Multiple Couriers Planning

Stefano Colamonaco, stefano.colamonaco@studio.unibo.it
Samuele Marro, samuele.marro@studio.unibo.it
Andrea Zecca, andrea.zecca3@studio.unibo.it

July 2023

1 Introduction

In this report, we will discuss the Multiple Courier Planning (MCP) problem and show how it is possible to solve instances of this task efficiently. The MCP problem is a well-known problem in the literature and in the optimization field. We have m couriers and n packages located at n cities. Each courier i has a load size l_i and each package j has a weight s_j. The aim of the problem is to decide a consistent assignment of the packages to the courier, such that each courier does not exceed its load size, while minimizing the maximum distance travelled by each courier. The MCP problem is closely related to other well-know optimization problems such as the Multiple Travel Salesmen Problem (MTSP)[1] and the Capacitated Vehicle Routing Problem (CVRP)[2, 4]. We can consider the MCP problem as a combination of the aforementioned problems because we have both the presence of multiple couriers (like in MTSP, where we have several salesmen) and each courier has limited capacity (like in CVRP).
An important characteristic of the MCP problem is given by the distance matrix being a *quasimetric* function [3], i.e. a function satisfying the following axioms:

- $\forall x, y \;\; d(x, y) \geq 0$
- $\forall x, y \;\; d(x, y) = 0 \iff x = y$
- $\forall x, y, z \;\; d(x, y) \leq d(x, z) + d(z, y)$

We can exploit the nature of the distance function, and in particular the last axiom (referred to as *triangular inequality*) in order to compute some bounds on the minimum distance which are useful to speed up the search of an optimal solution (see Section 2).
We approached this problem using different techniques, collecting the results obtained on instances of increasing difficulty (see Sections 3 to 6).
When testing our models, the instances are subject to a pre-processing phase (see Section 2), whose objective is to extrapolate useful information from the instance being tested, e.g. the minimum and maximum distance a courier can travel or the minimum and maximum load a courier can transport.
The models have been tested on Docker with the following configuration:

- CPUs: 8

- Memory: 3.8 GB
- Swap: 1GB
- Virtual disk limit: 64 GB

The Docker container was run on a PC with an Intel Core i5-8300H 8-cores, which has a clock speed of 2.30 GHz.
We did not follow a strict division of the workload, but rather organized ourselves such that all members contributed to essentially all aspects of the project.
The main obstacles we had to overcome during the development of the project were the following:

- Finding good bound for all the auxiliary variables;
- Representing integers in SAT;
- Removing spurious cycle in MIP;
- Translating and executing the SMT model in the SMT-LIB language.

2 Preprocessing

Before running a model on a given instance, we compute a series of auxiliary values that are used to bound either the value or the size of certain variables. These values are:

- **at_least_one**: 1 if each courier necessarily carries at least one package, 0 otherwise.
- **min_dist** and **max_dist**: The minimum and maximum distance travelled by *each* courier.
- **min_solution**: The minimum objective value.
- **time**: The maximum number nodes that any courier will visit.
- **min_load** and **max_load**: The minimum and maximum load carried by all couriers.

2.1 Computing the Additional Parameters

at_least_one The definition of *at_least_one* relies on two steps:

- Determining if each courier can carry a distinct package;
- Proving that if each courier can carry a distinct package and there exists a solution for the problem, then there exists an optimal solution where each courier carries at least one package.

The first part can be checked by assigning the packages to the couriers in simultaneous descending order (i.e. the heaviest package to the courier with the highest capacity, the second-heaviest package to the courier with the second-highest capacity, and so on). The second part is a consequence of the triangular inequality. Specifically, we want to prove that, if there exists an optimal solution, then there exists an optimal solution where each courier delivers at least one package.
We will prove this theorem by induction. Our inductive statement is “If there exists an optimal solution with $p < m$ couriers delivering at least one package, then there exists an optimal solution with $p + 1$ delivering at least one package.

Let P_p^* be an optimal solution with p couriers delivering at least one package and consider a courier i_2 that is not currently delivering any packages. Since we know that each courier can deliver at least one package, consider a package j (currently delivered by the courier i_1) whose load does not exceed the capacity of i_2: due to the triangular inequality, removing a node from the path of i_1 cannot increase its length. Additionally, recall that i_1 and i_2 both start from the depot d; therefore, due to the triangular equality, the original cycle of i_1 (which contains d and j) cannot be shorter than the new cycle of i_2 (which *only* contains d and j). In other words, we have found a new solution P_{p+1}^* which does not have a higher maximum distance than P_p^*, and since P_p^* is optimal, P_{p+1}^* is optimal too.

time *time* is the minimum of two values:

- The first is $n - m + 2$ if *at_least_one* is 1 and $n + 1$ otherwise. The value for *at_least_one* $= 1$ is due to the fact that, given a certain courier, at least $m - 1$ packages will be carried by the other couriers (one for each courier)
- The second is $1 +$ *max_package_count*, where *max_package_count* is the maximum number of packages that can be delivered by a courier (which is computed by progressively assigning the smallest remaining package to the courier with the highest capacity)

min_load *min_load* is defined as $\min_j s_j$ if *at_least_one* is 1 and 0 otherwise

max_load *max_load* is defined as the minimum between the maximum load and the sum of the *time* heaviest packages.

min_dist *min_dist* is defined as follows:

- If *at_least_one* is 1, *min_dist* is the minimum distance to either go from the depot to a package or from a package to the depot, times two
- If *at_least_one* is 0, *min_dist* is 0.

max_dist *max_dist* is defined as follows:

- If *at_least_one* is 0, *max_dist* is the sum over all nodes j (including the depot) of the maximum distance to reach the node j
- If *at_least_one* is 1, *max_dist* is the minimum between the previous value and the sum of the *time* $+ 1$ highest distances between any two nodes.

min_solution *min_solution* is defined as the maximum of the distance to travel from the depot to a node and back. In fact, each package will be delivered, and the length of a cycle containing the depot d and a node j will be at least as high as the length of a cycle containing only d and j (due to the triangle inequality).

3 CP Model

3.1 Decision Variables

We used the following decision variable:

- **x** is a 2-dimensional array of size $[1..m, 1..n+1]$ which values are in the interval $1..n+1$, with the meaning that $x[i, j_1] = j_2$ if and only if the i-th courier travel from the place j_1 to j_2. Is important to notice that $x[i, j_1]$ is equal to j_1 when the courier does not pass through it. The value $n+1$ refers to the depot (initial and ending point of the route for each courier).

3.1.1 Auxiliary Variables

In addition to the decision variables, we introduced the following auxiliary variables:

- **y** is an array of size $[1..m]$ whose values are in the interval *min_dist..max_dist*. This variable contains, for each courier, the total travelled distance.
- **effective_max_dist** represents the true value that we want to minimize when solving the problem under analysis, i.e. the maximum value of the vector y. The reason behind defining this variable is that it allows us to assign bounds that are different from those relative to the single elements of the vector y. The variable is defined in the range *min_solution..max_dist* and constrained to be equal to $\max(y)$.
- **load** is an array of size $[1..m]$ with values in the range *min_load..max_load* which represents the weight transported by each of the couriers.
- **numpacks** is an array of size $[1..m]$ with values in the range *at_least_one..time* which is useful to limit (both above and below) the number of packs that a courier can transport, in order to reduce the search space.
- **bins** is an array of size $[1..n]$ with values in the range $1..m$ that simply maps the packages to the courier they are assigned to. This is a tunneling variable used to exploit bin packing-related global constraints in the model.

3.2 Objective Function

The objective function is represented by the following expression:

$$\text{minimize } \max_i \sum_{j_1=1}^{n+1} distance(j_1, x_{i,j_1}) \tag{1}$$

3.3 Constraints

The following is an exhaustive list of the constraints of the model:

- **Each courier starts and ends at the depot:** Fundamental to comply with the description of the problem and correctly calculate the distance travelled.

- **Couriers cannot exceed their max load:** The load carried by a courier is calculated as the sum of the weights of all the packages it transports:

$$\sum_{j_1=1}^{n} (\text{if } x_{i,j_1} = j_1 \text{ then } 0, \text{ else } s_{j_1}) \tag{2}$$

  This value must be less than or equal to the maximum weight that the courier can carry.
- **Definition of the path for each courier:** This is probably the most important constraint, since it allows us to easily define for each courier in which order the packages should be delivered. It was implemented using the global constraint *subcircuit*().
- **Each package is delivered exactly once:** Following the definition of the problem, each place other than the depot must be visited exactly once. In our model this constraint is easily implemented by forcing the number of occurrences of $x[i, j_1] \neq j_1$ to be exactly one.
- **Assignment of packages according to weight:** To maximize performance in the selection of packages that each courier must transport, the global constraint *bin_packing_load*() was used.
- **Construction of the distance vector:** With this constraint, the vector containing the distances traveled by the couriers according to the considered route is constructed (following the formula (1) defined for the objective function).

Symmetry-Breaking Constraints In order to reduce the search space, we also introduced the following symmetry-breaking constraints:

- **Constraint between couriers who have the same maximum load capacity:** If two couriers can carry the same load, it means that they are interchangeable; if this situation is common, the search space ends up larger than necessary. For the implementation, the local constraint lex_lesseq() was used on the arrays identifying each pair of couriers having the same maximum transportable load.
- **Constraint on loads carried by couriers:** Considering two couriers A and B, where the maximum load transportable by A is greater than or equal to that transportable by B, then the load actually transported by A must be greater than or equal to the one actually transported by B.

3.4 Validation

Experimental Design Before the development of the current model, we built several alternative models for CP. Unfortunately, these other models were found to be significantly underperforming. We also tested every model using different search strategies both for the choosing of the next variable and for the next value in the domain of a variable to test, until we found the best search strategies for our model, which are:

1. *first_fail*: Choose the variable with the smallest domain size;

2. *indomain_min*: Assign the variable its smallest domain value.

We tested our model on two different solvers, **Gecode** and **Chuffed**; each one was tested with and without symmetry-breaking constraints. Additionally, we set a time limit of 300 seconds (5 minutes) for each test.

Experimental Results

Instance	Gecode	Gecode + SB	Chuffed	Chuffed + SB
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	211	**167**	211	**167**
8	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**
11	-	-	807	471
12	-	716	722	470
13	654	650	642	646
14	-	-	1250	-
15	-	-	-	-
16	956	314	827	**286**
17	-	-	-	-
18	-	-	1058	820
19	1231	627	1692	736
20	-	-	-	-
21	-	-	1264	1066

Table 1: Results on the CP model using Gecode and Chuffed with and without symmetry breaking using *first_fail* and *indomain_min*.

4 SAT Model

The construction of this model was particularly challenging, since Multiple Courier Planning is not a problem that lends itself particularly well to an entirely binary encoding. We implemented two different models with this paradigm: one using only low-level Boolean constraints, and one using a pseudo-Boolean helper function provided by the Z3 library. Pseudo-Boolean representations typically allow both linear inequalities and linear equalities over Boolean variables, which significantly improve the efficiency of the model. Furthermore, since the library used does not provide an entirely Boolean optimizer (since SAT is purely about satisfiability), it was necessary to implement a binary search algorithm in the space of the objective function to find the solution. In order to use the binary search without running the model from scratch every time we found a new bound, we used a mechanism implemented in Z3 library of push/pop. This paradigm creates a sequence of *local scopes*: assertions added within a *push* are retracted on the matching *pop*. The implementation of the binary search, apart from push/pop mechanism, follows standard practices and does not require an extensive study.

4.1 Decision Variables

We used the following decision variable:

- **v** is a 3-dimensional array of size $[1..m, 1..n+1, 0..time]$ such that $x[i, j, k] = 1$ if and only if the i-th courier is in position j at time k.

4.1.1 Auxiliary Variables

In addition to the decision variables we introduced the following auxiliary variables:

- **d** is a 3-dimensional array of size $[1..m, 1..n+1, 1..n+1]$ with the meaning that $x[i, j_1, j_2] = 1$ if and only if the i-th courier travel from j_1 to j_2.

4.2 Objective Function

Since SAT can only determine the satisfiability of a given problem (and thus cannot be used directly to solve an optimization query), formally assigning an objective function is an ill-defined task. However, in practice the objective function is the same as the one of other models, since the binary search ends up optimizing the maximum distance of any courier.

4.3 Constraints

Similarly to the other models, constraints encode the true complexity of the model. As we have already mentioned, we created two models that differ only on the implementation of the constraints. The first model makes extensive use of the pseudo-Boolean library functions; on the other hand, for the second model we tried to simulate the behavior of the library functions by representing each integer parameter with its one-hot encoding.
The main idea behind this model stems from a careful observation and simulation of how pseudo-Boolean constraint could be more or less efficiently implemented in a naive way. In particular, we focused our attention on the pseudo-Boolean constraints PbLe and PbGe, which both take as input a vector of tuples consisting in a Boolean variable x, a weight s and a value k. The former constraint formalizes the linear inequality $\sum_i x_i \cdot s_i \leq k$, while the latter $\sum_i x_i \cdot s_i \geq k$.

In order to recreate the behavior of the aforementioned constraints, we had to introduce several new variables to create a sort of one-hot encoding of each parameter. We will explain the workflow with an easy example.
Suppose that we have the following variables:

- **x**: a one-dimensional array of Boolean values of size n
- **s**: a one-dimensional array of integer values of size n
- **k**: an integer value

and we want to simulate the behavior of the constraint PbLe on x, s and a value k. We will apply the following steps:

1. For each pair (x_i, s_i), we introduce s_i new variables (one-hot encoding of s_i), named $N_{i,k}$ with $k \in \{1, \ldots, s_i\}$
2. We introduce a new constraint linking the newly created variables: $N_{i,1} \iff N_{i,2} \iff \cdots \iff N_{i,s_i}$
3. Finally, we can use the function *at_most_k* on the variables $N_{i,j}$ and the value k

In this way, we can simulate the behavior of both PbLe and PbGe (using *at_least_k*), although we pay a price in terms of efficiency and memory occupancy.
We now provide an exhaustive list of the constraints used. We will only focus on the part related to the implementation without pseudo-Boolean functions, since we think that they can provide more insight into the rationale.

1. **Each courier does not exceed its capacity:** To formalize the fact that a courier cannot transport a weight greater than $l[i]$, we set that the value of $x[i, j, k]$ repeated for $w[j]$ times of each of the parcels transported by the i-th courier must be less than its maximum transportable weight. This can be summarized by the following formulation:

$$\forall_i^m \text{at\_most\_k} \left(\bigwedge_{j=1}^{n} \bigwedge_{k=1}^{time} \bigwedge_{x=1}^{w[j]} v[i, j, k] \ , \ l[i] \right) \tag{3}$$

   (The $\bigwedge$ must be understood as a concatenation of truth values, as if we were constructing a clause).

2. **Each courier starts and ends at the depot:** This constraint is easy to implement, since we only need to force the conjunction between the variables encoding the position of each courier at the first and last time step, having the location equal to the depot.

$$\forall_i^m \bigwedge (v[i, n+1, 0], v[i, n+1, time]) \tag{4}$$

3. **Each courier cannot be in two places at the same time:** This is a constraint that is not necessary in other models due to a better formulation of the decision variable.

4. **Each package is delivered exactly once:** This constraint is simply a different implementation of the one described in the section Section 3.3 with the same purpose.

5. **Limit to the maximum distance traveled by a courier:** To formalize that a courier cannot travel a distance greater than $max\_distance$, we set that, for each courier i, the value of $d[i, j_1, j_2]$ repeated for $distances[j_1, j_2]$ times of each of the parcels transported by the courier must be less than the current maximum distance. This can be summarized by the following formulation:

$$\forall_i^m \text{at\_most\_k} \left(\bigwedge_{j_1=1}^{n+1} \bigwedge_{j_2=1}^{n+1} \bigwedge_{x=1}^{distances[j_1,j_2]} d[i, j_1, j_2], max\_distance \right) \quad (5)$$

   (The $\bigwedge$ must be understood as a concatenation of truth values, as if we were constructing a clause)

Of these constraints, the numbers 1,5 were represented in both pseudo-Boolean and standard versions. For the implementation of some of the constraints mentioned above, we used the functions *at_most_one()*, *at_least_one()*, *exactly_one()* and *at_most_k()*, respectively with Bit-Wise and Sequential encoding. We chose these encoding over the others available for performance reasons.

Symmetry-Breaking Constraints We also defined some symmetry-breaking constraints, although they were not used in the final version of the model since they ended up degrading its performance. The most likely explanation is that the computational overhead added by these constraints is greater than the advantage they bring to the search. The constraints are the following:

- **Once a courier has returned to the depot, it can no longer start again from there:** This constraint considerably reduces the search space, as the use of $n + 1$ (depot) is not limited by the previous constraints and could lead the couriers to start from the depot in a staggered way or to return there and then start again.
- **Constraint between couriers who have the same maximum load capacity:** This constraint is simply an alternative implementation of the one described in the section Section 3.3 with the same purpose.

4.4 Validation

Experimental Design The experimental setup is fully described in the previous paragraphs. In order to test the SAT models (both with and without pseudo-Boolean constraints), we used the Z3Py library, which supports SAT theories. We decided to use this approach due to the fact that the generation of the model sometimes can last more than 300 seconds on its own. Therefore, we used a soft timeout (passed to the model and computed as the residual time after the generation of the model) and a hard timeout (which kills the process if it overshoot the soft timeout). To limit the time spent during the test, we used a timeout of 300 seconds, which also includes the time spent generating the model.

Experimental Results

Instance	w/out PB	With PB Functions
1	**14**	**14**
2	-	**226**
3	**12**	**12**
4	-	**220**
5	206	**206**
6	-	**322**
7	-	195
8	-	**186**
9	-	**436**
10	-	**244**
11	-	-
12	-	-
13	-	1480
14	-	-
15	-	-
16	-	589

Table 2: Results using the standard SAT model and adding pseudo-Boolean functions. Instances beyond the 16th were omitted due to neither solver finding a solution.

5 SMT Model

5.1 Decision Variables

We used the following decision variable:

- **x** is a 2-dimensional integer array of size $[1..m, 0..time]$. The values of the array are then constrained in the interval $1..n+1$ in such a way that the meaning of the variable is the following: $x[i,k] = j$ if and only if the courier i is in position j at time k.

5.1.1 Auxiliary Variable

In addition to the decision variable, we introduced the following auxiliary variables:

- **y** is an integer array of size $[1..m]$ which represents the distance travelled by each of the couriers. The values of the array are then constrained in the interval $1 min\_dist..max\_dist$.
- **load** is an integer array of size $[1..m]$ which represents the weight transported by each of the couriers. The values of the array are then constrained in the interval $1 min\_load..max\_load$.
- **max_distance** is a variable representing the true value that we try to minimize while solving the problem under analysis. It is defined as IntVal() and constrained in the range $min\_solution..max\_dist$. As in the previous models, it is necessary to define this variable in order to assign some bounds in the optimization of the objective function.

Furthermore, some of the variables supplied as input have been redefined using the Z3 typing in order to favor indexing using the decision variable.

5.2 Objective Function

The objective function is:

$$\text{minimize } max\_distance = \max_i y_i \tag{6}$$

which is equivalent to:

$$\text{minimize } \max_i \sum_{k=0}^{time} distances(x_{i,k}\,, x_{i,k+1}\,) \tag{7}$$

5.3 Constraints

- **Each package is delivered exactly once:** To implement this constraint, we counted the total number of occurrences of each package in x and set this number to 1.
- **Each courier does not exceed its capacity**: The load of the i-th courier is computed as the sum of the weights of the packages assigned to it.

The other constraints are related to the bounding of the decision and auxiliary variables.

Symmetry-Breaking Constraints To reduce the search space as much as possible, we tried adding symmetry-breaking constraints, but similarly to other models, their overall effected was negative. These symmetry-breaking constraints were the same as those in the SAT model (see Section 4.3), with the addition of “Constraint on loads carried by couriers”, which follows the same rationale as the homonymous constraint in Section 3.3.

5.4 Validation

Experimental Design We built various alternative models for SMT, but for the sake of brevity we reported only the best-performing one. Similarly to other models, in this case we used a soft and a hard timeout, since the generation of some models lasted for more than 300 seconds. One of the factors that led us to implement the model using Z3Py is the fact that it provides APIs to transform the instance into the corresponding SMT-LIB2 version (through the *sexpr()* function). We then converted the model into its SMT-LIB2 version and ran a binary search on it using a Python script which, starting from an initial upper bound of the objective value, inserts a new constraint on the objective variable injecting SMT-LIB2 code and tries to solve the obtained model. Every time we succeed in solving a model, we store the intermediate results and update the bound of the objective value and solve again the model, until we cannot proceed further.

Experimental Results

Instance	Z3OPT	Z3OPT+SB	SMTLIB-CVC4+SB	SMTLIB-Z3+SB
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**	**322**	**322**
7	184	**167**	-	205
8	**186**	**186**	-	**186**
9	**436**	**436**	-	**436**
10	**244**	**244**	-	**244**

Table 3: Results using the SMT model both in Z3Py and SMT-LIB2. For SMT-LIB version only model with symmetry breaking has been considered. Instances beyond the 10th were omitted due to neither solver finding a solution.

6 MIP Model

6.1 Decision Variables

- **x** is a 3-dimensional Boolean array of size $[1..m, 1..n+1, 1..n+1]$ such that $x[i, j_1, j_2]$ is equal to 1 if and only if the i-th courier travels from $j1$ to $j2$

6.1.1 Auxiliary Variables

- **y** is an integer array of size $[1..m]$ such that $y[i]$ is the total distance travelled by the i-th courier. The variables are in the range $min\_dist..max\_dist$.

- **t** is a 2-dimensional Boolean array of size $[1..m, 1..n]$ such that $t[i, j]$ is true if and only if the i-th courier carries the j-th package.
- **z** is a special 2-dimensional integer array of size $[1..m, 1..n+1]$ that, for each courier i, records its "path counter". For example, if the i-th courier visits, in order, j_1, j_2 and j_3, then we will have $z[i, j_1] = 1$, $z[i, j_2] = 2$ and $z[i, j_3] = 3$.
- **v** is an integer variable that is equal to the maximum distance travelled by any courier. The variable is in the range $min\_solution..max\_dist$.

6.2 Objective Function

The objective function is:

$$\text{minimize } z = \max_i y_i \tag{8}$$

which is equivalent to:

$$\text{minimize } \max_i \sum_{j_1, j_2 = 1}^{n+1} distance(j_1, j_2) \cdot x_{i,j_1,j_2} \tag{9}$$

6.3 Constraints

- **A courier cannot travel from a node to itself**: This constraint can be easily implemented by setting $x[i, j, j] = 0$ for all $i \in 1..m, j \in 1..n+1$.
- **Each courier moves from and to the depot exactly once**: This is true even if the courier does not carry any packages, since in this case the movement is from the depot to itself.
- **Each courier does not exceed its capacity**: The load of the i-th courier is computed as the dot product of $t[i, :]$ and s.
- **Computation of t[i, j]**: Self-explanatory.
- **Computation of y[i]**: Self-explanatory.
- **All packages are carried by exactly one courier**: Self-explanatory.
- **If a courier arrives at a package, it must leave from it**: This constraint, coupled with the "each courier moves from and to the depot exactly once" one, ensures that there will be a cycle containing the depot (but does not ensure that it will be the only cycle)
- **The path counter is 0 for the depot**: Essentially further bounds the path counter, since in practice the maximum path counter will be equal to the maximum length of a courier path.
- **Anti-cycle constraint**: We define a path counter by implementing the constraint "If a courier i travels from j_1 to j_2, then $z[i, j_2] = z[i, j_2] + 1$. This trivially prevents any cycles, since there would always be one node that has a higher path counter than its successor. Note that this constraint is not present for the depot, thus allowing cycles that contain the depot.
- **v is an upper bound of all y[i]**: This constraint allows us to define v as $\max_i y[i]$ using only linear constraints.

6.4 Validation

Experimental Design We implemented the model using the Python library *python-mip*, which natively supports CBC and Gurobi and has an intuitive API. We tested our model on a Docker container using the CBC solver. For what concerns the tests with the Gurobi solver, we were able to only test it locally, since Gurobi requires an academic license which is not meant to be used in standalone Docker containers. Similarly to previous models, we set both a soft timeout (passed to the solver) and a hard timeout (which kills the process if it overshoots the soft timeout). This decision is motivated by the fact that the model building can phase can sometimes take more than 300 seconds.

Experimental Results

Instance	CBC	Gurobi (local)
1	**14**	**14**
2	**226**	**226**
3	**12**	**12**
4	**220**	**220**
5	**206**	**206**
6	**322**	**322**
7	**167**	**167**
8	**186**	**186**
9	**436**	**436**
10	**244**	**244**
11	-	-
12	-	-
13	-	534
14	-	-
15	-	-
16	-	**286**

Table 4: Results on MIP model using CBC and Gurobi solvers. Instances beyond the 16th were omitted due to neither solver finding any solution.

7 Comparison of Results

We report a chart detailing the behavior of the best-performing models of their paradigm, which shows the completion time for each of the first 10 instances available. Note how the most complicated instance in this set is number 7; in fact, this instance represents the junction point between the more trivial and the more complex instances and was our point of reference in the testing phase.

Instances 11 to 21 are very complex and only a few models have managed to find solutions for them, even if they are often not optimal.

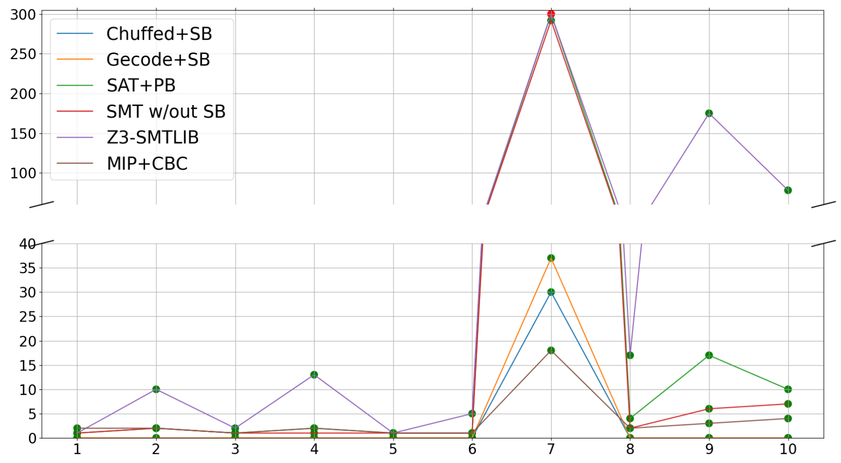

Figure 1: Comparison of the best-performing models in their paradigm on the first 10 instances.

8 Conclusions

The models presented show how it is possible to solve the Multiple Courier Problem using very different paradigms and approaches, achieving good and rather homogeneous levels of performance. The results of this project confirm what was explored during the lessons and known in the literature about the construction of models using the paradigms of Constraint Programming, SAT, Satisfiability Modulo Theory and Mixed Integer Programming. In particular, we noticed the following phenomena:

- Symmetry-breaking constraints are effective in some cases, but in other situations their impact in terms of computational complexity is higher than the effective gain obtained from the reduction of the search space;
- The use of global constraints and library functions was essential to maximize efficiency;
- The use of multiple solvers with different characteristics was interesting as it showed how following a different research approach, even if applied to the same model, can lead to different results.

Future work could expand upon our findings by testing with other (potentially commercial) solvers and by using more advanced heuristics to further bound the value of the variables.

References

[1] Omar Cheikhrouhou and Ines Khoufi. "A comprehensive survey on the Multiple Traveling Salesman Problem: Applications, approaches and taxonomy". In: *Computer Science Review* 40 (2021), p. 100369. ISSN: 1574-0137. DOI: `https://doi.org/10.1016/j.cosrev.2021.100369`. URL: `https://www.sciencedirect.com/science/article/pii/S1574013721000095`.

[2] V Praveen et al. "Vehicle routing optimization problem: a study on capacitated vehicle routing problem". In: *Materials Today: Proceedings* 64 (2022), pp. 670–674.

[3] *Quasi-metric function.* URL: `https://encyclopediaofmath.org/wiki/Quasi-metric#:~:text=A%20function%20d%3AX%C3%97,is%20called%20a%20quasi%2Dmetric.` (visited on 07/07/2023).

[4] Ted K Ralphs et al. "On the capacitated vehicle routing problem". In: *Mathematical programming* 94 (2003), pp. 343–359.