Multiple Couriers Planning - MCP

Bombardi Davide davide.bombardi@studio.unibo.it
Castelli Giorgia giorgia.castelli2@studio.unibo.it
Fratini Alice alice.fratini2@studio.unibo.it
Mone Madalina Ionela madalina.mone@studio.unibo.it

June 2024

Contents

1 Introduction

This report summarizes the strategies and results related to the project we conducted as part of the CDMO course. The project consists of an attempt of optimization for a Multiple Couriers Planning (MCP) problem using four techniques: Constraint Programming (CP), Satisfiability Testing (SAT), Satifiability Module Theories (SMT) and Mixed-Integer Programming (MIP).

Our project required the group a total of about 150 days due to labour circumstances; it was carried out through a shared definition and research process for variables and functions to minimize (refer to the bibliography for an exhaustive list of the papers that we consulted [1], [2] and [3]). Next, each technique was developed: in this stage, group sessions were planned for reviewing and discussing strategies and formulations. At the end of the process, the experiments were conducted jointly and the project documentation phase was completed.

For the definition of the variables and the constraints of our models, we took inspiration from the model proposed in [2]. We summarize below the definition of the input variables common to all the techniques we used. Where not specified otherwise, we used the notations defined in the project description. Note that, for convenience, instead of the singular origin point o, we have assumed the existence of m origin points ($2 \cdot m$ in CP: m starting points and m arrival points) all spatially coincident. This allowed us to consider a biunivocal correspondence between each courier and his origin (two biunivocal correspondences for CP, one with the starting points and the other with the arrival points).

In each technique we defined two sets of variables: *pred* is the set of variables that associate each item i to its predecessor (i.e. the distribution point reached immediately before i); *cour*, instead, is the set of variables that associate each courier to the items that he carries (or viceversa, each item with the courier who carries it). A third set of variables *avoid_loops* has been defined for SAT, SMT and MIP. This set of variables aims to avoid internal loops by establishing a strict order relation between the items: given two items i and j carried by the same courier c, if i is before j in the path traveled by c then $avoid\_loops[i]$ has less True variables / is a smaller integer than $avoid\_loops[j]$. The exact definition of these sets of variables differs in each technique.

If *distance(c)* is the distance that the courier c travels, the objective function is defined as follows:

$$f = \max_{\forall \text{ couriers } c} distance(c)$$

The implementation of this objective function, that will be described in detail in the corresponding sections, is quite straight forward in CP and MIP, while in SAT and SMT is obtained adding a new constraint for every new solution found.

One of the main difficulties faced by our group is the definition of an upper bound for the function to be minimized. We weren't able to find many articles and documents about MCP problems that use an objective function similar to

ours, since in the literature the objective function is usually the sum of the travelled distances, rather than the maximum. Our reflections on the subject didn't lead us to the definition of a fully satisfactory upper bound. We observed that due to the problem assumptions (in particular the triangular inequality) the worst possible scenario is that all the items have to be carried by one single courier. Therefore our upper bound for the objective function is the distance travelled following a random path that goes through all the distribution points. This upper bound could have been tightened by finding the shorter path that goes through all the distribution points. However, this becomes computationally expensive for large instances and its benefits didn't seem worth it, since usually $m > 1$ and every solution where at lest two couriers carry at least one item each gives a lower objective function's result. Other definitions of this upper bound require several assumptions (for example on the couriers' capacities) so we decided to exclude them.

A second difficulty encountered by our group concerns the approach to instances for which, within a maximum execution time of 300 seconds, no sub-optimal solutions have been found. Therefore, in an attempt to prioritize the quantity of the solutions found within the given time rather than the quality of the solutions found, we developed (for SAT and SMT) a pre-processing step to be done on the instances: this step reduces the number of items n by creating $k < n$ "clusters" of items and assuming that each cluster is carried by a single courier. The clusters are created pairing together the couple of items that are closest to each other, and iteratively repeat this until there are $\lfloor \frac{n}{m} \rfloor$ in at least one cluster. After this, a simpler search is performed inside each cluster, assuming $m = 1$ and thus excluding the *cour* variables and the constraints about couriers' capacities. In the end a new, smaller distance matrix is defined as well as a new items' sizes array, and a full search is performed on the new search space obtained considering each cluster as a single item.
This approach never finds the optimal solution, and therefore is not meant to be used on small instances, where the full search works fine, but rather on larger instances where the full search fails.

2 CP Model

2.1 Decision variables

As described in the Introduction, the main decision variables are represented by two arrays: *pred*, of length $2 \cdot m$ and domain $[1, n + 2 \cdot m]$, such that $pred_i$ is the predecessor of i, and *cour*, of length $2 \cdot m$ and domain $[1, m]$, such that $cour_i$ is the courier passing by i. The additional intermediate variable q, of length $2 \cdot m$ and domain $[0, max(l)])$, describes the volume carried by the corresponding courier upon arrival at location i -including item i.

2.2 Objective function

The model aims to minimize the objective function described in the Introduction, and formalized as follows:

$$distance = \max_{i \in [n+m+1, n+2m]} (dists_i)$$

where $dists_i$ is the distance traveled by the corresponding courier at the arrival at location i; the upper bound of *distances*, as well as that of $dists_i$, is $D_{n+1,1} + \sum_{i=1}^{n} D_{i,i+1}$, describing the length of a path that connects all packages, carried out by a single courier, as discussed in the Introduction. The lower bound of $dists_i$ is 0, while that of *distance* is $\max_{i \in 1,n}(D_{n+1,i} + D_{i,n+1})$, describing a path that goes through the single item location with the longest length from each starting and ending point: this definition follows from triangle inequality with an argument similar to the one used to define the upper bound.

2.3 Constraints

The first set of constraints defined for the model defines the *pred* and *cour* arrays: firstly, it is set by default that $pred_{n+i} = n + m + i$; an *alldifferent* constraint on *pred* also specifies that each predecessor must be different from the others. Furthermore, it is put that $cour_{n+m+i} = cour_{n+m+i} = i$. Finally, the link between the *pred* array and the *cour* array is set through a constraint stating that $cour_{pred_i} = cour_i$. It is furthermore specified that $q_i = 0$, i. e. each courier is empty at his starting point; the iterative definition of q is set through the constraint $q_i = q_{pred_i} + s_i$, meaning that the volume carried upon arrival at location i is equal to the sum of the volume carried when visiting the location prior to i and the volume of i; in addition, it is put that $q_i = q_{pred_i}$ for every ending point i; finally, the constraint $q_i < l_{cour_i}$ states that the volume carried by each courier must not be greater than its capacity.

Implied constraints Through an *at_least_one* constraint, it is specified that each courier must carry at least one item: this constraint is justified by the fact that, by triangular inequality, the transportation of more items by one courier will be less efficient than the carriage of the same items by two couriers; from this it follows that any solution that leaves the courier i inactive will be more inefficient than a solution that also involves courier i. It is to be noted, moreover, the attempt to introduce a constraint that explicitly prevents the occurrence of loops in the definition of the *pred* array, i.e. routes without starting or arrival points: this occurrence is already prevented by the combined action of the *alldifferent*(*pred*) and *at_least_one* constraints, but we attempted to to define explicitly this "*avoid loop*" constraint in the hope of increasing the computational efficiency of the project; performances, however, did not improve due to it. It is also to be noted the attempt, carried out with the same purpose as the previous experiment, to define a *succ* array -describing the successor relationship- similarly to the definition of *pred* array; this attempt, inspired by [2], did not result in an improvement in the performance of the model.

Symmetry breaking constraints A symmetry has been detected in the problem, concerning the scenario in which two couriers i and j have the same capacity: in this case solution 1, which associates the path x to i and the path y to j, is equivalent to solution 2, associating the path y to j and the path x to i. The symmetry was broken by exploiting the numerical ordering between the items and that between the couriers, stating that $(i < j \wedge l[i] = l[j] \wedge pred[k] = n + i \wedge pred[z] = n + i) \implies k < z$.

2.4 Validation

Experiments were conducted exploiting different combinations of solvers and search strategies. All configurations were tested both on a version of the code without the symmetry breaking constraint, and on the version presented in the project. We decided not to report the results of the first class of experiments as they invariably showed worse performances than the versions with symmetry breaking; below is a comparison of the five most promising configurations of solvers and search strategies for model resolution with symmetry breaking.

Experimental design Gecode and Chuffed solvers were used.

Gecode has been used in combination with three different search strategies:

- *"IndomainRandom"* [**G-IR**] - int_search(*pred*, dom_w_deg, indomain_random), int_ search(*cour*, dom_w_deg, indomain_random), restart_luby(250);
- *"IndomainRandom_ RelAndRec"* [**G-IRR**] int_search(*pred*, dom_w_deg, indomain_ random), int_search(*cour*, dom_w_deg, indomain_random), restart_luby(250), relax_and_ reconstruct(*pred*, 85);
- *"IndomainMin_ RelAndRec"* [**G-IMR**] int_search(*pred*, dom_w_deg, indomain_min), int_search(*cour*, dom_w_deg, indomain_min), restart_luby (250), relax_and_ reconstruct(*pred*, 85).

Chuffed has been used in combination with two different search strategies:

- *"Smallest"* [**C-S**] - int_search(*pred*, input_order, indomain_min), int_search(*cour*, smallest, indomain_min), restart_luby(250);
- *"InputOrder"* [**C-I**] - int_search(*pred*, input_order, indomain_min), int_search (*cour*, smallest, indomain_min), restart_luby(250).

Experimental results While Chuffed seems to perform good on small instances, Gecode-based search strategies seem to perform best in searching for a feasible solution on instances having bigger size. In particular, the combination of *indomain_random* and *relax_and_reconstruct* seem to be the best way to get a feasible solution quickly; on some instances, however, *indomain_min* could permorm better because of the particular input order of the values.

[inst]	G-IR	G-IRR	G-IMR	C-S	C-I
1	**14**	14	14	**14**	**14**
2	226	226	226	226	226
3	**12**	18	**12**	**12**	
4	220	220	N/A	N/A	N/A
5	**206**	206	206	**206**	**206**
6	322	322	322	N/A	**322**
7	172	167	167	N/A	541
8	186	186	N/A	N/A	N/A
9	436	436	N/A	N/A	N/A
10	244	244	N/A	N/A	N/A
11	809	N/A	N/A	N/A	N/A
12	482	468	N/A	N/A	N/A
13	1364	610	566	2156	N/A
14	N/A	N/A	N/A	N/A	N/A
15	N/A	N/A	1066	N/A	N/A
16	352	286	N/A	N/A	N/A
17	N/A	N/A	1675	N/A	N/A
18	1230	1221	N/A	N/A	N/A
19	447	334	N/A	N/A	N/A
20	N/A	N/A	1587	N/A	N/A
21	866	1138	N/A	N/A	N/A

Table 1: Objective Function's Values for CP Models.

3 SAT Model

3.1 Decision variables

In the main SAT model, *pred* and *cour* are two matrices of Boolean variables of shapes $(n+m) \times (n+m)$ and $m \times (n+m)$, respectively. In particular $pred_{i,j}$ is true iff item j is the predecessor of item i and $cour_{c,i}$ is true iff courier c carries item i. Lastly, *avoid_loops* has been defined as a $n \times \lfloor \frac{n}{m-1} + 1 \rfloor$ matrix of Boolean variables. As explained in the introduction, the variables listed in *avoid_loops* don't have an explicit semantic.
In the small SAT solver *cour* is not present, *pred* has the same shape of main model's *pred* but with $m = 1$, and *avoid_loops* is a $n \times n$ matrix.

3.2 Objective function

The optimization in SAT (for both the main model and the smaller solver) is done by iteratively checking the satisfaiability of the problem. At the beginning of each iteration a constraint is added to the model in order to force the solver to find a better solution. In particular, if *best_obj* is the value of the objective

function (computed following the definition given in the introduction), for each courier c we define the following set of boolean expressions:

$$tmp\_dist = \bigcup_{\substack{i=1\dots n \\ j=1\dots n}} \left[\bigsqcup_{k=1\dots D_{i,j}} \{cour_{c,i} \wedge pred_{j,i}\} \right] \cup$$
$$\bigcup_{i=1\dots n} \left[\bigsqcup_{k=1\dots D_{n,i}} \{pred_{i,n+c}\} \right] \cup$$
$$\bigcup_{i=1\dots n} \left[\bigsqcup_{k=1\dots D_{i,n}} \{pred_{n+c,i}\} \right]$$

where $\sqcup$ is the disjoint union, and then we add an *at most* k constraint on *tmp_dist* with $k = best\_obj - 1$. The disjoint union ensures that the number of True expressions in *tmp_dist* is equal to the distance traveled by c, so if we impose that all couriers' paths must be shorter than *best_obj* we obtain that the courier that travel the most travels still less than *best_obj* and therefore the objective function on a new found solution will give a lower result.
Note that this constraint is removed from the solver at each iteration in order to re-add it with a lower *best_obj*.

3.3 Constraints

The first set of constraints added to the solver concerns the assignment of the items to the couriers. Since each item has to be carried by one courier, we add an *exactly one* constraint on the first n columns of *cour*. For the remaining m columns it was sufficient to add $cour_{i,n+i}$ and $\neg cour_{i,n+j}$ for $i = 1 \dots n$ and $j = 1 \dots m$, since their role is to simplify the definition of the other constraints. The weight-capacity constraint with the same method used for the implementation of the objective function. In particular, for every courier c, we define:

$$cour\_weight = \bigcup_{i=1\dots n} \left[\bigsqcup_{k=1\dots s_i} \{cour_{c,i}\} \right]$$

and we add an *at most* k constraint on *cour_weight* with $k = l_c$.
For what concerns *pred*, since each item/origin point has exactly one predecessor and each item/origin point is predecessor of exactly one item, we simply add an *exactly one* constraint to each column and to each row of *pred*.
A further constraint is needed to maintain consistency between *cour* and *pred*. In particular, if item j is predecessor of item i and i is carried by courier c, then c carries j as well. This was added using a simply implication constraint for every couple of items and for every courier. Lastly we defined two sets of constraints regarding *avoid_loops*. The first set consists in the following implications: for

each row i of *avoid_loops* and for each column k

$$\neg\ avoid\_loops_{i,k} \Rightarrow \neg \left(\bigvee_{j=1\ldots k} avoid\_loops_{i,j} \right)$$

Thanks to this set of implications, in each column j of *avoid_loops*, if $avoid\_loops_{i,j}$ is true then all the variables in the column j at rows with higher indexes are true. In other words, the columns of *avoid_loops* are of the type (*False, ..., False, True, ..., True*). The second set of constraints that concerns *avoid_loops* are: if *col* is the number of columns of *avoid_loops*, for every couple of items i and j

$$pred_{i,j} \Rightarrow \bigvee_{k=1\ldots col} (avoid\_loops_{i,k} \wedge \neg\ avoid\_loops_{j,k})$$

In other words, if j is the predecessor of i then the i-th row of *avoid_loops* has more True variables than the i-th row.
The small SAT model has the same constraints except the ones concerning *cour*, that aren't needed.

3.4 Validation

The full search performs quite well on small instances, due to its capability of reaching the optimal solution, while on larger instances fails to find any solution. The clustering approach can't prove optimality because of his structure but it finds good quality solutions way quicker than the full search. The objective function's values can be observed in table 2. Note that if the value is **bold** the model has proved its optimality.

4 SMT Model

4.1 Decision variables

In the main SMT model, *pred* and *cour* are two arrays of Int variables, both of shapes $(n + m)$. $pred_i$ is equal to j iff item j is the predecessor of item i and $cour_i$ is equal to c iff courier c carries item i. Lastly, *avoid_loops* has been defined as a n-long array of BitVec variables (with 16 bits each). As explained in the introduction, the variables listed in *avoid_loops* don't have an explicit semantic.
In the small SMT solver *cour* is not present, *pred* has the same shape of main model's *pred* but with $m = 1$, and *avoid_loops* is still long n.

4.2 Objective function

The way we implemented the objective function is similar to what we did for SAT. The main difference is that we haven't used the disjoint union and the *at*

inst	FullSearch	Clustering
1	**14**	N/A
2	**226**	314
3	**12**	N/A
4	**220**	313
5	**206**	N/A
6	**322**	362
7	**167**	185
8	**186**	192
9	**436**	499
10	**244**	268
11	N/A	531
12	N/A	N/A
13	1626	696
14	N/A	N/A
15	N/A	N/A
16	507	356
17	N/A	N/A
18	N/A	N/A
19	N/A	N/A
20	N/A	N/A
21	N/A	N/A

Table 2: Objective Function's values for SAT Models.

most k constraint, instead we've added a pseudo-boolean inequality in order to set an upper bound to the distance traveled by each courier.

4.3 Constraints

In SMT we've set the bounds of the Int variables with simple inequalities. The variables in *cour* can assume values in $[0, m[$, the variables in *pred* can assume values in $[0, m+n[$, and the variables in *avoid_loops* can assume values in $[0, n[$. The weight-capacity constraint is implemented as follows: for each courier c

$$\sum_{i=1}^{n} \delta_{i,c} \cdot s_i \leq l_c \qquad \text{where } \delta_{i,c} = 1 \text{ if } cour_i = c, \text{ 0 otherwise}$$

Since all the items must have a different predecessor, we've added the $Different$ constraint on *pred*, while for maintaining the consistency between *pred* and *cour* we added a constraint similar to the one added in SAT for the same purpose. Lastly, a set of constraint concerning the *avoid_loops* array has been added. The reasoning behind it is the same we made for the *avoid_loops* matrix in SAT: for every couple of items i and j

$$(pred_i = j) \Rightarrow (avoid\_loops_i \ > \ avoid\_loops_j)$$

4.4 Validation

SMT Models behave similarly to SAT Models, but in SMT the advantages of the clustering methods are clearer: while the full search fails to find any solution as soon as the istances gets bigger, the clustering approach, in some cases, menage to reach a sub-optimal solution. The objective function's values can be observed in table 3. Note that if the value is **bold** the model has proved its optimality.

inst	FullSearch	Clustering
1	**14**	N/A
2	**226**	314
3	**12**	N/A
4	**220**	313
5	**206**	N/A
6	**322**	356
7	183	185
8	**186**	186
9	**436**	436
10	**244**	244
11	N/A	N/A
12	N/A	N/A
13	N/A	704
14	N/A	N/A
15	N/A	N/A
16	N/A	389
17	N/A	N/A
18	N/A	N/A
19	N/A	N/A
20	N/A	N/A
21	N/A	N/A

Table 3: Objective Function's values for SMT Models.

5 MIP Model

This section describes two Mixed-Integer Programming (MIP) models we developed. The first model is an attempt to maintain the *pred* and *cour* arrays/matrices, following the CP, SAT and SMT structure. We implemented this model using the Gurobi library in Python. In order to formulate the constraints in a linear manner we defined a 3D matrix that implies an high number of variables and thus, in the second model, we've decided to directly use a 3D matrix in order to reduce the number of variables. For this model we used the CBC solver.

5.1 Decision variables

First Model. In the fisrt MIP model we've defined *pred* and *cour* as Gurobi Binary matrix of shape $(n+m, n+m)$ and $(m, n+m)$ respectively, such that $pred_{i,j}$ is true iff the j-th item is the predecessor of the i-th item and $cour_{i,j}$ is 1 iff the i-th courier takes the j-th item. Then we defined *avoid_loops* as a Gurobi integer vector of shape n, serving the purpose to avoid the occurrence of paths having no starting point: this vector has a lower bound $lb = 1$ and an upper bound $ub = n$. In order to compute the distances traveled by the couriers, we had to define the 3D Gurobi Binary matrix *prod* of shape $m \times (n+m) \times (n+m)$, that represents the product of *cour* and *pred*. In particular $prod_{c,i,j}$ is 1 iff $cour_{c,i} \wedge pred_{i,j}$ is 1. Lastly, *dist* is a Gurobi integer vector of shape m, which stores the maximum distance for each courier.

Second Model. In the second MIP model we've defined *cour_pred* as binary matrix of shape $m \times (n+1) \times (n+1)$, such that $cour\_pred_{c,i,j}$ is 1 if the courier c carries the item i and the item j is the predecessor of i. *avoid_loops* and *dist* have the same definitions and shapes given in the first model.

5.2 Objective function

In both our MIP models, the objective function is expressed through a inter variable *max_dist* that represent the maximum distance traveled by any courier. This is achieved differently in the two models: in the first one we added the Gurobi constraint $addGenConstrMax$ that makes *max_dist* equal to the maximum of *dist*, while in the second model we added the constraint $max\_dist \geq dist_c$ for all the couriers c.

5.3 Constraints

First Model. We ensured that each item j is carried by exactly one courier with the constraint $\sum_{i=0}^{m-1} cour_{i,j} = 1$ and we set the origin of each courier c with the constraint $cour_{c,n+c} = 1$ and $cour_{c,n+i}$ for every $i = 1 \dots m$, $i \neq c$. To ensure that the weight of the packages carried by each courier does not exceed the courier's capacity, for every courier c we added the constraint $\sum_{i=0}^{n-1} cour_{c,i} \cdot s_i \leq l_c$.
For what concerns the *pred* matrix, to ensure that each item/origin point has exactly one predecessor and that each item/origin point is predecessor of exactly one other item, we imposed that the sum over the elements of any column and any row is equal to one. Then we made sure that there is consistency between *pred* and *cour* by adding the constraint $cour_{c,j} \geq cour_{c,i} + pred_{i,j} - 1$. In fact, if c carries i and j is predecessor of i then $cour_{c,j} \geq 1 + 1 - 1 = 1$, while if c doesn't carry i or j is not predecessor of i, the constraint doesn't influence the domain of $cour_{c,j}$. To avoid internal loops, we defined the constraint:

$$avoid\_loops_i - avoid\_loops_j \geq n \cdot (pred_{i,j}) - n + 1 \quad \forall i, j = 1 \dots n$$

that, thanks to the bounds of the variables of *avoid_loops*, defines a strict order relation between items carried by the same courier.
The behaviour of *prod* is defined by 3 sets of constraints: for every items i and j and for every courier c

- $prod_{c,i,j} \leq cour_{c,i}$. This ensures that $prod_{c,i,j}$ can be 1 only if $cour_{c,i}$ is 1. If $cour_{c,i}$ is 0, then $cour_{c,i,j}$ must be 0;
- $prod_{c,i,j} \leq pred_{i,j}$. This ensures that $prod_{c,i,j}$ can be 1 only if $pred_{i,j}$ is 1.
- $prod_{c,i,j} \geq cour_{c,i} + pred_{i,j} - 1$. This ensures that if both $cour_{c,i}$ and $pred_{i,j}$ are 1 then $prod_{c,i,j}$ must be 1, in fact if both these variables are 1, then $cour_{c,i} + pred_{i,j} = 2$ and then $prod_{c,i,j}$ must be at least 1;

Thanks to the *prod* matrix we can define the constraints that set $dist_c$ equal to the distance traveled by the courier c:

$$dist_c = \sum_{i=1}^{n} (pred_{i,n+c} \cdot D_{n,i}) + \sum_{i=1}^{n} (pred_{n+c,i} \cdot D_{i,n}) + \sum_{i=1}^{n} \sum_{j=1}^{n} (prod_{c,i,j} \cdot D_{j,i})$$

Second Model. To impose that each item has exactly one predecessor and that each item is predecessor of exactly one other item, it is sufficient to state that

$$\sum_{c=1}^{m} \sum_{j=1}^{n+1} (cour\_pred_{c,i,j}) = 1 \qquad \forall i = 1 \dots n$$

$$\sum_{c=1}^{m} \sum_{i=1}^{n+1} (cour\_pred_{c,i,j}) = 1 \qquad \forall j = 1 \dots n$$

These constraints also ensures that each item is carried exactly one courier. In addition to these, we must state that for every courier c, $\sum_{i=1}^{n+1} (cour\_pred_{c,i,n}) = 1$ and $\sum_{j=1}^{n+1} (cour\_pred_{c,n,j}) = 1$ in order to ensure that each courier starts and ends at the origin.
To establish the consistency of courier-predecessors on each path, we added the following constraint: for every couple of items i and j and for every courier c, $cour\_pred_{c,i,j} \leq \sum_{h=1}^{n+1} (cour\_pred_{c,h,j}) = 1$.
For what concerns the weight-capacity constraints, we stated that for every courier c

$$\sum_{i=1}^{n} \sum_{j=1}^{n+1} (cour\_pred_{c,i,j} * s_i) \leq l_c$$

The array *avoid_loops* has the same constraints defined in the first model but $pred_{i,j}$ is substituted by $\sum_{c=1}^{m} (cour\_pred_{c,i,j})$.
Lastly, for the *dist* array, for every courier c we added

$$dist_c = \sum_{i=1}^{n+1} \sum_{j=1}^{n+1} (cour\_pred_{c,i,j} \cdot D_{j,i}) \,.$$

5.4 Validation

The first model was solved using Gurobi with three different focuses of the solution strategy. The first one [**G-D**] is balanced between finding new feasible solutions and proving that the current solution is optimal, the second one [**G-F**] prioritize finding feasible solution quickly and the third one [**G-O**] is more interested in proving optimality.
The Second Model uses the CBC Solver and has been tried with the three different focuses [**CBC-D**], [**CBC-F**], [**CBC-O**] as well.
The objective function's values can be observed in table 4. Note that if the value is **bold** the model has proved its optimality. The models perform great on small instances and menage to prove optimality in the majority of them, while on bigger instances they fail to find any solution at all. This is probably beacuse of the high number of variables an constraints of both models. For what concerns the time used by the models to conclude the search, the Gurobi-based model outperforms the CBC one.

inst	**G-D**	**G-F**	**G-O**	**CBC-D**	**CBC-F**	**CBC-O**
1	**14**	**14**	**14**	**14**	**14**	**14**
2	**226**	**226**	**226**	226	226	226
3	**12**	**12**	**12**	**12**	**12**	**12**
4	220	**220**	**220**	220	220	220
5	**206**	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	322	322	322
7	167	167	167	190	190	190
8	**186**	**186**	**186**	186	186	186
9	436	436	436	436	436	436
10	244	244	244	244	244	244
11	N/A	N/A	N/A	N/A	N/A	N/A
12	N/A	N/A	N/A	N/A	N/A	N/A
13	498	422	N/A	N/A	N/A	N/A
14	N/A	N/A	N/A	N/A	N/A	N/A
15	N/A	N/A	N/A	N/A	N/A	N/A
16	N/A	N/A	N/A	N/A	N/A	N/A
17	N/A	N/A	N/A	N/A	N/A	N/A
18	N/A	N/A	N/A	N/A	N/A	N/A
19	N/A	N/A	N/A	N/A	N/A	N/A
20	N/A	N/A	N/A	N/A	N/A	N/A
21	N/A	N/A	N/A	N/A	N/A	N/A

Table 4: Objective Function's Values for MIP models.

6 Conclusion

From the implementation of the four approaches, it was clear how difficult it can be to deal with NP-hard problems, since the performances of every model we tried get significantly worse as soon as the size of the instances got bigger, often failing in giving any result within the fixed time limit. It was clear, however, how the implementation choices and the different search strategies could significantly impact such performances. With this in mind, trying other search strategies with different implementation choices could lead to further improvements on the optimization of the model.

References

[1] Thibaut Feydy et al. *Priority Search with MiniZinc*. 2017.

[2] Philip Kilby and Paul Shaw. "Chapter 23 - Vehicle Routing". In: *Handbook of Constraint Programming*. Ed. by Francesca Rossi, Peter van Beek, and Toby Walsh. Elsevier, 2006, pp. 799–834.

[3] Konstantin Sidorov and Alexander Morozov. "A review of approaches to modeling applied vehicle rooting problems". In: arXiv.org, 2021.