Multiple couriers problem

Abdul Moqeet - abdul.moqeet@studio.unibo.it
Fabio Giordana - fabio.giordana@studio.unibo.it

December 2024

1 Introduction

This report describes how we tackled the multiple couriers problem using all the four paradigms described during the course. It contains the definition of models and solving techniques used together with the best results obtained for each approach. At the beginning, we divided the work as follows: CP and SAT to Fabio Giordana and SMT and MIP to Abdul Moqeet; after gathering the initial ideas, both of us worked on everything depending on who was having promising ideas on how to refine the models and improve the results.

1.1 Implied constraint and objective bounds

Since we know that the triangular inequality $D_{i,j} \leq D_{i,k} + D_{k,j} \quad \forall i, j, k = 1, \ldots, n+1$ holds and that $n \geq m$, we can assume as an implied constraint that each courier should carry at least one item. In fact, for every path, we can state that:

$$D_{n+1,j} + D_{j,n+1} \leq D_{n+1,k} + D_{k,j} + D_{j,n+1} \quad \forall j, k = 1, \ldots, n$$

The objective function common to all models is to minimize the maximum distance traveled by the couriers. The actual implementation of the objective changes for the different approaches, so it will be discussed in the next sections.

As a lower bound for the objective, we consider the maximum distance traveled carrying only one item:

$$lowerbound = \max_{j=1,\ldots,n}(D_{n+1,j} + D_{j,n+1})$$

The upper bound depends on the presence of the implied constraint. To define it, we first introduce an array containing the maximum leaving distance for each node j, also considering the depot: $maxvalues = [\max_{k=1,\ldots,n+1}(D_{j,k}) \forall j = 1, \ldots, n+1]$ and then we sort it in ascending order: $sortedvalues = sort(maxvalues)$. Without the implied constraint, we simply sum all the values in the array:

$$upperbound = \sum_{j=1}^{n+1} sortedvalues_j$$

With the implied constraint, we consider that m-1 couriers deliver the items with the lower leaving distances, so we impose the upper bound:

$$upperbound = \sum_{j=m}^{n+1} sortedvalues_j$$

The lower bound has proven itself as vastly more useful in improving the performances than the upper bound.

1.2 Symmetry breaking constraints

For the symmetry breaking constraints, we consider two different types of symmetries: symmetry on the couriers load capacities and symmetry on the distances traveled. For the first one, since the only unique trait of the couriers is their maximum load capacity, we impose on ordering in the actual load carried by each of them:

$$l_i < l_j \Rightarrow size_i \leq size_j \; \forall i = 1, \ldots, m, \; j = i+1, \ldots, m$$

where $size_i$ is the actual load carried by courier i. Since two couriers with the same maximum capacity can always swap their paths, we also impose an ordering constraint for the paths in this specific situation:

$$l_i == l_j \Rightarrow path_{i,n+1} \leq path_{j,n+1} \; \forall i = 1, \ldots, m, \; j = i+1, \ldots, m$$

where $path_{i,n+1}$ is the first item reached from the depot in the path of courier i.

For the second type of symmetry, we exploit the fact that if the distances matrix is symmetric, then a path and its reverse will always have the same cost, since $D_{j,k} == D_{k,j} \; \forall j, k = 1, \ldots, n+1$ so we impose that:

$$symmetric \Rightarrow path_{i,n+1} \leq path_{i,last} \; \forall i = 1, \ldots, m,$$

where $path_{i,n+1}$ is the first item reached from the depot in the path of courier i and $path_{i,last}$ is the last one before returning to the depot.

2 CP model

We implement our solution for the MCP in CP using Minizinc.

2.1 Decision Variables

The variables for our cp model are:

- Integer matrix path $\in N^{m \times n+1}$ where $path[i,j] == k$ with $k! = j$ if and only if courier i goes from item j to k, while $path[i,j] == j$ if and only if courier i does not carry item j. The domain for each variable in the matrix is $\{1, \ldots, n+1\}$
- Integer array cost $\in N^m$ where cost[i] represent the actual distance traveled by courier i.

2.2 Objective function

The objective function is to minimize the maximum cost variable: $\max_{i=1,\dots,m}(cost[i])$. We applied the bounds described in section 1.1.

2.3 Constraints

2.3.1 Constraints on cost

We impose that the cost variables actually represent the distance traveled by couriers, so:

$$cost[i] = \sum_{j=1..n+1, path[i,j]!=j} D[j, path[i,j]] \; \forall i = 1, \dots, m$$

2.3.2 Subcircuit constraint

The subcircuit constraint is a minizinc global constraint that impose that path[i, 1..n+1] describes an Hamiltonian sub-cycle, with the characteristics given in section 2.1:

$$subcircuit(path[i, 1..n+1] \; \forall i = 1, \dots, m$$

2.3.3 Constraints on loads

The constraints on loads enforces the fact that each courier does not exceed his maximum load capacity:

$$\sum_{j=1..n, path[i,j]!=j} s[j] \leq l[i] \; \forall i = 1, \dots, m$$

2.3.4 Constraint on the starting position

This constraint enforces that for each courier carrying at least one item, the path must start from the depot:

$$|j \in 1..n \; |path[i,j]! = j| > 0 \Rightarrow path[i, n+1]! = n+1 \; \forall i = 1, \dots, m$$

2.3.5 Exactly one constraint

The constraint forces each item to be carried exactly once:

$$|i \in 1..m | path[i,j]! = j| == 1 \; \forall j = 1, \dots, n$$

2.3.6 Implied constraint

With the implied constraint we force each courier to have at least one item in his path:

$$path[i, n+1]! = n+1 \; \forall i = 1, \dots, m$$

By using this constraint, we can avoid expressing constraint of section 2.3.4, since we are already defining the correct starting position

2.3.7 Symmetry breaking constraints

For the symmetry breaking constraints, we impose all the constraint discussed in section 1.2:

- $l[i] < l[k] \Rightarrow \sum_{j=1..n, path[i,j]!=j} s[j] \leq \sum_{j=1..n, path[k,j]!=j} s[j] \ \forall i = 1, \ldots, m, \ k = i+1, \ldots, m$
- $l[i] == l[j] \Rightarrow path[i, n+1] \leq path[j, n+1] \ \forall i = 1, \ldots, m, \ j = i+1, \ldots, m$
- $symmetric \Rightarrow path[i, n+1] \leq argmax(path[i, 1..n+1]) \ \forall i = 1, \ldots, m,$

2.4 Validation

2.4.1 Experimental design

We run all the experiments on an ASUS Zenbook UX431FN, Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz, 8,00 GB RAM with a time limit of 300 seconds. We experimented using two different CP solvers: Gecode and Chuffed. For Gecode, the best performances were reached with `dom_w_deg` variable selection and `in_domain_random` value order applied on the path variables, especially when also applying a Large Neighbourhood Search (LNS) coupled with `restart_luby(100)`, while a restart strategy alone had mixed effects on the performances. For Chuffed, where the mentioned search annotations and LNS are not available, we used `random_order` and `indomain_min` coupled with just `restart_luby(100)`.

2.4.2 Experimental results

We show the results only for the instances that were not solved to optimality by all the configurations of models and solvers within the given time limit.

Instance	Gc	Gc LNS	Gc LNS Impl	Gc LNS Impl Sym S	Gc LNS Impl Sym D	Gc LNS Impl Sym	Ch	Ch LUBY	Ch Impl	Ch Impl LUBY	Ch Impl Sym D	Ch Impl Sym D LUBY
7	303	**167**	**167**	174	**167**	174	**167**	**167**	**167**	**167**	**167**	**167**
8	297	**186**	**186**	**186**	**186**	**186**	**186**	**186**	**186**	**186**	**186**	**186**
11	1376	490	402	N/A	465	N/A	N/A	932	955	901	1006	N/A
12	691	**346**	350	N/A	**346**	N/A	N/A	570	382	538	537	N/A
13	888	536	568	564	498	534	N/A	924	1806	926	1806	986
14	1136	681	698	N/A	689	N/A	N/A	N/A	1549	N/A	1656	N/A

15	968	725	723	N/A	731	N/A	N/A	N/A	1038	N/A	1046	N/A
16	523	**286**	**286**	N/A	**286**	N/A	N/A	297	**286**	359	**286**	N/A
17	1934	1133	1131	1209	1161	1212	N/A	N/A	1310	N/A	1346	N/A
18	1234	587	554	N/A	600	N/A	N/A	1185	1291	1258	1374	N/A
19	562	**334**	**334**	N/A	**334**	N/A	N/A	491	**334**	511	**334**	N/A
20	N/A	1046	1111	1101	1060	1163	N/A	N/A	1664	N/A	1664	N/A
21	N/A	510	469	N/A	569	N/A	N/A	1027	1207	994	1256	N/A

Considering Gecode, LNS search strategy is the aspect that is more clearly improving the performances of the solver, while the symmetry breaking constraints on the couriers capacities are making the performances worse, probably because they are reducing the search space too much, not allowing the solver to find even a sub optimal solution within the time limit. The symmetry breaking constraint on the distances matrix is not consistently improving the performances for Gecode, while it is making them slightly worse for Chuffed. The implied constraint has much more effect on improving the performances for the Chuffed solver rather than for Gecode. The luby restart strategy alone in Chuffed is not consistently improving the performances.

3 SAT model

We implement our solution for the MCP in SAT using the Z3 Solver API in Python. All the mathematical operations between numbers reported in the next sections have been implemented by representing the integer numbers in binary notation using boolean variables and then by computing the specific operation by means of boolean constraints on those variables. Since one of the challenges in solving SAT problems is the scalability, we try to reduce as much as possible the number of variables and constraints used, in particular when dealing with binary representations of integer numbers.

3.1 Decision variables

The variables of the SAT model are as follows (the notation used considers that python arrays and matrices are indexed starting from 0):

- Boolean tensor path $\in \{0,1\}^{m\times(n+1)\times(n+1)}$ where path[i][j][k] is True if and only if courier i+1 goes from node j+1 to node k+1.
- Boolean matrix a $\in \{0,1\}^{m\times n}$ where a[i][j] is True if and only if item j+1 is assigned to courier i+1.

- Boolean matrix order $\in \{0,1\}^{n \times bits}$ where array order[j] encodes the binary representation of the position of the item j+1 in its courier path; bits represent the minimum number of bits necessary to write the position of the item.
- Boolean matrix loads $\in \{0,1\}^{m \times bits}$ where array loads[i] represents the actual load carried by courier i+1; for each courier, the number of bits used to write the load is the same as the number of bits necessary to write his maximum capacity.
- Boolean matrix dist $\in \{0,1\}^{m \times bits}$ where array dist[i] represents the actual distance traveled by courier i+1; the number of bits used is the same as the number of bits used for the upper bound.

3.2 Objective function

The objective function is to minimize dist[i] $\forall i = 0, \ldots, m-1$. The imposed bounds are the ones discussed in section 1.1.

3.3 Constraints

3.3.1 Exactly one constraint

The constraints forces each item to be carried exactly once. For the implementation of the exactly one constraint we choose the sequential encoding.

$$\forall j = 0, \ldots, n-1 \; \exists! i = 0, \ldots, m-1 \; s.t. a[i][j]$$

3.3.2 Self loop constraint

This constraint prevents any courier from moving from one node to the node itself:

$$\forall i = 0, \ldots, m-1 \; \forall j = 0, \ldots, n \; \neg path[i][j][j]$$

3.3.3 Starting and finishing in the depot

This constraints impose that each path must start and end in the depot:

$$\forall i = 0, \ldots m-1 \bigvee_{j=0}^{n-1} a[i][j]$$
$$\Rightarrow (\exists! j = 0, \ldots, n-1 \; s.t. \; path[i][n][j] \bigwedge \exists! j = 0, \ldots, n-1 \; s.t. \; path[i][j][n])$$

3.3.4 Constraints between path and assignment variables

These constraints are used to ensure consistency between path and assignments variables, ensuring that each item assigned to courier i+1 is leaved and reached once:

- $\forall i = 0, \ldots, m-1 \forall j = 0, \ldots n-1, a[i][j]$

$$\Rightarrow (\exists! k = 0 \ldots n \; s.t. \; path[i][j][k] \bigwedge \exists! k = 0 \ldots n \; s.t. \; path[i][k][j])$$

$$\bullet\ \forall i = 0, \ldots, m-1 \forall j = 0, \ldots n-1, \neg a[i][j]$$
$$\Rightarrow (\forall k = 0 \ldots n\ \neg path[i][j][k] \bigwedge \forall k = 0 \ldots n\ \neg path[i][k][j])$$

3.3.5 Ordering constraint

With these constraints we impose a coherent ordering between items in the same path, to avoid sub-loops outside of the main path:

$$\bullet\ \forall i = 0, \ldots, m-1\ \forall j = 0, \ldots n-1, path[i][n][j]$$
$$\Rightarrow \forall k = 0, \ldots, bits\ \neg order[j][k]$$

Seeing order[j] as a binary number, we are imposing its value to 0, since j+1 is the first item in its path.

$$\bullet\ \forall i = 0, \ldots, m-1\ \forall j = 0, \ldots n-1,\ \forall k = 0, \ldots, n-1\ path[i][j][k]$$
$$\Rightarrow order[k] == order[j] + 1$$

3.3.6 Constraint on distances

We force the dist variables to actually represent the distances traveled by couriers:

$$\forall i = 0, \ldots, m-1\ dist[i] == \sum_{j=0..n, k=0..n, path[i][j][k]} D[j][k]$$

3.3.7 Constraints on loads

We force the loads variables to actually represent the loads carried by couriers and to not exceed the couriers maximum capacities:

$$\bullet\ \forall i = 0, \ldots, m-1\ loads[i] == \sum_{j=0..n-1, a[i][j]} s[j]$$

$$\bullet\ \forall i = 0, \ldots, m-1\ loads[i] <= l[i]$$

3.3.8 Implied constraint

To force each courier to carry at least one item, we impose:

$$\forall i = 0, \ldots, m-1\ \bigvee_{j=0}^{n-1} a[i][j]$$

3.3.9 Symmetry breaking constraints

We impose all the symmetry breaking constraints discussed in section 1.2:

- $l[i] < l[k] \Rightarrow loads[i] \leq loads[k]\ \forall i = 0, \ldots, m-1,\ k = i+1, \ldots, m-1$
  This specific constraint was imposed to the first solver in the split-first order-second approach discussed in the next section.

- $l[i] == l[j] \Rightarrow path[j][n] \leq path[i][n]\ \forall i = 0, \ldots, m-1,\ j = i+1, \ldots, m-1$
  For this constraint, we impose an ordering between paths by considering the couple of n+1 boolean variables encoding which items are reached from the depot by the two couriers as two integer numbers and by imposing a

mathematical ordering between them. This specific constraint was not used in the split-first order-second approach discussed in the next section, since expressing it in terms of assignments variables for the first solver introduced too much overhead, worsening the overall performances.

- $symmetric \Rightarrow ((path[i][j][n] \wedge path[i][k][j]) \Rightarrow path[i][k] \leq path[i][n])\ \forall i = 1, \ldots, m,\ \forall j = 0, \ldots, n-1\ \forall k = 0, \ldots, n-1$
  In this constraint we are imposing the ordering in the same way of the previous constraint. This specific constraint was imposed to the second solver in the split-first order-second approach discussed in the next section.

3.4 Validation

3.4.1 Experimental design

We run all the experiments on an ASUS Zenbook UX431FN, Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz, 8,00 GB RAM with a time limit of 300 seconds. We used two different search strategies in Z3py. In the first one, we use one solver combined with a binary search strategy. By using this strategy we were able to reduce the objective value at each iteration more rapidly than with a linear search, allowing us to effectively reduce the number of bits used to compute the dist variables at each iteration without introducing too much overhead. For the second search strategy, we first used a solver to find a suitable assignment between couriers and items and then we inject it into a second solver to find the best order between items. We found this split-first order-second strategy more suitable to the structure of our problem (especially for the time limit of 300 seconds) rather than a order-first split-second, because every ordering after a suitable assignment generates at least a suboptimal solution.

3.4.2 Experimental results

For instances from 1 to 10, we only show results for instances not always solved to optimality; for instances above 10, we show results only for instances for which at least one feasible solution was found.

Instance	Base	Implied	Implied Sym on S	Implied Sym on D	Implied Sym	Symmetry	2 Sol Base	2 Sol Implied	2 Sol Implied Sym on S	2 Sol Implied Sym on D	2 Sol Implied Sym	2 Sol Symmetry
7	201	224	181	224	181	183	221	202	187	202	187	176
13	N/A	1422	1446	1598	1998	N/A	1062	1184	1184	1166	1166	1070

The two solver approach seems to work slightly better, especially for instance 13 (the biggest instance solved), but since the evaluation of the different combinations of models and search strategies is based on just two instances which are the only ones not always solved to optimality, it is difficult to make reliable assertions about the performances. It is also worth to underline the fact that most of the instances from 11 to 21 exceeds the time limit just for the encoding of model and constraints.

4 SMT model

We implemented our solution for the MCP in SMT using the Z3 Solver API in Python, including Propositional Logic Theory for the boolean variables and LIA Theory for integers. We experimented the same two search strategies used in SAT, but with slightly different models: in the single solver case, we do not use assignment variables. Furthermore, additional variables are also used to express symmetry breaking constraints on the couriers capacities. In the next sections we will indicate when specific constraints and variables are used only in specific contexts.

4.1 Decision variables

As in SAT, the notation for matrices and arrays considers that those structures are indexed starting from 0:

- Common variables:
  - Integer matrix path $\in N^{m\times(n+1)}$ where path[i][j]==k with k!=j if and only if courier i+1 goes from node j+1 to node k+1. The domain for each variable in the matrix is $\{0,\ldots,n\}$
  - Integer array order $\in N^n$ where order[j] == k if and only if item j+1 is in position k in its courier path. The domain for each variable in the array is $\{0,\ldots,n\}$
  - Integer array dist $\in N^m$ where dist[i] represents the actual distance traveled by courier i+1.
  - Integer variable obj $\in N$ such that $obj == max(dist[i])\forall i = 0,\ldots,m-1$
- Additional variables for the split-first order-second search strategy:
  - Boolean matrix assign $\in \{0,1\}^{m\times n}$ where assign[i][j] is True if and only if courier i+1 carries item j+1.
- Additional variables for symmetry breaking constraint on sizes:
  - Integer array sizes $\in N^m$ where sizes[i] represents the actual load carried by courier i+1.

4.2 Objective function

The objective function is to minimize the obj variable. The imposed bounds are the ones discussed in section 1.1.

4.3 Constraints

4.3.1 Exactly one constraint

We impose that every item is delivered exactly once:

- One solver formulation: $\sum_{i=0..m-1, path[i][j]!=j} 1 == 1 \; \forall j = 0, \ldots, n-1$
- Two solvers formulation: $\sum_{i=0..m-1, assign[i][j]} 1 == 1 \; \forall j = 0, \ldots, n-1$

4.3.2 All different constraint

We impose different values for all the variables representing a specific courier path, using Z3 Distinct function:

$Distinct(path[i]) \; \forall i = 0, \ldots, m-1$

4.3.3 Starting in the depot

All paths must start in the depot:

$\sum_{j=0..n-1, path[i][j]!=j} 1 >= 1 \Rightarrow path[i][n]! = n \; \forall i = 0, \ldots, m-1$

4.3.4 Ordering constraints

To avoid sub-loops, we impose a coherent ordering between items in the same path, first for the starting items:

$If \sum_{i=0..m-1, path[i][n]==j} 1 == 1 \; then \; order[j] == 0 \; else \; order[j] > 0 \; \forall j = 1, \ldots, n-1$ And then for items in intermediate positions:

$If path[i][j] == k \wedge k! = j \; then \; order[k] == order[j]+1 \; \forall i = 1, \ldots, m-1 \; \forall j = 1, \ldots, n-1 \; \forall k = 1, \ldots, n-1$

4.3.5 Constraint on distances

Each dist variable must represent the actual distance traveled by each courier:

$dist[i] == \sum_{j=0..n, k=0..n, j!=k, path[i][j]==k} D[j][k] \; \forall i = 0, \ldots, m-1$

4.3.6 Constraint on couriers capacities

Each courier can not exceed his maximum capacity:

- One solver formulation: $\sum_{j=0..n-1, path[i][j]!=j} s[j] <= l[i]\ \forall i = 0, \ldots, m-1$
- For the two solvers formulation we exploit the PbLe Z3 function: $PbLe(\{(assign[i][j], s[j]) | j = 0..n-1\}, l[i])\ \forall i = 0, \ldots, m-1$

4.3.7 Constraints between path and assignment variables for the two solver strategy

These constraints are used to ensure consistency between path and assignments variables, ensuring that each item assigned to courier i+1 is leaved and reached once:
$\forall i = 0, \ldots, m-1\ \forall j = 0, \ldots n-1\ If\ assign[i][j] then\ path[i][j]! = j\ else\ path[i][j] == j$

4.3.8 Implied constraint

With the implied constraint we force each courier to have at least one item in its path:

- One solver formulation: $path[i][n]! = n\ \forall i = 0, \ldots, m-1$
- Two solvers formulation: $\bigvee_{j=0}^{n-1} assign[i][j]\ \forall i = 0, \ldots, m-1$

4.3.9 Symmetry breaking constraints

We impose all the symmetry breaking constraints discussed in section 1.2 applying them in the different search strategies as already discussed in section 3.3.9; for the first, we also impose that the auxiliary sizes variables actually represent the loads carried by each courier

- – One solver formulation: $sizes[i] == \sum_{j=0..n-1, path[i][j]!=j} s[j]\ \forall i = 0, \ldots, m-1$
  – Two solvers formulation: $sizes[i] == \sum_{j=0..n-1, assign[i][j]} s[j]\ \forall i = 0, \ldots, m-1$

  $l[i] < l[k] \Rightarrow sizes[i] \leq sizes[k]\ \forall i = 0, \ldots, m-1,\ k = i+1, \ldots, m-1$
- $l[i] == l[j] \Rightarrow path[i][n] \leq path[j][n]\ \forall i = 1, \ldots, m-1,\ j = i+1, \ldots, m-1$
- $symmetric \Rightarrow path[i][n] \leq argmax(path[i])\ \forall i = 0, \ldots, m-1,$

4.4 Validation

4.4.1 Experimental design

We run all the experiments on an ASUS Zenbook UX431FN, Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz, 8,00 GB RAM with a time limit of 300 seconds.We used the same search strategies discussed for SAT in section 3.4.1

4.4.2 Experimental results

For instances from 1 to 10, we only show results for instances not always solved to optimality; for instances above 10, we show results only for instances for which at least one feasible solution was found.

Instance	Base	Implied	Implied Sym on S	Implied Sym on D	Implied Sym	Symmetry	2 Sol Base	2 Sol Implied	2 Sol Implied Sym on S	2 Sol Implied Sym on D	2 Sol Implied Sym	2 Sol Symmetry
7	234	368	353	368	353	315	168	184	216	184	216	187
9	**436**	**436**	**436**	**436**	**436**	**436**	**436**	**436**	**436**	**436**	**436**	506
10	**244**	**244**	**244**	**244**	**244**	**244**	**244**	**244**	**244**	**244**	**244**	436
13	N/A	N/A	N/A	N/A	N/A	N/A	1574	1104	1068	1126	994	1436
16	N/A	N/A	N/A	N/A	N/A	N/A	550	466	377	466	377	502

The two solvers strategy is the only one finding feasible solution for instances above 10. For those instances, both implied and symmetry breaking constraints are improving the performances, while worsening them for instance 7. As for SAT, it is difficult to make more significant and general assumptions due to the small number of instances not always solved to optimality.

5 MIP

We implemented our solution for the MCP in MIP using the solver independent language AMPL and its Python API, in order to compare performances with different solvers.

5.1 Decision variables

- Binary tensor path $\in \{0,1\}^{m\times(n+1)\times(n+1)}$ where path[i,j,k]==1 if and only if courier i goes from node j to node k.
- Positive integer array order $\in N^n$ used as auxiliary variable to implement MTZ sub-tour elimination in Hamiltonian sub-cycles.
- Positive integer array size $\in N^n$ where size[i] represent the actual load carried by courier i.

- Positive integer variable object $\in N^n$ that represents the longest distance traveled by any courier. We set the upper and lower bounds for this variable as discussed in section 1.1.

5.2 Objective function

The objective function to minimize just returns the object variable.

5.3 Constraints

5.3.1 Single departure constraint

Each node leaved once:

$$\sum_{i=1..m,k=1..n+1} path[i,j,k] == 1 \; \forall j = 1,\ldots,n$$

5.3.2 Single arrival constraint

Each node reached once:

$$\sum_{i=1..m,k=1..n+1} path[i,k,j] == 1 \; \forall j = 1,\ldots,n$$

5.3.3 Starting position constraint

Each path must start at the depot (for empty couriers we can have path[i,n+1,n+1]==1):

$$\sum_{j=1..n+1} path[i,n+1,j] == 1 \; \forall i = 1,\ldots,m$$

5.3.4 Ending position constraint

Each path must finish at the depot (for empty couriers we can have path[i,n+1,n+1]==1):

$$\sum_{j=1..n+1} path[i,j,n+1] == 1 \; \forall i = 1,\ldots,m$$

5.3.5 Zero diagonal constraint

We make impossible to leave and reach the same node, except for the depot:

$$path[i,j,j] == 0 \; \forall i = 1,\ldots,m \; \forall j = 1,\ldots,n$$

5.3.6 Courier consistency constraint

Each node must be reached and leaved by the same courier:

$$\sum_{k=1..n+1} path[i,j,k] == \sum_{k=1..n+1} path[i,k,j] \; \forall i = 1,\ldots,m \; \forall j = 1,\ldots,n$$

5.3.7 Sub-tour elimination constraints

To avoid sub-tours we implement MTZ constraints, using order variables and a big-M parameter for linearization set to n.

- $order[j] \leq 1 + M \times (1 - path[i, n+1, j])\ \forall i = 1, \ldots, m\ \forall j = 1, \ldots, n$
- $order[j] \leq order[k] + 1 + M \times (1 - path[i, j, k])\ \forall i = 1, \ldots, m\ \forall j = 1, \ldots, n\ \forall k = 1, \ldots, n$
- $order[j] \geq order[k] + 1 - M \times (1 - path[i, k, j])\ \forall i = 1, \ldots, m\ \forall j = 1, \ldots, n\ \forall k = 1, \ldots, n$

5.3.8 Constraints on loads

- $size[i] == \sum_{j=1..n+1, k=1..n} path[i, k, j] \times s[k]\ \forall i = 1, \ldots, m$
- $size[i] \leq l[i]\ \forall i = 1, \ldots, m$

5.3.9 Constraint on object variable

We force the object variable to be greater or equal then all the distances traveled by couriers:

$$\sum_{j=1..n+1, k=1..n+1} path[i, j, k] \times D[j][k] \leq object\ \forall i = 1, \ldots, m$$

5.3.10 Implied constraint

To enforce that each courier carries at least one item, we just rewrite constraints in section 5.3.3 and 5.3.4 with $j = 1, \ldots n$ and impose the constraint in section 5.3.5 also to the depot.

5.3.11 Symmetry breaking constraints

We impose all the symmetry breaking constraints discussed in section 1.2:

- We impose this constraint $\forall i = 1, \ldots, m\ \forall j = i+1, \ldots, m$, when we know that $l[i] < l[j]$: $size[i] \leq size[j]$
- We impose these constraints $\forall i = 1, \ldots, m\ \forall j = i+1, \ldots, m$, when we know that $l[i] == l[j]$: $\sum_{k=1..n} path[i, n+1, k] \times k \leq \sum_{k=1..n} path[j, n+1, k] \times k$
- We impose these constraints $\forall i = 1, \ldots, m$ when we know that the matrix of the distances is symmetrical: $\sum_{j=1..n} path[i, n+1, j] \times j \leq \sum_{j=1..n} path[i, j, n+1] \times j$

5.4 Validation

5.4.1 Experimental design

We run all the experiments on an ASUS Zenbook UX431FN, Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz, 8,00 GB RAM with a time limit of 300 seconds on two different solvers: Gurobi and Highs.

5.4.2 Experimental results

For instances from 1 to 10, all the instance were solved to optimality, so we only show instances above 10 for which at least one feasible solution was found.

Instance	Gr Base	Gr Implied	Gr Implied Sym on D	Gr Implied Sym on S	G Implied Sym	Highs Base	Highs Implied	Highs Implied Sym on D	Highs Implied Sym on S	Highs Implied Sym
12	555	694	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
13	482	464	530	570	524	N/A	N/A	N/A	N/A	N/A
16	**286**	**286**	**286**	N/A	N/A	N/A	N/A	N/A	N/A	N/A
19	595	**334**	481	N/A	N/A	N/A	N/A	N/A	N/A	N/A

While both solvers were able to always prove the optimality of the results for the smaller instances, Gurobi outperformed Highs for the greater ones, even certifying the optimal values for instances 16 and 19. For Gurobi, the implied constraint generally improved the performances, while the symmetry breaking constraints, especially the ones on the couriers capacities, have make them worse.

6 Conclusion

Among all the approaches, the best results were obtained by CP, while the others struggled more to find feasible solutions for the bigger instances. In our opinion, the most determining factor for the performances of the models was the time limit, which has penalized in most of the cases models that were reducing too aggressively the solution space, not allowing the solver to find feasible solutions in a small amount of time. This can be seen by the overall negative impact of the symmetry breaking constraints (especially the ones on the couriers capacities).

References

[1] Fadi A. Aloul, *Search techniques for SAT-based Boolean optimization*, Journal of the Franklin Institute, 2006

[2] Christian Prins, Philippe Lacomme, Caroline Prodhon, *Order-first split-second methods for vehicle routing problems: A review*, Transportation Research Part C: Emerging Technologies, 2014

[3] T. Sawik *A note on the Miller-Tucker-Zemlin model for the asymmetric traveling salesman problem*, Bulletin of the Polish Academy of Sciences, Technical Sciences 64, 2016

[4] *Numerical accuracy*, https://mp.ampl.com/modeling-numerics.html