Multiple couriers planning problem

Edoardo Saturno (edoardo.saturno@studio.unibo.it)
Guglielmo Biagini (guglielmo.biagini@studio.unibo.it)
Riccardo Marvasi (riccardo.marvasi@studio.unibo.it)

September 3, 2023

1 Introduction

This report describes the implementation of three distinct models (CP, SMT and MIP) for optimizing the **multiple couriers planning problem**. Some **constant parameters** are shared by all three models: the *distance matrix* D of size $(n+1) \times (n+1)$ (where n is the number of items), the vector of *couriers load* l of size m (number of couriers) and the vector of items' load s of size n. For all the three models, couriers have been sorted by their load size in a decreasing way, so that, if x_i is the load size of courier i it is true that:

$$x_i \geq x_{i+1} \geq x_{i+2} \geq \cdots \geq x_m \tag{1}$$

This simple pre-processing step allowed us to define some of the symmetry breaking constraints.
The objective function, that is roughly the same for all the models (only implemented differently), is defined as:

$$max(\text{PathCost})$$

where PathCost is a 1D array of length m, where each element indicates the total path traveled by a courier. For what concerns the CP model all three of us worked together, SMT was handled by Riccardo and Edoardo while the MIP model was developed by Guglielmo. For all the three models, anyway, there has been a constant exchange of information and ideas, that were then specifically implemented singularly.

2 CP Model

2.1 Decision Variables

- Path : A matrix with m rows and cap columns, in which each element can assume a value going from 0 to n. It is used to construct the path of each courier: each row corresponds to a courier and the values present on that

row are the items carried by him, delivered in that order.
The zeros present on the rows have no meaning, they are used just to "fill" the matrix.
N.B. cap is an integer, u.b. to the number of items taken by each courier. If the lowest value of the vector l is greater than the highest value of the vector s (i.e. every courier can carry at least one item, because even the heaviest item is still lower than the lowest load size), cap is equal to $n - m + 1$, otherwise it is equal to n.
The matrix can be represented as:

$$\text{Path} = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,\text{cap}} \\ p_{1,2} & p_{2,2} & \cdots & p_{2,\text{cap}} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m,1} & p_{m,2} & \cdots & p_{m,\text{cap}} \end{bmatrix} \tag{2}$$

If, for example, courier 1 has to deliver items 6 and 5 and cap = 4 the first row of Path will be:

$$\text{Path}[1] = \begin{bmatrix} 6 & 5 & 0 & 0 \end{bmatrix} \tag{3}$$

The couriers start from the origin point and proceed to all the distributions points present on the row until the first occurrence of a 0. At that point it is computed the distance from the last distribution point visited and the origin.

2.2 Objective Function

The objective function, as for the other models, is to minimize max(PathCost) (the definition of the objective variable has been already discussed in section 1.1). In particular, for the CP model, it is defined as:

$$\text{PathCost[i]} = D_{n+1,i_1} + \sum_{j=1}^{n} (D_{i_j,i_{j+1}}) + D_{i_n,n+1} \tag{4}$$

in which $(i_1, \cdots, i_n)$ are the n items assigned to courier i and $D_{a,b}$ represents the distance between the distribution point a and the distribution point b. So, for example, if items x, y and z are assigned to the courier i, its corespondent PathCost is computed as:

$$\text{PathCost[i]} = D_{n+1,x} + D_{x,y} + D_{y,z} + D_{z,n+1} \tag{5}$$

PathCost[i] can assume a value that goes from 0 (when a courier does not take any item) to $\max(D) * (n+1)$, because the worst possible case is when a courier has all the items (n) multiplied by the 'worst' distance, i.e. the maximum of D. Minimizing PathCost allows to find the optimal solution in terms of total distance travelled singularly by each courier.

2.3 Constraints

We defined several constraints for the model:

- The first constraint stands that if all the couriers can take any item (see the definition of cap in section 1.1), then **each courier must have at least one item**.

$$\text{if cap} == n - m + 1 \text{ then forall } (i \text{ in } 1..m) \text{ (nItem}[i] >= 1) \tag{6}$$

- The **total load of each courier i must be less or equal than its maximum load size**.

$$\text{forall } (i \text{ in CCOURIER}) \text{ (TotLoad[i]} \leq l[i]) \tag{7}$$

- We impose that in the whole matrix there must be exactly n elements different from zero.

$$\text{count(i in 1..m, k in 1..cap)(Path[i,k]!=0)=n} \tag{8}$$

- **The sum of Path's elements must be equal to the sum of the items' number**

$$\text{sum}(i \text{ in} 1..m,\ k \text{ in} 1..\text{cap})(\text{Path[i,k]})) == \frac{n * (n+1)}{2} \tag{9}$$

- **Each item must be assigned only once**

$$\text{allDifferentExcept(Path,S)} \tag{10}$$

2.3.1 Symmetry breaking constraints:

The way in which we defined Path (with all the non-zero elements aligned on the left) allowed us to eliminate a good number of symmetric solutions. In fact, for example, consider those three possible versions of the same row of Path:

$$\begin{bmatrix}5 & 2 & 0 & 0\end{bmatrix} \tag{11}$$

$$\begin{bmatrix}0 & 5 & 2 & 0\end{bmatrix} \tag{12}$$

$$\begin{bmatrix}0 & 0 & 5 & 2\end{bmatrix} \tag{13}$$

They are exactly the same, as the courier i delivers items 5 and 2 in this order. So, without constructing the matrix Path in the way we did, we would obtain many symmetric solutions that would slow down the solver enormously.

- **Courier x_i must have more items than courier x_{i+1}.**

$$\text{forall}(i \text{ in } 1..(m-1))(\text{nItem[i]} \geq \text{nItem[i+1]}) \tag{14}$$

This constraint is possible since couriers are sorted in decreasing order by their load size. This constraint allows to break many symmetries, avoiding to explore all those solutions in which items assigned were simply exchanged by couriers, with no change in terms of PathCost.

- **Courier x_i must carry more weight than courier x_{i+1}**

$$\text{forall}(i \text{ in } 1..(m-1))(\text{TotLoad[i]} \geq \text{TotLoad[i+1]}) \tag{15}$$

The motivations and reasoning that led to the formulation of this constraint are the same of the one above.

2.3.2 Implied constraints

- **First courier must have a number of items greater or equal to the remaining items divided by the remaining couriers** (if it's possible).

$$\text{if cap} == (n-m+1) \text{ then nItem[1]} \geq (n - \text{nItem[1]}) \text{ div } (m-1) \tag{16}$$

This constraint is implied by equation (14) and by equation (5). It speeds up the computation.

- **Constraint on the number of items assigned**:

$$\text{if cap} == (n-m+1) \text{ then forall}(i \text{ in } 2..m-1)(\text{nItem[i]} \leq (n \text{ div } i)) \tag{17}$$

If it is the case in which all the couriers must have at least one item, then the number of items taken by a courier must be lower or equal than the total number of items divided by the number of the courier. This constraint is again implied by the first symmetry breaking constraint and by the one that assigns at least one item to all the couriers. Calling x_i the number of items assigned to the i-th courier we can say that:

$$\text{if cap} == (n-m+1) \text{ then forall}(i \text{ in } 2..m-1)(\text{nItem[i]} \leq (n \text{ div } i)) \tag{18}$$

So, if we can deduce that if:

$$x_i = \frac{n}{i} \Rightarrow x_{i-1} \geq \frac{n}{i},\ x_{i-2} \geq \frac{n}{i},\ \cdots \tag{19}$$

At this point, summing all the items carried by the first i couriers it comes out that:

$$x_1 + x_2 + \ \cdots\ x_i \geq i\frac{n}{i} \equiv n \tag{20}$$

meaning that the couriers left have no items to take, and this goes against the first constraint described. So it helps to speed up the computation process by quickly eliminating some solutions that would have been eliminated later anyway.

2.4 Validation

To replicate the results showed in "table 1" run the following Python code: "MiniZinc_model.py". This script will run the model on all the 21 instances, both using Gecod 6.3.0 and then Chuffed 0.11.0, and output the results as

.json files. The heuristic used is dom_w_deg, that chooses the variable with the smallest value of domain size divided by weighted degree, which is the number of times it has been in a constraint that caused failure earlier in the search. For the Gecode solver the indomain_random method,which assigns the variable a random value from its domain, was used, meanwhile for the Chuffed solver, since we can't use the random one, the indomain_min method, which assigns to the variables its smallest domain variable, was used. We performed the experiments on a computer with an Intel(R) Core i5 with 2.11 GHz, 8 Gb of Ram and 64 bits Windows 10 operating system. The version of MiniZinc we used is MiniZinc 2.7.6.
The results presented in Table 1 show that the model successfully manage to solve the first 10 instances using symmetry breaking constraints and implied constraints both using Gecode and Chuffed solvers, on the other hand there were some problems without those constraints in terms of optimality.

2.4.1 Experimental Results

ID	Gecode	Gecode + SB + IC	Chuffed	Chuffed + SB + IC
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	167	167	167	167
8	186	**186**	**186**	**186**
9	436	**436**	436	**436**
10	244	**244**	244	**244**
11	1524	1372	N/A	N/A
12	1444	1558	N/A	N/A
13	1476	N/A	N/A	N/A
14	1614	N/A	N/A	N/A
15	1341	N/A	N/A	N/A
16	518	1297	N/A	N/A
17	N/A	N/A	N/A	N/A
18	1759	N/A	N/A	N/A
19	495	N/A	N/A	N/A
20	N/A	N/A	N/A	N/A
21	2166	3186	N/A	N/A

Table 1: Results using Gecode and Chuffed, with and without breaking symmetries constraints (SB) and implied constraints (IC).

3 SMT Model

3.1 Decision Variables

PathCost and cap,defined as in section 1.1. In addiction to those, we defined also:

- Assignments : A 1D array of length n, where each position of the array is an item and the value corresponding to the particular index is the courier that carries it. Its elements have lower bound 1 and upper bound m. It has the form

$$\text{Assignments} = \begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix} \tag{21}$$

  in which, if $a_1 = 3$ it means that the first item is given to the third courier.

- Path: Defined exactly as for the CP model (see section 2.1).

- maximumWeight : A 1D array of length 2 that contains, the heaviest item of s and its index.

- TotWeight An integer indicating the total weight carried by a courier.

- count: an integer indicating the total number of items carried by a courier.

- meanItem: An integer computed as n div m (integer division).

- maximumVector : A vector of length n in which each element is an array composed of two elements: the first element is the i-th delivery point and the second is the furthest point from it. So for example:

$$\text{maximumVector}[1] = [1,\ 3] \tag{22}$$

  means that the delivery point with the maximum distance from 1 is 3. This vector is useful to avoid to explore those solutions in which a courier has to deliver two items with distributions points very far from eachother.

3.2 Objective Function

The objective function is to minimize max(PathCost), exactly as in CP.

3.3 Constraints

3.3.1 Constraints on Assignments

- **Upper and lower bound on the number of items assigned to each courier.**

$$\text{optimizer.add(And(count} \leq \text{cap, count} \geq 1) \tag{23}$$

  The lower bound, as for the CP model, is 0 if it stands that cap == n, it is 1 otherwise.

- **TotLoad of each courier must be less or equal than the corespondent maximum load size**, as required by the problem.

$$\text{optimizer.add(totWeight[i-1]} \leq \text{l[i-1])} \tag{24}$$

- **Heaviest item is always assigned to the first courier** (which has the largest load size).

$$\text{optimizer.add(assignments[maximumWeight[1]]==1)} \tag{25}$$

3.3.2 Constraints to avoid worst couplings

These constraints have been thought to avoid to consider those solutions in which a courier has to deliver two items with distributions points very far from each other, which are surely not efficient in terms of path cost.

- We iterate through all the vectors that compose maximumVector and impose that each couple of the vectors must be assigned to different couriers. In the following equations item is the i-th element of maximumVector.

$$\text{optimizer.addSoft(assignments[item[0]]!=assignments[item[1]])} \tag{26}$$

  By doing so we make sure that solutions where two items very far one from eachother are assigned to the same courier are pruned. We defined it as a "soft constraint", because sometimes it is too aggressive and risks to render 'unsat' some instances, mainly when n is low.

3.3.3 Symmetry breaking constraints

As in the CP model, the matrix Path has been defined in a way to have all the non-zero elements aligned on its left side, in order to avoid the type of symmetry described in section 2.1.

- We impose that the **total weight of a courier i must be higher or equal than the total weight of courier $i+1$**.

$$\text{optimizer.add(totWeight[i]} > \text{totWeight[i+1])} \tag{27}$$

  In this way we eliminate many symmetries, because if two couriers have a large load size and both of them can potentially bring all items, the solver does not explore solution where more items are assigned to the second courier.

- For the same reason we impose that the **number of items carried by courier i must greater or equal than the items carried by courier $i+1$**.

$$\text{optimizer.add(nItem[i]} \geq \text{nItem[i+1])} \tag{28}$$

- The **first courier must carry a number of items greater or equal than meanItem** (because this is the extreme case without going into symmetric assignments)

$$\text{optimizer.add}(\text{nItem}[0] \geq \text{meanItem}) \tag{29}$$

3.3.4 Implied constraints

- We iterate through all the 2D vectors that compose maximumVector. When the heaviest item is found, we assign the other index of the vector to the second courier. This constraint is obviously implied by equation 26.

- As in the CP model, we impose that the **number of items carried by courier i must be lower or equal than the total number of items divided by the number of the courier**.

$$\text{optimizer.add}(\text{nItem}[\text{i}] \leq \frac{n}{i+1}) \tag{30}$$

  Reasons behind this constraint are the same exposed in section 2.3. This constraint is implied by the fact that we imposed that nItem[i] $\geq$ nItem[i+1], but its presence is useful to speed up the computation.

3.4 Validation

To replicate the results showed in "table 2" run the following Python code: "smt_model.py". This script will run the model on all the 21 instances using the z3 4.12.1 solver and output the results as *.json* files. The hardware and the software were the same as the CP model. We can see that symmetry breaking constraint improve the results and we are able to obtain optimal solution for the first ten instances excluding the seventh one. On the other hand in the case of using only the implied constraint we obtain worse results and are able to obtain optimal solution only for the first, third, fifth and sixth instance.

3.4.1 Experimental Results

Instance	IC	IC + SB
1	**14**	**14**
2	N/A	**226**
3	**12**	**12**
4	N/A	**220**
5	**206**	**206**
6	**322**	**322**
7	N/A	N/A
8	N/A	**186**
9	N/A	**436**
10	N/A	**244**
11	N/A	N/A
⋮	⋮	⋮
21	N/A	N/A

Table 2: Results using Z3 solver with and without symmetry breaking constraints (SB).

4 MIP model

4.1 Decision Variables

The decision variables chosen for the MIP model are:

- Path : A tensor in which the first dimension is m and each element is a n+1 × n+1 matrix. So it has the form:

$$\text{Path} = \begin{bmatrix} \begin{bmatrix} p_{1,1,1} & p_{1,1,2} & \cdots & p_{1,1,n+1} \\ p_{1,2,1} & p_{1,2,2} & \cdots & p_{1,2,n+1} \\ \vdots & \vdots & \ddots & \vdots \\ p_{1,n+1,1} & p_{1,n+1,2} & \cdots & p_{1,n+1,n+1} \end{bmatrix} \\ \\ \vdots \quad \vdots \\ \\ \begin{bmatrix} p_{m,1,1} & p_{m,1,2} & \cdots & p_{m,1,n+1} \\ p_{m,2,1} & p_{m,2,2} & \cdots & p_{m,2,n+1} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m,n+1,1} & p_{m,n+1,2} & \cdots & p_{m,n+1,n+1} \end{bmatrix} \end{bmatrix} \tag{31}$$

  The matrices that compose the tensor are boolean matrices in which an element $p(v, i, j) = 1$ if courier v goes from i to j. The matrices have dimension $n + 1$ because from 1 to n we represent the distribution points of the items, index $n + 1$ is the origin.

- t : A time variable to avoid sub-tours.
- b : Binary variables introduced to linearize the max problem
- totload : An integer that indicates the total weight carried by a courier. Its upper bound is 0. It is defined as a weighted sum of the 1s present in the matrix of courier v:

  lpSum([(s[j]*path[v][i][j]) for i in Rows for j in range(0,n)]) (32)

  So if the v-th courier has its element $[i][j]$ equal to 1 we multiply it by the j-th item and so on with the other ones of the matrix.
- nItem : An 1D array composed of m elements. The value corresponding to the v-th index of nItems define the number of items carried by the courier v. It's defined as:

  nItem[v]=lpSum([1*(path[v][i][j]) for i in Rows for j in range(0,n)]) (33)

  So basically it checks, for each matrix v, how many 1s are present and that number is nItem[v]. On the matrices we check only among the first $n \times n$ elements in order to avoid considering the return to the origin. The lower bound of its elements is 0 and the upper bound is n.

4.2 Objective Function

As for the other models, the goal is to minimize max(PathCost). PathCost is defined exactly as totload but instead of multiplying the ones by the weight of the item it is multiplied by the distance between items i and j.

[(D[i][j] * path[v][i][j]) for i in Rows for j in Columns] (34)

Then we defined y as max(PathCost) linearizing the max problem [2].

4.3 Constraints

We now introduce the constraints used for MIP :

- **Each distribution point must be visited exactly once**. This is ensured by checking that in all the m matrices, for a fixed j and iterating on i exactly one value is equal to 1.

  lpSum([path[v][i][j] for i in Rows for v in Couriers]) == 1 (35)
- **If min(l) > max(s) each courier must have at least one item**.

  lpSum([path[v][n][j] for j in Columns]) == 1 (36)

To do so, we make sure that each courier leaves the origin, that is equivalent to stand that it delivers at least one item. **If the condition is not met, the constraint is softer**.

$$\text{lpSum}([\text{path}[v][n][j] \text{ for j in Columns}]) \leq 1 \tag{37}$$

- **Couriers cannot be stuck in a distribution point.**

  $$(\text{lpSum}([\text{path}[v][i][j] \text{ for i in Rows }]) - \text{lpSum}([\text{path}[v][j][i] \text{ for i in Columns }]))==0 \tag{38}$$

  To be sure of the fact that, if a courier reaches a distribution point i then it also leaves it, we define that subtraction with which we check that, given a courier v:

  $$\text{if path[v,i,j]==1, then also path[v, j, i] for another i must be 1.} \tag{39}$$

- **Each courier must have a totload lower or equal than its maximum load size**.

  $$totLoad[v] \leq l[v]$$

4.3.1 Symmetry breaking constraints

In a very similar way to the other models:

- **Courier v must carry more items than courier $v+1$.**

  $$\text{nItem[v]} \geq \text{nItem[v+1]} \tag{40}$$

- **Courier v must carry more weight than courier $v+1$.**

  $$\text{totLoad[v]} \geq \text{totLoad[v+1]} \tag{41}$$

4.3.2 Constraint to avoid sub-tours

A sub-tour by a courier is a round tour that returns back to where it starts, but does not visit all the distribution point of the packages carried by the courier. For instance if the courier k has the packages [1,3,4,7] a normal path for the courier could be:

$$(n+1) \to 1 \to 7 \to 3 \to 4 \to (n+1)$$

where n is the number of all the packages and n+1 represents the origin, but if we do not add a constraint on sub-tours we could end up with solutions like

$$(n+1) \to 1 \to (n+1), \;\; 7 \to 3 \to 4 \to 7$$

but this is not a suitable solution for the problem since the courier must do only one tour starting from the origin and going back to it. To avoid sub-tours we can use a formulation proposed by Miller, Tuckler and Zemlin [3]: introduce for each courier continuous decision variables representing times at which a location is visited are added. A variable for each location except origin node is added $t[v][i] = 1$ time at which location i is visited by courier v,$v = [1..m]$ and $i = [1, \dots n]$. Finally what is required are the constraint:

$$t_{v,j} > t_{v,i} \qquad if \quad path_{v,i,j} == 1 \tag{42}$$
$$t[v][j] >= t[v][i] - (2n) * (1 - path[v][i][j])$$

Lets consider the previous example to prove that this constraint avoid sub-tours:

$$path_{k,7,3} = 1 \quad \Rightarrow \quad t_{k,7} > t_{k,3}$$
$$path_{k,3,4} = 1 \quad \Rightarrow \quad t_{k,3} > t_{k,4}$$
$$path_{k,4,7} = 1 \quad \Rightarrow \quad t_{k,4} > t_{k,7}$$

But since $t_{k,3} > t_{k,4}$ it must be true that $t_{k,3} > t_{k,7}$ that is impossible.

4.4 Validation

To replicate the results showed in "table 3" run the following Python code: "MIP_model.py". This script will run the model on all the 21 instances, using CBC solver from the Pulp 2.7.0 library, and output the results as *.json* files. The hardware and the software were the same as the CP model. Results show how the model performs in a quite efficient way on the first ten instances, but cannot solve the more complex instances that go from 11 to 21. Even in this case, the presence of symmetry breaking constraints and implied constraints improves the efficiency of the model.

4.4.1 Experimental Results

5 Conclusions

We tried to solve the Multiple Couriers Planning problem using three different approaches (CP, SMT and MIP). To determine the performances of each approach we tested our models on 21 instances with increasing difficulty. The results showed that CP is the best approach since it solves (even if not always in an optimal way) 13 instances, while z3 and MIP stop at 10. In all three models was clear that the symmetry breaking constraints efficiently improved the results. The ouctcome of the experiments could be improved by using more powerful machines and more aggressive constraints: in this last case we could have encountered some problems because some good solutions could have been pruned.

Instance	w/out IC and SB	with IC and SB
1	**14**	**14**
2	**226**	**226**
3	**12**	**12**
4	220	**220**
5	**206**	**206**
6	**322**	**322**
7	186	183
8	186	**186**
9	436	436
10	244	**244**
11	N/A	N/A
⋮	⋮	⋮
21	N/A	N/A

Table 3: Results using CBC solver with and without implied constraints and symmetry breaking constraints

References

[1] Peter J. Stuckey, Kim Marriott, Guido Tack, The MiniZinc Handbook, 2020, https://www.minizinc.org/doc-2.7.6/en/index.html

[2] Roberto Amadini, CDMO slides, 2023 .

[3] Aayush Aggarwal, Techniques for subtour elimination in TSP: Theory and implementation in Python,2020, https://medium.com/swlh/techniques-for-subtour-elimination-in-traveling-salesman-problem-theory-and-implementation-in-71942e0baf0c