Multiple Couriers Planning
University of Bologna
CDMO Report

Kankana Ghosh, Luca Mongiello, Marco Solimè

February 12, 2024

Contacts
kankana.ghosh@studio.unibo.it
luca.mongiello2@studio.unibo.it
marco.solime@studio.unibo.it

1 Introduction

In this report, we propose and evaluate different methodologies for solving the **Multiple Couriers Planning** problem, a well-known NP-Hard problem in which a fleet of couriers must deliver a set of items to clients in an efficient and fair manner. The goal is to minimize the maximum distance of any courier while visiting intermediate nodes, hoping to achieve a fair balance in term of traveled distance among the drivers.

The instances are provided in *.dat* text files following a standard format. We refer to each parameter following this convention: m is number of couriers, n is the number of items to deliver, s is the size of each item, l is load capacity of each courier, and d is point-to-point distance matrix. Regarding the latter, the last row and column is reserved to the depot, so the final dimension is $(n+1)*(n+1)$.

1.1 Pre-processing phase

1.1.1 Lower/Upper Bounds

To achieve a fair division among drivers, we encourage each courier to leave the depot at least once. In practice, we consider the total distance a courier would cover if it were to reach the closest node to the depot and come back. So, as lower bound, we simply find the minimum distance from the depot to any other node scanning through the last row and column of the distance matrix. As the upper bound, we deploy a greedy algorithm based on **Nearest Neighbor Search**, as it is common in TSP-like problems.

1.1.2 Auxiliary matrices

During the preprocessing phase, we compute two Boolean matrices: is_bigger_mat and is_equal_mat. Both of them have size $m*m$, as each row/column corresponds to a courier. We compare each courier against all the others. We have

$$is\_bigger\_mat(k1, k2) = True \leftrightarrow l(k1) > l(k2)$$

meaning that we have positive values if courier $k1$ is bigger than $k2$.

Similarly, we have

$$is\_equal\_mat(k1, k2) = True \leftrightarrow l(k1) = l(k2)$$

keeping track of same-sized couriers. These two matrices help us to impose symmetry breaking constraints.

1.2 Project Organization

To efficiently manage the project, we equally distributed the workload among the team members. While collaboration and mutual assistance were encouraged,

allowing us to complete the project in roughly six months, challenges arose, including: the development for suitable models for each technology, coping with limited resources, navigating the intricacies of each chosen methodology having limited and sparse documentation, several trial and error to achieve optimal models.

2 CP Model

2.1 Decision variables

Before diving into the variable design, we first define a series of useful sets on which we built our variable and constraint definitions.

$$NODE = 1..n + 2 * m$$

We define a set of integers that ranges from 1 up to $n + 2 * m$. The idea is to reserve the first n numbers to the nodes representing clients, and the remaining numbers to the single depot.

$$START\_NODE = (n{+}1)..(n{+}m) \quad \textbf{and} \quad END\_NODE = (n{+}m{+}1)..(n{+}2{*}m)$$

We also create some additional sets to better define our model.

$$COURIER = 1..m \quad , \quad ITEM = 1..n \quad , \quad LOC = 1..(n + 1)$$

Regarding the depot node, we decided to create a series of "dummy" nodes that represent the starting and ending location. In more practical terms, we exploited the global constraint $circuit()$, that indeed adresses the problem of subtours and multi-item-deliveries. We then built our variables on top of the aforementioned sets.

next, **back** are arrays in $NODE$ that respectively represent the following and preceding node, representing the forward ($next$) and backward ($back$) path. The domain is once again $NODE$.

courier is an array in $NODE$ that represents the courier assignment for each node. The domain is $COURIER$, that lays in $1..m$.

int_load is an array in $NODE$ that represents the intermediate load of each courier, spanning from 0 to l.

traveled is an array in $COURIER$ that represent the traveled distance for each courier. Each variable lays between carefully precomputed lower and upper bounds.

2.2 Objective function

This is how we keep track of the traveled distance for each courier. Notice that with $courier(j) = i$ we filter out all paths covered by the other couriers.

$$traveled(i) = \sum_{j \in NODE | courier(j) = i} dx(j, next(j))$$

where $i \in COURIER$. Once we have all the traveled distances, we just impose $obj = max(traveled)$ and minimize that quantity.

2.3 Constraints

(0) Start-End Link: we link the starting nodes with the end nodes on the backward path.

$$back(i) = i + m - 1, i \in (n+2) \dots (n+m)$$

$$back(n+1) = n + 2 * m$$

(1) End-Start link: once a courier ends the tour, it must reach the starting node of the following courier.

$$next(i) = i - m + 1, i \in (n+m+1) \dots (n+2*m-1)$$

$$next(n+2*m) = n+1$$

(2) Fix start/end node: mark each start/end node with the corresponding courier.

$courier(i) = i - n, i \in START\_NODE$, $courier(i) = i - n - m, i \in END\_NODE$

(3) Empty start: initially, each courier is empty

$$int\_load(i) = 0, i \in START\_NODE$$

(4) Next-Back link: following nodes are linked to the previous nodes. This is the constraints that links together the two variables.

$$next(back(i)) = i, i \in NODE \quad \textbf{and} \quad back(next(i)) = i, i \in NODE$$

(5) TSP-like circuit: we avoid sub-tours and multiple deliveries at the same location.

$$circuit(next) \quad \textbf{and} \quad circuit(back)$$

(6) Path-consistency: each path is correctly assigned to each courier

$$courier(back(i)) = courier(i), i \in ITEM$$

$$courier(next(i)) = courier(i), i \in ITEM$$

(7) Intermediate-Weight: accumulate item weights along the path.

$$int\_load(i) = int\_load(next(i)), i \in START\_NODE$$

$$int\_load(i) + s(i) = int\_load(next(i)), i \in ITEM$$

(8) Capacity-Check: the accumulated weight must be lower than the courier capacity. Note that we are also checking if the intermediate load go over the maximum capacity of the couriers; this is to avoid dead-ends.

$$int\_load(i) \leq l(courier(i)), i \in ITEM$$

$$l(i) \geq int\_load(n + m + i), i \in COURIER$$

Implied constraints

We decided to use both the list of successors and predecessors *in tandem* as it allows us to express more high level constraints that we hope lead to more efficient solving, as the solver has a smaller search space to explore. Despite not being strictly necessary, we imposed the *circuit*() constraint upfront, both on *next* and *back* variables, following a more eager approach. Consequently, we had to link the *next* and *back* variables to be consistent and have an unique view on the forward/backward path.

Symmetry breaking constraints

(1) Bigger-sized couriers load more weight.: given two different couriers, the first one bigger than the other, compare the load in the ending nodes. The total load of the bigger courier must be bigger or equal than the second one.

$$(l(i) > l(j) \wedge i \neq j) \rightarrow int\_load(n + m + i) \geq int\_load(n + m + j)$$

where $i \in COURIER, j \in COURIER$, l is the load for each courier.

(2) Equally-sized couriers touch different nodes.: given two different couriers with same capacity, state that the path of the first one must be lexicographically less or equal than the second one. $eq(courier_1, courier_2)$ is a custom function that takes as input two couriers and returns True if they are the same, False otherwise.

$$(l(i) = l(j) \wedge j > i) \rightarrow lex\_less([eq(courier(k), i)], [eq(courier(w), j)])$$

where $i \in COURIER, j \in COURIER, k \in ITEM, w \in ITEM$

2.4 Validation

N.	Gecode (no SB)	Gecode (SB)	Chuffed (no SB)	Chuffed (SB)
1	**14**	**14**	**14**	**14**
2	235	226	**226**	**226**
3	**12**	**12**	**12**	**12**
4	220	**220**	220	**220**
5	**206**	**206**	**206**	**206**
6	322	**322**	**322**	**322**
7	203	236	167	225
8	256	-	186	186
9	436	-	-	300
10	270	-	244	244
11	1753	-	-	-
12	-	-	-	-
13	1282	1248	-	-
14	1776	-	-	-
15	1486	-	-	-
16	-	-	-	-
17	1523	-	-	-
18	1588	-	-	-
19/20	-	-	-	-
21	1687	-	-	-

Table 1: Best solutions found within time limit using Gecode and Chuffed, with and without symmetry breaking constraints, for all 21 instances.

Experimental design

To solve the instances, we created a MiniZinc model and solved it using both Gecode and Chuffed solvers. To assess the potential benefits of symmetry breaking constraints, we repeated the tests with and without them. We imposed a time limit of 5 minutes and reported the optimal (proved) solutions in bold.

Experimental results

The Gecode solver was able to find feasible solutions for both small and large instances. Interestingly, the symmetry breaking constraints seemed to help only for the first instances, and slowed down convergence speed for larger ones. The symmetry breaking variant of the Chuffed solver had slightly better performance, proving optimality for instance 4. However, neither variant of Chuffed was able to find a feasible solution for any instance from instance 11 onwards.

3 SMT Model

3.1 Decision variables

x is a 3D matrix of booleans of dimension $(n+1)*(n+1)*(m)$. It represents the connections between nodes for each courier. More specifically, $x(i,j,k) = True$ iff courier k goes from node i to node j.

u is an auxiliary variable of length $n+1$ of sort Int which is a weight array. This array solves the sub-tour problem. It says that if a courier leaves node i and reaches node j, the weight of j must be one unit more than node i.

dist is an array of size m of sort Int; it collects the total distance traveled by each courier.

tot_load is an array of size m of sort Int; it records the total load for each courier.

3.2 Objective function

As objective we create a variable *obj* of sort Int, and we impose it to be at least as big as all the variables in *dist* array. Then, we simply minimize this quantity.

3.3 Constraints

(0) Lower - Upper bound

$$dist(k) \leq upBound \wedge dist(k) \geq lowBound, k \in COURIER$$

(1) No Self-loop: the main diagonal of x for each courier must be zero.

$$\bigwedge_{i=0}^{n-1} \bigwedge_{k=1}^{m} \neg x(i,i,k)$$

(2) Each node is visited once: impose an *exactly_one* constraint as a conjunction of *at_most_one* and *at_least_one*.

$$\bigwedge_{j=0}^{n-1} exactly\_one(\{x(i,j,k) | i = 0..n, k = 0..(m-1)\})$$

$$\bigwedge_{j=0}^{n-1} exactly\_one(\{x(j,i,k) | i = 0..n, k = 0..(m-1)\})$$

(3) Leave once - Arrive once: given courier k, there must be only a 1 in the last row of x and in the last column of x, meaning that the courier must

arrive at the depot from only one node. n is a fixed constant and alternatively represents the last row and last column of the x matrix.

$$\bigwedge_{k=0}^{m-1} exactly\_one(\{x(n,j,k)|j=0..(n-1)\})$$

$$\bigwedge_{k=0}^{m-1} exactly\_one(\{x(j,n,k)|j=0..(n-1)\})$$

(4) Capacity Constraint: to constraint the $tot\_load(k)$ variable to be the total traveled distance for courier k, consider the k^{th} channel of the x matrix, and sum the distances from node i to j when $x(i,j,k) = True$.

$$\bigwedge_{k=0}^{m-1} tot\_load(k) \leq l(k)$$

where $l(k)$ is the total capacity for courier k.

(5) N. incoming arcs = N. outgoing arcs: for each given courier and node, there must be one way in and one way out. The constraint calculates all the incoming and outgoing arcs for any given node and impose them to be equal. This holds for any fixed $j = \{0, 1, \ldots, n\}$, $k \in \{0, 1, \ldots, m-1\}$.

$$in\_arcs = \sum_{i=0}^{n} ite(x(i,j,k),1,0) \quad \textbf{and} \quad out\_arcs = \sum_{i=0}^{n} ite(x(j,i,k),1,0)$$

$$in\_arcs = out\_arc$$

(6) Subtour elimination: to avoid couriers form subtours – eg. 1→ 4, 4→ 1, 2→ 3, 3→ 2 – set an auxiliary variable array named u in tandem with the so-called **"Big-M trick"**.

$$u(i) + 1 \leq u(j) + M * (1 - ite(x(i,j,k),1,0))$$

$$u(i) + 1 \geq u(j) - M * (1 - ite(x(i,j,k),1,0))$$

(7) Traveled-Distances: given courier k, scan the k^{th} channel in matrix x and simply sum the distances of the active links. Impose also that the objective variable obj must be bigger than all the distances at the same time. This is the quantity to then minimize.

$$\bigwedge_{k=0}^{m-1} dist(k) = \sum_{i,j=0..n} ite(x(i,j,k),d(i,j),0) \quad , \quad \bigwedge_{k=0}^{m-1} obj \geq dist(k)$$

Symmetry breaking constraints

(1) Bigger-sized couriers load more weight. Create a Boolean matrix named is_bigger_mat with dimensions $m * m$ and then impose that bigger-sized couriers load more weight.

$$is\_bigger\_mat(k_1, k_2) = True \land k_1 \neq k_2 \rightarrow tot\_load(k_1) \geq tot\_load(k_2)$$

(2) Equally-sized couriers touch different nodes. Create a matrix named is_equal_mat with dimensions $m*m$ and then impose that equally-sized couriers touch different nodes.

$$is\_eq\_mat(k_1, k_2) = True \land k1 > k2 \rightarrow \neg(x(i, j, k_1) \land x(i, j, k_2))$$

3.4 Validation

Experimental Design

We coded the model using the Z3 Python API (Z3Py). Optimal solution are highlighted in bold.

Experimental Results

N.	Z3 w/out SB	Z3 + SB
1	**14**	**14**
2	282	**226**
3	12	**12**
4	-	220
5	**206**	**206**
6	322	**322**
7	-	174
8	251	**186**
9	-	436
10	-	245
11-21	-	-

Table 2: Results obtained using Z3 solver with and without symmetries constraints.

The impact of symmetry breaking constraints is immediately apparent, as they allow the solver to find solutions for up to instance 10. For instance 11 and beyond, both variants struggle to find even feasible solutions and sometimes even run out of memory when defining the model.

4 MIP Model

4.1 Decision variables

x is the usual 3D Boolean matrix of shape $(n+1)*(n+1)*m$. We have $x(i,j,k) = 1$ iif courier k goes from node i to node j.

u is the auxiliary array of integers variables of size $n+1$ used to solve the sub-tour problem. Each variable ranges from 1 up to $n+1$. As a reference point, the depot weight is set to 1, whereas the weights of the traversed nodes are gradually increased by one.

dist is an array of integer variables of length m that contains the traveled distances for each courier.

tot_load is an array of integer variables of length m that represents the total load of each courier at the end of the route.

4.2 Objective function

We simply constraint an integer variable z to be greater or equal than all the variables in the distance array. Then, we minimize this quantity.

$$z \geq dist(i), i = 0..(m-1) \quad , \quad minimize(z)$$

4.3 Constraints

(1) No-Self loops: the main diagonal of the x matrix must be set to zero for each courier.

$$x(i,i,k) = 0, i = 0..n, k = 0..(m-1)$$

(2) One Node - One Visit: the implied constraint scans all the x matrix row by row and make sure the sum is 1. Does not include the last column since at length all the couriers must converge to the depot and does not include the last row since all the couriers start from the depot.

$$\sum_{k=0..(m-1), i=0..n} x(i,j,k) = 1, j = 0..(n-1)$$

(3) Arrive and depart from depot once: alternatively fix the last row and last column for each courier, and state that the courier must leave the starting node (or ending node) exactly once. n is is a fixed constant and represents the last row/column of the x matrix.

$$\sum_{j=0..(n-1)} x(n,j,k) = 1, k = 0..(m-1) \quad \textbf{and} \quad \sum_{i=0..(n-1)} x(i,n,k) = 1, k = 0..(m-1)$$

(4) Load constraints: impose that the total load of courier k must be equal to the sum of the weights of the items that it distributes. Then, constraint the total load of each courier to be less or equal then the total capacity it can bear.

$$tot\_load(k) = \sum_{i=0..n, j=0..(n-1)} x(i,j,k) * s(j), k = 0..(m-1)$$

$$tot\_load(k) \leq l(k), k = 0..(m-1)$$

(5) N. arcs in is equal to the N. arcs out: if courier k_1 reaches node i once, courier k_1 must also leave once. So, the sum of each column must be equal to the sum of the respective row.

$$\sum_{i=0..n} x(i,j,k) = \sum_{i=0..n} x(j,i,k) \quad , \quad j = 0..n, k = 0..(m-1)$$

(6) Subtour elimination with Big-M trick: state that $u(i)+1 = u(j)$ when $x(i,j,k) = 1$ for courier k. When $x(i,j,k)$ is active, M is reduced to zero and the constraints hold. Instead, when $x(i,j,k) = 0$, the constraints are relaxed. This way, we force each courier to visit nodes of increasing weights, forbidding them to retrace already visited nodes. Set also the depot weight to 1.

$$u(i)+1 \leq u(j) + M*(1-x(i,j,k)) \quad \textbf{and} \quad u(i)+1 \geq u(j) - M*(1-x(i,j,k))$$

$$u(n) = 1 \quad , \quad i,j = 0..(n-1), k = 0..(m-1)$$

(7) Keep track of traveled distances: the distance of courier k must be equal to the summation of the distances between the nodes it traversed. Then, impose that an integer variable z must be bigger of all the distances.

$$dist(k) = \sum_{i,j=0..n} x(i,j,k) * d(i,j) \quad , \quad z \geq dist(k) \quad , \quad k = 0..(m-1)$$

Symmetry breaking constraints

(1) Bigger-sized couriers load more weight. Create a Boolean matrix named is_bigger_mat with dimensions $m * m$ and then impose that bigger-sized couriers load more weight.

$$is\_bigger\_mat(k1, k2) = True \wedge k1 \neq k2 \rightarrow tot\_load(k1) \geq tot\_load(k2)$$

(2) Equally-sized couriers touch different nodes. Impose that if different couriers with same capacity cannot simultaneously touch the same nodes.

$$x(i,j,k1) + x(i,j,k2) \leq 1 \quad k = 0..(m-1), \quad k1, k2 = 0..(m-1) \quad i,j = 0..n$$

4.4 Validation

Experimental design

To test the MIP models, we developed two models: one is written in **PuLP** language and uses the **CBC** solver and the other one uses the **Gurobi** solver, with the model based on the **Gurobi API**.

Experimental results

N.	Gurobi w/out SB	Gurobi + SB	PuLP w/out SB	PuLP + 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	**167**	**167**	198	216
8	**186**	**186**	**186**	186
9	**436**	**436**	436	436
10	**244**	**244**	244	323
11	747	533	-	-
12	348	986	-	-
13	500	420	758	844
14/15	-	-	-	-
16	286	302	-	-
17/18	-	-	-	-
19	340	611	-	-
20	-	-	-	-
21	576	-	-	-

Table 3: Table reporting the best solution (bold if optimal) for each instance.

The Gurobi solver without symmetry breaking constraints outperforms all other variants, finding solutions for even large instances (16, 19, and 21). Similarly to the CP and SMT implementations, the symmetry breaking version does not seem to bring any benefit. Regarding PuLP, the CBC solver without symmetries found and proved almost all instances up to number 10, but became impractical from instance 11 onward. Once again, symmetry constraints did not seem to help.

5 Conclusions

We solved the Multiple Courier Planning problem using different strategies, leveraging the strengths of CP modeling, SMT, and MIP. For each variant, we

developed an enhanced version that tries to break symmetries and potentially boost convergence speed. After assessing, evaluating, and comparing the models, the CP and MIP variants stood out from the rest.

It is not evident the benefit of symmetry breaking constraint variants. The SMT model, finds solutions even for mid-sized instances (9,10); while, the lightweight version struggles a lot even in the first cases (4). For the CP model, Chuffed+SB seems to work slightly better than the standard version; instead, Gecode+SB has the only effect to slow down convergence speed as the plain version finds solutions even for large instances (17,18,21). For the MIP model, it is difficult to spot a significant improvement of the enhanced models. In general, the CP Gecode lightweight model and the MIP Gecode models outperformed all other variants, as they provided decent results also for large instances. On the contrary, the SMT model with Z3 is probably the least efficient and prone to find solutions.

6 References

[1] NNS: Nearest Neighbour Search, www.thechalkface.net, *thechalkface*
[2] MiniZinc library, www.minizinc.org, *MiniZinc doc*
[3] Z3Py documentation, ericpony.github.io, *Github*
[4] Gurobi documentation, www.gurobi.com, *Gurobi doc*