CDMO project

Marco Sangiorgi marco.sangiorgi24@studio.unibo.it
Andrea Fossà andrea.fossa@studio.unibo.it

September 2024

1 Introduction

The following report describes different approaches to solve the Multiple Courier Planning problem (MCP) using Constraint Programming (CP), SAT and Mixed-Integer Programming (MIP) strategies. The results of different optimization approaches exploiting different formulations and underling solvers are discussed in the following chapters.

Our project took about a month of work during which the workload was split into CP for Marco Sangiorgi, SAT for Andre Fossagit s, while MIP being tackled together. Main difficulties were dealing with API interfaces of python wrapper of solvers, alongside with modeling strategies for bounds and symmetry breaking constraints.

1.1 Parameters

The input parameter of the problem are summarized in Table 1 below.

Name	**Dimension**	**Domain**	**Description**
m	1	$\mathbb{N}$	Number of couriers
n	1	$\mathbb{N}$	Number of items
l	m	$\mathbb{N}$	Couriers capacities
s	n	$\mathbb{N}$	Item sizes
D	$(n+1) \times (n+1)$	$\mathbb{N}$	Items distance martix

Table 1: MCP input parameters.

The distance matrix parameter D may not be symmetrical, and satisfies the triangle inequality relation where each path from a node i to node j is always shorter than passing from a third node k:

$$D_{i,j} \leq D_{i,k} + D_{k,j}$$

1.2 Bounds

For performance perspective, is crucial to set Upper Bounds ($\mathcal{UB}$) and Lower Bounds ($\mathcal{LB}$) of the variables within the model to limit their domain reducing the search space and the computational time.

Lower Bound For the Lower Bound of the objective function we consider one courier travelling the only item that yields the shortest travelling distance, namely the shortest path DEPOT→ ITEM→ DEPOT. Since the distance matrix D can be highly asymmetric, no further consideration are made and this simple bound is kept.

$$\mathcal{LB} = \min_i(D_{n+1,i} + D_{i,n+1})$$

Upper Bound For the Upper Bound a small CP model is used to find a quick initial solution (**CP/bin_pack.mzn**) where items get assigned to couriers satisfying the underling bin-packing problem of MCP. Then, a greedy algorithm calculate a short-as-possible cycle path c_k from/to DEPOT given each items partition p_k for each courier k by taking greedy minimal steps, as shown below.

$$P_k = card(p_k) \quad allitems = \bigcup_k p_k \quad \forall k \in 1..m$$

$$c_{k,0} = \min_i(D_{DEPOT,i}) \quad c_{k,i} = \min_{j_i}(Dc_{k,i-1}, j_i) \quad c_{k,P_k+1} = D_{c_{k,P_k},DEPOT} \quad \forall i \in p_k$$

Here given a courier k, the index j_i iterates over the remaining not-yet-chosen items, namely those in the set: $p_k/\{c_{k,0}, c_{k,1}, \ldots, c_{k,i-1}\}$.

On the other hand, having an $\mathcal{UB}$ of this kind is not good for restart-based search strategies (see discussion in chapter 2). We opted for a less robust but more effective in practice upper bound that consider a single courier travelling through all the item nodes, and selecting the minimum path c_1 in the same greedy way explained above.

$$\mathcal{UB} = \sum_{i=0}^{P_k} D_{c_{1,i},c_{1,i+1}}$$

1.3 Hardware

We run our experiments on an Intel i7-12700H 2.30GHz with 14 cores and 16GB RAM.

2 CP Model

The Constraint Programming approach heavily relies on the formulation of the classical Vehicle Routing Problem (VRP) described in [2] and [1]. These model

uses successor variables to model the path of each courier, allowing for efficient encoding of the Hamiltonian cycles travelled by each courier.

The overall view of the problem shift from multi-Hamiltonian Tours to single-Hamiltonian Tour introducing additional replicas of the depot node, transforming the original problem graph G in the extended graph G_R as explained in [2].

2.1 Variables Domains

For readability reasons the following domain sets has been used:

- $ITEMS$: goes from 1 to n , identifies each item to be reached by a number.
- $COURIERS$: goes from 1 to m, identifies each courier by a number.
- $NODES$: goes from 1 to $(n+2*m)$, identifies each node in the G_R graph.
- $DEPOT\_NODE$: goes from $(n+1)$ to $(n+2*m)$, identifies each replica depot node in the G_R graph.
- $START\_NODES$: goes from $(n+1)$ to $(n+m)$, identifies the m depot starting node in the G_R graph.
- END_NODES: goes from $(n+m+1)$ to $(n+2*m)$, identifies the m depot starting node in the G_R graph.
- END_NODES_NOLAST: goes from $(n+m+2)$ to $(n+2*m)$, same as END_NODES excluding first node, useful in assignment constraints later on.

2.2 Decision Variables

The CP model decision variables are an array of successors/predecessor for paths over graph G_R ($succ/pred$), an array of item assignments to courier ($bins$), an array of cumulative distance (d_cum), and the maximum distance travelled by any courier (max_d), as shown in Table 2.

Name	**Dimension**	**Domain**	**Meaning**
$succ$	$n+2*m$	$NODES$	$succ_j = i \iff$ Node i follows node j
$pred$	$n+2*m$	$NODES$	$pred_j = i \iff$ Node i precedes node j
$bins$	$n+2*m$	$COURIERS$	$bins_i = k \iff$ Courier k takes node i
d_cum	$n+2*m$	$[0, UB]$	$d\_cum_i = y \iff$ Courier $bins_i$ has travelled y to reach i.
max_d	1	$[LB, UB]$	$max\_d = y \iff \max_i(d\_cum_i) = y$

Table 2: CP Model Variables

2.3 Objective function

The objective function is to minimize the maximum distance travelled by any courier, which is the same as minimizing the maximum of the cumulative distances d_cum.

$$minimize \quad max\_d = \max_i(d\_cum_i)$$

2.4 Constraints

The model constraints classified as circuit, loads, channeling, redundant and symmetry breaking constraints are explained below.

2.4.1 Circuit Constraints

- Each end depot replica nodes in graph G_R must be linked to the following start depot node to ensure circuitation, this is modeled initializing the *succ* array so that the i^{th} node in END_NODES precedes the $(i+1)^{th}$ node in $START\_NODES$, except the first which precedes the m^{th} starting node:
  - $\forall i \in END\_NODES\_LAST$, $succ_i = i - m + 1$
  - $i = END\_NODES_m$, $succ_i = START\_NODES_1$
- The overall tour of couriers must be an Hamiltonian cycle, this is imposed by the global constraint *circuit* on the *succ* array:
  - $circuit(succ)$

2.4.2 Loads Constraints

- Each courier must choose which item with size s_i to travel to without exceeding its load capacity l_k, this is done by the global constraint *bin_packing_capa* partitioning in the $ITEMS$ positions of *bins*:
  - $bin\_packing\_capa(l, bins[ITEMS], s)$

2.4.3 Channeling Constraints

- All items assigned to a courier k must be in the same path travelled by courier k in array *succ*, meaning that starting from the k^{th} END_NODES each predecessor in *succ* must have same value k in *bins*. That said, taking a node i in END_NODES we can retrieve $k = i - n - m$. Recursively:
  - $\forall i \in END\_NODES, \quad bins_i = i - n - m$
  - $\forall i \in ITEMS, \quad bins_{succ_i} = bins_i$

2.4.4 Distance Constraints

- The distance travelled by each courier up to a certain node i is stored in the *d_cum* array, adding distance travelled at the node that precedes i and the lookup distance $D_{i,succ_i}$ paying attention not exceeding D dimension:
  - $\forall i \in START\_NODES, \quad (d\_cum_i = 0) \wedge (d\_cum_{succ_i} = D_{DEPOT,succ_i})$
  - $\forall i \in ITEMS, \quad d\_cum_{succ_i} = d\_cum_i + D_{i,min(succ_i,DEPOT)}$

2.4.5 Redundant constraints

The *pred* array is itself a redundant variable since it can be derived from the array *succ*, but it greatly helps propagation.

- Array *pred* follows the same circuit constraint has done for *succ*:
  - $circuit(pred)$
- The *pred* array is linked to the *succ* array by the the fact that the successor of the predecessor is the node itself and vice-versa.
  - $(succ_{pred_i} = i) \wedge (pred_{succ_i} = i)$
- Channeling constraints can be reversed using *pred* and enforced having the same meaning explained before:
  - $\forall i \in START\_NODES, \quad bins_i = i - n$
  - $\forall i \in ITEMS, \quad bins_{pred_i} = bins_i$

2.4.6 Symmetry breaking constraints

- From the model can be noticed that couriers with the same load capacities can be exchanged, so a global lexicographical order constraint on the *bins* array is added, as follows:
  - $\forall k1, k2 \in COURIERS \;\; s.t. \;\; (k1 < k2) \wedge (l_{k1} = l_{k2})$, $lex\_lesseq(bins_j == k2, bins_j == k1) \qquad \forall j \in NODES$

Heuristics

- Two couriers $k1, k2$ carrying a total respective amount of items $loads_{k1}, loads_{k2}$ can be exchanged if happens that $loads_{k1} \leq l_{k2} \wedge loads_{k2} \leq l_{k1}$. We apply global lexicographical order constraint to all those couriers having a capacity greater or equal to the lowest one that can carry all the items. Ideally, having many couriers this would lead the solver to ignore the permutation of many, if not all of them.
  - $\forall k \in COURIERS, \quad HEU = l_{\underset{k}{argmin}(l_k * card(COURIERS))}$
  - $\forall k1, k2 \in COURIERS \;\; s.t. \;\; (k1 < k2) \wedge (l_{k1} \geq HEU) \wedge (l_{k1} \geq HEU), \quad lex\_lesseq(bins_j == k2, bins_j == k1) \qquad \forall j \in NODES$

2.5 Validation

Experimental design The CP model implemented in MiniZinc is tested on Gecode and Chuffed solvers with and without Symmetry breaking constraints (SB) and Heuristics (Heu), using 4 different configuration. The first (**CP/mcp.mzn**), where a sequential search annotation is used to explore first *succ*, *pred* and then *bins* arrays following a *first_fail* policy with *indomain_split* placement strategy. The second (**CP/mcp_2R.mzn**), which add a Relax & Reconstruct annotation ($R\&R$) to implement a simple LNS strategy with 90% of *succ* array reused upon restart. The third (**CP/mcp_3R.mzn**), which add a Luby Restart annotation (Res, with $scale = 200$) to avoid getting stuck in deep part of search trees, largely improving results on hard instances. The fourth (**CP/mcp_D.mzn**), which use Decomposition (Des) of the main global constraint, mainly for Chuffed solver but with no improvements.

Experimental results

	w/out SB		w/SB							
			w/out Heu		w/Heu					
					w/out $R\&R$			w/$R\&R$		
					w/out Dec		w/Dec	w/out Res		w/Res
ID	Gcd	Cfd	Gcd	Cfd	Gcd	Cfd	Cfd	Gcd	Cfd	Gcd
1	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	14
2	**226**	**226**	**226**	**226**	**226**	**226**	**226**	**226**	**226**	226
3	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	12
4	**220**	**220**	**220**	**220**	**220**	**220**	**220**	**220**	**220**	220
5	**206**	**206**	**206**	**206**	**206**	**206**	**206**	**206**	**206**	206
6	**322**	**322**	**322**	**322**	**322**	**322**	**322**	**322**	**322**	322
7	167	167	167	167	167	167	167	167	**167**	167
8	**186**	**186**	**186**	**186**	**186**	**186**	**186**	**186**	**186**	186
9	436	436	436	436	436	**436**	**436**	436	**436**	436
10	224	224	224	224	224	**224**	**224**	224	**224**	224
11	783	1000	783	1000	783	1003	1011	783	N/A	393
12	381	364	381	364	381	364	364	381	N/A	364
13	1076	1076	1076	N/A	1076	N/A	N/A	1076	N/A	480
14	1073	1700	1073	1700	1073	1700	1700	1073	N/A	704
15	1232	1289	1232	1292	1232	1291	1341	1232	N/A	693
16	286	286	286	286	286	286	286	286	496	286
17	1481	1423	1481	1429	1481	1429	1456	1481	N/A	1107
18	827	1374	827	1374	827	1374	1374	827	N/A	590
19	390	334	390	334	390	334	334	390	N/A	390
20	1504	1503	1504	1503	1504	1503	1503	1504	N/A	1037
21	892	1335	892	1336	892	1336	1320	892	N/A	527

Table 3: CP models results In **bold** the optimal value, while underlined the best value with the shortest response time.

3 SAT Model

We use the solver Z3 through python environment to model SAT. One peculiarity of SAT is that only works with Boolean variables because it solve propositional logic formulas. Some encodings have been tried but the following was in the end the best one.

3.1 Decision variables

- Assignment it's a boolean matrix containing the relation courier-item: *a[i][j]* $= 1$ iff courier$i$ deliver j
- Route r is a matrix that store the connection used by the courier: if a courier goes from object j to k then it's 1. This is implemented for every courier so we obtain a 3D matrix *(m)x(n+1)x(n+1)* (+1 for the origin).
- Time t it's a boolean matrix that takes in to account the order: $t[i][j] = 1$ iff object$i$ it's been deliver as$j$*-th*
- L: loads of the courier
- S: sizes of the item
- D: distance matrix

3.2 Objective function

We struggled a lot to define the right way to find the upper bound. In the end, we used two methods: the first one is the standard and easy one, while the second is more complex and has been discussed in the introduction.
The standard one: $\sum_{i=m}^{n} \text{max\_distances}[i-1]$
Then we implemented the Linear and the Binary search.

- **Linear:** Adjust iteraively the UB (upper bound) thanks to the solver findings
- **Binary:** It adjusts the upper and lower bounds by checking feasibility for the midpoint of the current bounds, refining the search space iteratively

3.3 Constraints

Each object must be assigned to exactly one courier:

$$\forall j \in \{1, \ldots, n\}, \quad \text{exactly\_one}\left(\{a_{i,j} \mid i \in \{1, \ldots, m\}\}\right)$$

where exactly_one_seq($\{a_{i,j}\}$) ensures that exactly one courier is assigned to object j.

Maximum Load Constraints Each courier's load must not exceed their maximum capacity:

$$\text{courier\_loads}[i] = \sum_{j=0}^{n-1} a_{i,j} \cdot s[j]$$

$s[j]$ = size of the $(j+1)$-th item.

$$\forall i \in \{0, \ldots, m-1\}, \quad \text{courier\_loads}[i] \leq l_{\max}[i]$$

Delivery Time Constraints Each object must be delivered exactly once in the time table:

$$\forall i \in \{1, \ldots, n\}, \quad \text{exactly\_one}(t_i)$$

Route Constraints

- **Diagonal Constraints:** For each courier, the diagonal of r must be zero (I can't go from item i to itself):

  $$\forall i \in \{1, \ldots, m\}, \quad \forall j \in \{1, \ldots, n\}, \quad r_{i,j,j} = 0$$

- **Row Constraints:** For each courier i, the row $r_{i,j}$ has a 1 if and only if courier i delivers object j:

  $$\forall i \in \{1, \ldots, m\}, \quad \forall j \in \{1, \ldots, n\}, \quad a_{i,j} \rightarrow \text{exactly\_one\_seq}(r_{i,j})$$

  Means that if a courier has passed from j there must be a one in the line s.t the path j to another item it's present

  $$\forall i \in \{1, \ldots, m\}, \quad \forall j \in \{1, \ldots, n\}, \quad a_{i,j} = 0 \rightarrow \text{all\_false}(r_{i,j})$$

  Else it doesn't

  $$\forall i \in \{1, \ldots, m\}, \quad \text{exactly\_one\_seq}(r_{i,n})$$

  Every courier leave from the origin

- **Column Constraints:** For each courier i, the column $r_{i,k}$ has a 1 if and only if courier i reaches object k:

  $$\forall i \in \{1, \ldots, m\}, \quad \forall k \in \{1, \ldots, n\}, \quad a_{i,k} \rightarrow \text{exactly\_one\_seq}\left(\{r_{i,j,k} \mid j \in \{1, \ldots, n+1\}\}\right)$$

  $$\forall i \in \{1, \ldots, m\}, \quad \forall k \in \{1, \ldots, n\}, \quad a_{i,k} = 0 \rightarrow \text{all\_false}\left(\{r_{i,j,k} \mid j \in \{1, \ldots, n+1\}\}\right)$$

  $$\forall i \in \{1, \ldots, m\}, \quad \text{exactly\_one\_seq}\left(\{r_{i,j,n} \mid j \in \{1, \ldots, n+1\}\}\right)$$

- **Successive Constraints:** To avoid loops and ensure the correct sequence of deliveries:

$$\forall i \in \{1, \ldots, m\}, \quad \forall j, k \in \{1, \ldots, n\}, \quad r_{i,j,k} \rightarrow \text{successive}(t_j, t_k)$$

$$\forall i \in \{1, \ldots, m\}, \quad \forall j \in \{1, \ldots, n\}, \quad r_{i,n,j} \rightarrow t_{j,0}$$

  If a courier takes the path

$$j \rightarrow k$$

  there is the check that in the Time variables k is after j. And also that if there is

$$Origin \rightarrow j$$

  j has to be the first in the time table

Distance The distances done by a courier must be equal to the sum of every distance of the path between two objects that a courier has traversed.

$$\text{distances}[i] = \sum_{(j,k) \text{ for which } r[i,j,k]=1} D[j,k]$$

Symmetry breaking constraints Symmetry breaking is employed to reduce the exploration of equivalent symmetric solutions.

- The couriers are sorted by their maximum load
- During the solution process, the variables representing assignments (`a`) and routes (`r`) are re-ordered based on this permutation.
- After solving, the variables are re-mapped to their original order according to the initial permutation.

3.4 Validation

Experimental design For the Validation we run 2 different SAT models. The second model the sequential one uses two different solvers the second one uses two different solver one to find an assignment matrix a, then with that the second can develop an optimal routing plan. For each solver we used a lot of different modality for example changing the symmetry breaking constraint or the upper bound. The result contains the best combinations. In tabel 4 we can see the result.

4 MIP Model

4.1 Decision variables

- A: $A[i, j, k]$ is 1 if courier i travels from node j to node k; otherwise, it is 0. This binary tensor defines the route for each courier. Specifically, it indicates whether a courier travels from one node to another.

Methods List					
N	**base.sb**	**base.sb.UB**	**seq**	**seq.UB**	**seq.sb.UB**
1	**14**	**14**	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**	**226**
3	**12**	**12**	**12**	**12**	**12**
4	**220**	**220**	**220**	**220**	**220**
5	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**	**322**
7	187	224	202	223	303
8	**186**	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**	**244**

Table 4: Comparison of results across different methods. The values in bold indicate optimal results

- T: $T[k]$ represents the visit sequence of item k and is between 1 and n. This array encodes the sequence in which items are visited. For each item k, $T[k]$ denotes the order in which it is visited.
- **Obj**: The objective variable is bounded between `obj_lower_bound` and `dist_upper_bound`. This variable represents the maximum distance traveled among all couriers. The goal is to minimize this value.

4.2 Objective function

In the model Obj function is the objective function that the solver has to minimize. The function return the variable Obj.

4.3 Constraints

Objective Constraints **Constraint** `def_Obj`:

$$\text{Obj} \geq \text{tot\_dist}[i], \quad \forall i \in \text{m}$$

The objective must be at least the distance traveled by courier i.

Constraints to Create A

- **Constraint** `one_arrival_per_node`:

$$\sum_{i \in \text{m}, j \in \text{n}+1} A[i,j,k] = 1, \quad \forall k \in \text{ITEMS}$$

Each item k must be visited exactly once by a courier.

- **Constraint** `one_departure_per_node`:

$$\sum_{i \in \mathrm{m}, k \in \mathrm{n}+1} A[i, j, k] = 1, \quad \forall j \in \mathrm{n}$$

Just one courier departs from the $j - th$ delivery point.

- **Constraint** `origin_arrival` **e** `origin_departure`:

$$\sum_{j \in \mathrm{n}+1} A[i, j, n+1] = 1, \quad \forall i \in \mathrm{m}$$

Each courier must return to the origin (node $n + 1$) exactly once.

$$\sum_{k \in \mathrm{n}+1} A[i, n+1, k] = 1, \quad \forall i \in \mathrm{m}$$

Each courier must depart from the origin (node $n + 1$) exactly once.

- **Constraint** `no_self_loop`:

$$A[i, j, j] = 0, \quad \forall i \in \mathrm{m}, \forall j \in \mathrm{n}$$

No courier should have self-referential loops (i.e., they cannot stay at the same node).

- **Constraint** `balanced_flow`:

$$\sum_{k \in \mathrm{n}+1} A[i, k, j] = \sum_{k \in \mathrm{n}+1} A[i, j, k], \quad \forall i \in \mathrm{m}, \forall j \in \mathrm{n}$$

For each courier, the number of arrivals at and departures from each node must be balanced.

- **Constraint** `load_capacity`:

$$\sum_{j \in \mathrm{n}+1, k \in \mathrm{n}} A[i, j, k] \cdot \mathrm{size}[k] \leq \mathrm{capacity}[i], \quad \forall i \in \mathrm{m}$$

The load capacity of each courier must not be exceeded.

Constraints to Create T

- **Constraint** `first_visit`:

$$T[k] \leq 1 + 2n \cdot (1 - A[i, n+1, k]), \quad \forall i \in \mathrm{m}, \forall k \in \mathrm{n}$$

The item k must be the first visited if $A[i, n + 1, k] = 1$.

- **Constraint** `successive_visit_1`:

$$T[j] - T[k] \geq 1 - 2n \cdot (1 - A[i,k,j]), \quad \forall i \in \text{m}, \forall j,k \in \text{n}$$

If $A[i,k,j] = 1$, then item j must be visited immediately after k.

- **Constraint** `successive_visit_2`:

$$T[j] - T[k] \leq 1 + 2n \cdot (1 - A[i,k,j]), \quad \forall i \in \text{m}, \forall j,k \in \text{n}$$

Limits the sequence difference between j and k if $A[i,k,j] = 1$.

Implied constraints

- **Constraint** `implied_constraint`:

$$A[i, n+1, n+1] = 0, \quad \forall i \in \text{m}$$

Each courier must transport at least one item, so there should be no self-referential loops at the origin.

Symmetry Breaking Constraint

- **Constraint** `symmetry_breaking`:

$$\sum_{j \in \text{n}, k \in \text{n}} A[i,j,k] \cdot \text{size}[k] \geq \sum_{j \in \text{n}, k \in \text{n}} A[i+1,j,k] \cdot \text{size}[k], \quad \forall i \in \{1, \ldots, m-1\}$$

Ensures that the load of each courier is ordered according to capacity, reducing symmetry in the model.

4.4 Validation

The model was implemented in AMPL and executed using various solvers, including HiGHS, COIN Branch and Cut (CBC), Gurobi, and CPLEX. For each solver, the runtime was limited to 300 seconds.

Experimental design For the evaluation run the MIP model on our PC, using VScode. The key for AMPL (student license) is in the code.

Experimental results

5 Conclusions

We have concluded that the CP, as evidenced by the results, is the best strategy in our case. Future developments would regard the adoption of Clustering technics in the pre-processing step to reduce the number of items and the search space. Both results and time would benefit from it.

N	highs	cbc	gurobi	cplex
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**	217	**167**	**167**
8	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**
11	N/A	N/A	N/A	N/A
12	N/A	N/A	N/A	N/A
13	N/A	N/A	538	NaN/A

Table 5: Results for different solvers. Na indicates cases where no result was available.

References

[1] B. Backer, V. Furnon, P. Shaw, P. Kilby, and P. Prosser. Solving vehicle routing problems using constraint programming and metaheuristics. *J. Heuristics*, 6:501–523, 09 2000.

[2] L. Di Gaspero, A. Rendl, and T. Urli. Balancing bike sharing systems with constraint programming. *Constraints*, 21, 02 2015.