	985	801	554	631	1124	1124	1124
19	476	476	**334**	**334**	**334**	**334**	**334**
20	1784	1803	902	–	1339	1340	1339
21	635	756	419	510	1114	1116	1086

3 SAT model

3.1 Decision variables

The problem has been modeled through three main Boolean matrices:

- $A_{c,i}$, with $c \in COURIERS$ and $i \in ITEMS$, for couriers assignments (`assignments` in the code), where $A_{c,i} = 1$ if the item i is assigned to courier c.
- $P_{c,i,j}$, with $c \in COURIERS$ and $i, j \in NODES$, for paths assignments (`paths` in the code), where $P_{c,i,j} = 1$ if the courier c goes from node i to node j.
- $U_{i,k}$, with $i, k \in ITEMS$, used for subtour elimination with the *MTZ formulation*, where $U_{i,k} = 1$ if the item i is delivered as the k-th in the related courier path.
- *max_distance*, which is a vector of dimension *n_bits*(*upper_bound*) containing the binary encoding of the maximum distance traveled by the couriers.

3.2 Objective function

The objective function takes into consideration the paths P to extract the values from the distance matrix D, described as:

$$\max_{c \in COURIERS} \left(\sum_{i,j \in NODES} P_{c,i,j} \; D_{c,i} \right) \tag{9}$$

3.3 Constraints

For all subsequent constraints, we decided to use the *Naive Pairwise formulation* for the `exactly_one` constraint, due to its simplicity and low computational cost.

Assignments and Path related constraints

- Each pack must be delivered and each courier must take at least one pack:

$$\texttt{exactly\_one}(A[..,i]) \quad \forall i \in ITEMS \tag{10}$$

$$Not(P_{c,\ n+1,\ n+1}) \quad \forall c \in COURIERS \tag{11}$$

- The start and the end of the couriers paths must be the depot:

$$\texttt{exactly\_one}(P[c, n+1, ..]) \quad \forall c \in COURIERS \tag{12}$$

$$\texttt{exactly\_one}(P[c, .., n+1]) \quad \forall c \in COURIERS \tag{13}$$

- Channelling constraints to link the A and P matrices:

$$A_{c,\ i} \implies \texttt{exactly\_one}(P[c, i, ..]) \wedge \texttt{exactly\_one}(P[c, .., i]) \ \wedge \tag{14}$$

$$A_{c,\ i} \implies Not(P_{c,\ i,\ i}) \ \wedge \tag{15}$$

$$Not(A_{c,\ i}) \implies (\ \forall j \in NODES:\ Not(P_{c,\ i,\ j}) \ \wedge \ Not(P_{c,\ j,\ i})\ ) \tag{16}$$

$$\forall c \in COURIERS,\ \forall i \in ITEMS$$

Load related constraints

- We decided to take advantage of the *Pseudo-Boolean constraint* PbLe in the Z3 library [1] to define the inequality between the load carried by each courier and its capacity, which ensures that:

$$\sum_{i \in ITEMS} A_{c,\ i} s_i \leq l_c \quad \forall c \in COURIERS \tag{17}$$

MTZ subtour elimination constraints

- Assignment of the ordering to the U matrix starting from the first items in the paths and then in sequential way:

$$P_{c,\ n+1,\ i} \implies U_{i,\ 1} \quad \forall c \in COURIERS,\ \forall i \in ITEMS \tag{18}$$

$$\begin{gathered}(P_{c,\ i,\ j} \ \wedge \ U_{i,\ k}) \implies U_{j,\ k+1} \\ \forall c \in COURIERS,\ \forall i, j, k \in ITEMS\end{gathered} \tag{19}$$

- Each pack must be delivered once:

$$\texttt{exactly\_one}(U[i, ..]) \quad \forall i \in ITEMS \tag{20}$$

- MTZ subtour elimination constraint [2]:

$$\begin{gathered}\forall i, j, k_1, k_2 \in ITEMS:\ (U_{i,\ k_1} \wedge U_{j,\ k_2}) \implies \\ k_1 + k_2 + 1 \leq (n-1)\ *\ (1 - \texttt{at\_least\_one}(P[.., i, j]))\end{gathered} \tag{21}$$

Distance related constraints

- *max_distance* has a restricted range of values between the *lower_bound* and an *upper_bound* that changes during the solving procedure (further details in Section 3.4.1).

3.3.1 Symmetry breaking constraint

The constraint is formally the same as in CP, which here in SAT is modeled as follows:

$$\begin{gathered}\left(s_i = s_j \ \wedge \ \textstyle\sum_{p_1 \in ITEMS} A_{c_1,\ p_1} s_{p_1} = \sum_{p_2 \in ITEMS} A_{c_2,\ p_2} s_{p_2}\right) \implies \\ (A_{c_1,\ i} \ \wedge \ A_{c_2,\ j} \ \wedge \ c_1 < c_2) \\ \forall i, j \in ITEMS, \forall c_1, c_2 \in COURIERS:\end{gathered} \tag{22}$$

3.4 Validation

3.4.1 Experimental design

We have conducted different experiments with the Z3 solver regarding the search strategy, with both the `Optimize` and `Solver` engines of the Z3Py library. In the end, we decided to use the second one to control the procedure as follows.

The optimization process is driven by an *Adaptive Binary Search* that changes the upper bound of the solution in an iterative behavior. In particular, it updates the upper bound of the maximum distance traveled according to the Binary Search algorithm, until the new upper bound value is in a large neighborhood of the lower bound. In that case, the Binary Search is aborted and the `Solver` is constrained to find a solution that is simply lower than the one in the previous iteration.

The experiments were conducted on a MSI Prestige with an Intel i7-1280p chip, 16 GB of RAM, running Windows 11.

3.4.2 Experimental results

The results are showed in the Table 2. As we can see, only for the first ten instances the model is able to find a solution, which is optimal for all of them, except for the `inst07`. For the remaining ones, the construction of the model itself requires more than 300 seconds. Regarding the quality of the results, what we can evince is that the speed advantage of the symmetry braking constraints is lower than the time spent to impose them. However, it is mostly related to the scalability, since for simple instances the outcome is the same and the bigger the instance and the worse is the performance.

Table 2: SAT results. Results in **bold** are solved to optimality.

Id	base	symm
1	**14**	**14**
2	**226**	**226**
3	**12**	**12**
4	**220**	**220**
5	**206**	**206**
6	**322**	**322**
7	187	216
8	**186**	**186**
9	**436**	**436**
10	**244**	**244**
11 - 21	–	–

4 SMT model

4.1 Decision variables

The SMT model is based on the logic of quantifier-free linear integer arithmetic and operates using the following decision variables:

- **paths** $\rightarrow$ a three-dimensional list of boolean variables of size $m \times n \times n$ such that:

$$paths_{c,i,j} = \begin{cases} \text{True} & \text{if the edge } i \rightarrow j \text{ is part of the path for courier } c \\ \text{False} & \text{otherwise} \end{cases}$$

- **order** $\rightarrow$ a list of integers of size n indicating the sequence in which items are delivered. This variable is used to implement subtour elimination via the Miller-Tucker-Zemlin formulation [2].

- **max_dist** $\rightarrow$ an integer variable representing the maximum distance covered among all couriers.

4.2 Objective function

The objective function is enforced by the *max_dist* variable:

$$\sum_{i \in NODES} \sum_{j \in NODES} paths_{c,i,j} \times distances_{i,j} \leq max\_dist. \quad \forall c \in COURIERS \tag{23}$$

4.3 Constraints

- **Domain of *order*:** each entry in *order* must lie between 0 and $n-1$. This ensures that no courier delivers more than n items:

$$order_i \;\geq\; 0 \;\wedge\; order_i \;\leq\; n-1 \quad \forall i \in ITEMS \tag{24}$$

- **Unique item delivery:** it ensures that every item is delivered exactly once.

$$\sum_{c \in \text{COURIERS}} \sum_{i \in NODES} paths_{c,i,j} = 1 \quad \forall j \in \text{ITEMS} \tag{25}$$

- **Path-flow constraint:** it guarantees that for every courier and item, the number of arrivals equals the number of departures.

$$\sum_{i \in NODES} paths_{c,i,j} = \sum_{i \in NODES} paths_{c,j,i} \quad \forall c \in COURIERS, \forall j \in \; ITEMS \tag{26}$$

- **Subtour elimination (MTZ):** it prevents internal cycles by ensuring a strict visitation order:

$$paths_{c,i,j} \implies order_i < order_j \quad \forall\, c \in COURIERS,\ \forall\, i,j \in ITEMS \tag{27}$$

- **Capacity constraint:** each courier c has a maximum load capacity l_c:

$$\sum_{i \in NODES} \sum_{j \in ITEMS} paths_{c,i,j} \ \times\ s_j \ \leq\ l_c \quad \forall\, c \in COURIERS \tag{28}$$

- **No self-loops:** no courier can have an edge $i \to i$:

$$\sum_{c \in COURIERS} \sum_{i \in NODES} paths_{c,i,i} \ =\ 0. \tag{29}$$

- **Each route must begin and end at the depot:** for each courier c, exactly one arc out of the depot and one arc into the depot:

$$\sum_{i \in ITEMS} paths_{c,n+1,i} = 1 \ \wedge \sum_{j \in ITEMS} paths_{c,j,n+1} = 1 \quad \forall\, c \in COURIERS \tag{30}$$

4.3.1 Symmetry breaking constraints

- **Symmetry breaking for couriers of equal capacity:** if two couriers c_1 and c_2 have the same *load* and $c_1 < c_2$, we fix a lexicographical order on the first visited item:

$$\begin{gathered} paths_{c_1,n+1,i} \ \wedge\ paths_{c_2,n+1,j} \implies i < j \\ \forall i,j \in ITEMS,\ \forall c_1, c_2 \in COURIERS \end{gathered} \tag{31}$$

  where $c_1 < c_2$ and $lc_1 = lc_2$.

4.4 Validation

4.4.1 Experimental design

After exploring various approaches, we focused our efforts on a plain arc-based formulation in Z3 (using Z3Py). We attempted to enhance performance by adding symmetry-breaking constraints but observed no significant improvements.
Building on our Z3Py model, we developed a Python module to generate SMT-LIB `.smt2` instances using Z3's `sexpr()` function. To optimize the search for the best solution, we implemented a binary search strategy on the `max_dist` variable. The process iteratively refines upper and lower bounds, dynamically adjusting the constraints within the `.smt2` file before re-solving.
The experiments were conducted on a MacBook Pro with an Apple M3 Pro chip, 18 GB of RAM, running macOS Sonoma 14.5.

4.4.2 Experimental results

The results of our experiments are presented in Table 3. For the first ten instances, the plain arc-based approach and the version with symmetry-breaking constraints successfully computed optimal solutions in just a few seconds. Neither approach was able to find solutions for the subsequent, larger instances within the imposed timeout.
Due to time constraints, only the first ten instances were converted to SMT-LIB `.smt2`; the remaining larger ones could not be converted within the given timeout.
Among these, CVC5 demonstrated the best performance.

Table 3: SMT results. Bold entries indicate instances solved to optimality.

	SMT-LIB (base)			z3py	
Id	z3	cvc4	cvc5	base	symm
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	-	397	422	**167**	**167**
8	**186**	**186**	**186**	**186**	**186**
9	-	**436**	**436**	**436**	**436**
10	-	246	**244**	**244**	**244**

5 MIP model

The MIP model is inspired by the *Three-index Vehicle-Flow formulation* [4].

5.1 Decision variables

- $x_{i,j,c}$ is a 3 dimensional Boolean variable, $x_{i,j,c} = 1$ if and only if the courier c moves from node i to node j and 0 otherwise.
- $y_{i,c}$ is a 2 dimensional Boolean variable, $y_{i,c} = 1$ if the courier c delivered the package i.
- $u_{i,c}$ is a 2 dimensional continuous variable, which represents the cumulated demand serviced by vehicle c when arriving at node i. It is used for the Miller-Tucker-Zemlin constraint.
- *maxdistance* is a continuous variable used to linearize the objective function. The objective is to minimize this variable.

- *$UpBound_i$* is a one dimensional continuous variable. It has been introduced to linearize the upper bound computation of the objective function.
- *LoBound* is an continuous variable. It has been introduced to linearize the lower bound computation of the objective function.

5.2 Objective function

The objective function is enforced by the *maxdistance* variable:

$$\sum_{i \in NODES} \sum_{j \in NODES} D_{i,j} \cdot x_{i,j,c} \leq maxdistance \qquad \forall c \in COURIERS \tag{32}$$

This impose an upper bound of the distance travelled by any courier, ensuring that no courier travels a distance grater than this variables.

5.2.1 Objective bounds

The bounds of *maxdistance*, discussed in Section 1, have been realized through the variables *UpBound* and *LoBound* and the following constraints to overcome non-linearity of the max function.

- Upper Bound definition

$$UpBound_i \geq D_{i,j} \quad \forall i, j \in NODES \tag{33}$$

- Lower Bound definition

$$LoBound \geq D_{n+1,i} + D_{i,n+1} \quad \forall i \in ITEMS \tag{34}$$

5.3 Constraints

We defined the following constraints:

- Each Item can be delivered by exactly one courier.

$$\sum_{c \in COURIERS} y_{i,c} = 1 \quad \forall i \in ITEMS \tag{35}$$

- The load carried by the courier must be lower than or equal to their capacity.

$$\sum_{i \in ITEMS} y_{i,c} \cdot s_i \leq l_c \quad \forall c \in COURIERS \tag{36}$$

- A courier cannot travel from one position back to itself.

$$x_{i,i,c} = 0 \quad \forall c \in \ COURIERS, \ \forall i \in NODES \tag{37}$$

- Any courier must start in the depot. The constraint that every courier must end in the depot is implied by this constraint and the path-flow.

$$\sum_{i \in NODES} x_{n+1,i,c} = 1 \quad \forall c \in COURIERS \tag{38}$$

- Path-flow constraint ensure that the number of times a vehicle enters a node is equal to the number of times it leaves that node.

$$\sum_{i \in NODES} x_{i,j,c} = \sum_{i \in NODES} x_{j,i,c} \quad \forall j \in ITEMS,\ \forall c \in COURIERS \tag{39}$$

- Channelling constraint to link the variables x and y.

$$\sum_{j \in NODES} x_{i,j,c} = y_{i,c} \quad \forall i \in ITEMS\ \forall c \in COURIERS \tag{40}$$

- MTZ constraint is nonlinear in its original form, it needs to be linearized using the Big-M trick, resulting in the following linear inequality:

$$\begin{gathered} u_{i,c} - u_{j,c} + 1 \leq n \cdot (1 - x_{i,j,c}) \\ \forall c \in COURIERS,\ \forall i,j \in ITEMS,\ i \neq j \end{gathered} \tag{41}$$

5.3.1 Implied Model

For the implied model, we added the constraint that every courier must deliver at least one package.

$$\sum_{i \in ITEMS} y_{i,c} \geq 1 \quad \forall c \in COURIERS \tag{42}$$

5.3.2 Symmetry Model

For the symmetry model, we added the symmetry breaking constraint related to the couriers' capacity:

$$\begin{gathered} \sum_{i \in ITEMS} y_{i,c1} \leq \sum_{i \in ITEMS} y_{i,c2} \\ \forall c1, c2 \in COURIERS,\ c1 < c2,\ l_{c1} = l_{c2} \end{gathered} \tag{43}$$

5.4 Validation

5.4.1 Experimental design

For the MIP models, we choose to use the solver-independent language AMPL [3]. The workflow is based on the construction of three different models: the initial one, the implied one, and the implied enhanced with symmetry breaking.

We decided to use three of the solvers provided by the AMPL framework HiGHS, SCIP, and Gurobi. The versions of the solvers were as follows: HiGHS 1.8.1, SCIP 9.0.1 and Gurobi 12.0.0. Gurobi and Highs options where setted to use only 1 thread.

The experiments were conducted on a MacBook Pro with an Apple M1 Pro chip, 16 GB of RAM, running macOS Sequoia 15.1.1.

5.4.2 Experimental results

In Table 4, we present the results of the MIP models. We can observe that the first ten instances are always solved optimally. The implied model with symmetry breaking improves the performances in the SCIP solver, while perform poorly on Highs. Overall the implied model seems to have the best performances. About the three solvers we can notice that Gurobi outperform the other solvers, finding an optimal solution also in the instance number 12, while SCIP seems to be the one with the worst results.

Table 4: MIP results. Results in **bold** are solved to optimality.

	SCIP			HiGHS			Gurobi		
Id	base	implied	impl + sb	base	implied	impl + sb	base	implied	impl + sb
1	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**	**226**	**226**	**226**	**226**	**226**
3	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
4	**220**	**220**	**220**	**220**	**220**	**220**	**220**	**220**	**220**
5	**206**	**206**	**206**	**206**	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**	**322**	**322**	**322**	**322**	**322**
7	**167**	**167**	**167**	**167**	**167**	**167**	**167**	**167**	**167**
8	**186**	**186**	**186**	**186**	**186**	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**	**436**	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**	**244**	**244**	**244**	**244**	**244**
11	–	–	–	–	–	–	526	550	537
12	–	1590	–	677	494	–	**346**	373	376
13	680	792	612	560	540	558	444	432	454
14	–	–	–	–	–	–	–	–	–
15	–	–	–	–	–	–	838	–	–
16	–	621	614	**286**	**286**	305	**286**	**286**	**286**
17	–	–	–	–	–	–	–	–	–
18	–	–	–	–	–	–	633	568	–
19	–	–	654	426	509	629	**334**	**334**	**334**
20	–	–	–	–	–	–	–	–	–
21	–	–	1825	550	–	–	548	611	–

6 Conclusions

We experimented several models and starting from the same core idea we tried to enhance it depending on the approach. Overall, all approaches are able to optimally solve the smaller instances while, for bigger ones, only CP and MIP were able to at least produce a suboptimal solution. CP achieves the best results, thanks to global constraints and LNS relaxation. We noted that symmetry breaking constraint generally tend to not improve our results, and in some cases, surprisingly worsen them.

References

[1] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. “A Translation of Pseudo Boolean Constraints to SAT”. In: *JSAT* 2 (Mar. 2006), pp. 191–200. DOI: `10.3233/SAT190021`.

[2] C. E. Miller, A. W. Tucker, and R. A. Zemlin. “Integer Programming Formulation of Traveling Salesman Problems”. In: *J. ACM* 7.4 (Oct. 1960), pp. 326–329. ISSN: 0004-5411. DOI: `10.1145/321043.321046`. URL: `https://doi.org/10.1145/321043.321046`.

[3] David M. Gay Robert Fourer and Brian W. Kernighan. In: *AMPL: A Modeling Language for Mathematical Programming.* 1993. URL: `https://ampl.com/learn/ampl-book/`.

[4] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem.* Ed. by Paolo Toth and Daniele Vigo. 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`.

[5] Paolo Toth and Daniele Vigo. *Vehicle Routing.* Ed. by Daniele Vigo and Paolo Toth. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2014. DOI: `10.1137/1.9781611973594`. eprint: `https://epubs.siam.org/doi/pdf/10.1137/1.9781611973594`. URL: `https://epubs.siam.org/doi/abs/10.1137/1.9781611973594`.