Multiple couriers planning problem

Lorenzo Cassano lorenzo.cassano2@studio.unibo.it
Jacopo D'Abramo jacopo.dabramo@studio.unibo.it
Kilian Tiziano Le Creurer kilian.lecreurer@studio.unibo.it

July 2023

1 Introduction

The following report contains the main ideas and strategies proposed in order to solve the multiple couriers planning problem using several optimization techniques. The approaches used to solve the task are widely different both regarding the formalization and the underlying solvers. Even so, there are some commonalities shared by all our approaches.

Regarding the completion time, the project took three month of non daily work. The general tasks related to the experiments with different solver and strategies were generally completed by a single element of the group, while the construction of the model were done working together. The main difficulties were mostly related to solving the hard instances using python based interfaces, such as z3 and pulp. Indeed, when the size of the instances grown all models suffered from ram completion issues when creating the variables and the constrained models.

In the next paragraphs are provided the main commonalities of of the encoding and solving procedure.

1.1 Variables

The input variables, namely the number of couriers m, the number of items n, the array of weights of each item w, the array of maximum load of each courier s and the matrix of distances D, for which the triangular inequality between distances is ensured by definition.

The model variables representing the total distance and weight for each courier, respectively d and l and the objective function, defined as $\max(d)$.
The only preprocessing step applied is been to sort in descending order the array s of maximum size. This step is been useful to introduce symmetry breaking

constraints efficiently. Then, a boolean variable is been introduced.

$$at = \begin{cases} true & \min(l) \geq \max(w) \\ false & otherwise \end{cases}$$

The value of at, namely all travel, is been extremely important because it guarantees that the optimal solution can be reached constraining all the couriers to travel.

1.2 Bounds

The bounds of the objective function are determined by some consideration on the structure of the problem; first, every distribution center must be visited, so a choice for the upper bound distance depends on the value of the at variable. First, let dp be the distance to complete a generic path. Let, d_i be the i^{th} row of D. We can derive from the structure of the matrix distance that:

$$d_p \leq \sum_{i \in path} max(d_i) = dp^u$$

Now, the bound is set by approximating the path in the case where even one courier could travel trough all the distribution center's and in the case where each courier is forced to carry an item. The trivial path to compute the bound is set as follows: the first $m-1$ items are assigned to a single courier, while all the others are assigned to the m^{th} one. The path traversed by the last one must be the longest possible one. Let dp^u be the upper bound for a path with $n-m+1$ items and d_i the i^{th} row of D. The upper bound for the objective becomes:

$$ub = \begin{cases} \max(dp_u) & at = True \\ \sum_i \max(d_i) & otherwise \end{cases}$$

The lower bound can be set by considering that the smallest possible distance is the one where a courier visits only a single distribution center, so let r_n, c_n be respectively the last row and the last column of D and let i_r, i_c be the indexes of the maximum for the row and the column.

$$lb = \max(c_{i_r} + \max(r), r_{i_c} + \max(c))$$

The array of distances d were bounded too. It kept the same upper bound of the objective, while the lower bound was selected again exploiting the flag at. More specifically the lower bound results to be

$$lb_d = \begin{cases} 0 & at = False \\ \min(path) & at = True \end{cases}$$

1.3 Constraints

A symmetry breaking constraint was set regardless of the approach. More precisely, the set of final loads w_k of each courier must be ordered. This constraint helps to reduce the size of the exploration tree deleting a certain number of equivalent solution.

$$w_k \geq w_{k+1}$$

A constraint which forces each courier to travel if *at* flag is true. This reduces significantly the search space keeping the solution optimal, indeed, given this assumption there must be at least an optimal assignment where every courier travels due to the triangular inequality.

1.4 Hardware

The experiments presented below, including Constraint Programming, SAT, SMT, and Linear Programming, were conducted on a computer with the following specifications:

- RAM: 8.0 GB
- Processor: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz
- Clock Speed: 2419 Mhz
- Cores: 4
- Logical Processors: 8

2 CP

Constraint programming is an optimization technique which involves enumerating all possible variable-value combinations via a systematic backtracking tree search. The constraint programming model were built using the minizinc language and then tested with two different solvers. Several encodings of the problem have been tried, the following is the final model.

2.1 Decision Variables

The proposed minizinc model is based on the following variables.

- An matrix $Asg \in \mathbb{N}^{(n+1)xm}$, with domain $[1, .., n+1]$ which represent the travel schedule for each courier. In general, the courier k goes from distribution center i to distribution center j if $Asg_{k,i} = j \; \forall i, j \in [1, .., n+1]$. So, the index represent the starting distribution center and the value the ending one.

- An array $packages \in \mathbb{N}^{n+1}$, with domain $[1, .., m]$ representing the courier assigned to each item. $loads_i = k$ if the i^{th} package is carried by the k^{th} courier.
- The array of distances, couriers load and the maximum as already introduced in Section 1.

2.2 Constraints

The constraints used in order to solve the task are as most as possible global, in order to exploit the strategies and the efficiency of the underlying solvers on them.

- The first ones are needed to guarantee the coherence of the Asg matrix with the task. First, each couriers schedule must is a subcircuit.

  $$subcircuit(Asg_k)\ \forall k \in [1, .., m]$$

  Then, the courier path must begin and end at the depot, if the courier travels.

  $$Asg_{k,n+1} \rightarrow count(Asg_k, n+1, 1)$$

  Finally, all distribution center must be visited once. Namely, all column of Asg must sum to one. Let c_i be the i^th column of Asg.

  $$count(Asg_i, i, courier - 1)\ \forall i \in [1, .., n+1]$$

- Then, there's the constraint on the weight limits, which is been implemented through the array $packages$. To do so, it's fundamental to channel Asg and $packages$.

  $$Asg_{k,i} \neq i \rightarrow packages_i = k\ \forall k \in 1..m\, \forall i \in 1..n$$

  $$bin\_packing\_capa(l, packages, w)$$

- Finally, the distance and the maximum has been added to the constraints set.

2.3 Validation

Experimental design

The CP model has been tested using two solvers which are:

- Gecode
- Chuffed

Moreover, all the instances has been run with and without symmetry breaking in order to make a comparison between different solvers with different configurations. The search stategies were explored and the more effective is been the first fail with domain minimum hierarchy.

Experimental results
In the following, there are the results using the two solvers with and without symmetry breaking.

ID	Chuffed + SB	Chuffed w/out SB	Gecode + SB	Gecode w/out 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**	**167**	**167**
8	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**
11	1199	1199	1114	1086
12	862	862	755	725
13	1356	1498	1594	1592
14	1762	1762	N/A	N/A
15	1393	1393	N/A	N/A
16	**286**	**286**	**286**	**286**
17	1547	1560	N/A	N/A
18	1573	1573	N/A	N/A
19	1000	796	450	471
20	1570	1570	N/A	N/A
21	1572	1572	1399	1383

Table 1: Results obtained by CP model using first fail strategy

2.4 Others Experiments

During the workflow several experiments have been tried. One notable experiment was performed on the model based on the global constraint *bounded_dpath*. This constraint can be applied to a graph structure, in particular, it enforces the presence of a path from a node s to a node w.
Although this approach is intriguing as it models the problem using a graph structure, its performance is worse than the proposed one, this may be due to the fact that the constraint required a boolean representation of the graph.

3 SAT

The solver Z3 has been used through a python environment to model SAT. The particularity of SAT w.r.t. the other optimization techniques is that it only works with Boolean variables, being a technique to solve propositional logic formulas.

Several encodings have been tried, the following is the one with less spatial complexity in terms of variables and constraints, which also resulted in the best performances.

3.1 Decision variables

This model is based on the following variables.

- First, the matrix of assignments $Asg \in \{0,1\}^{m\times(n+1)}$, where $Asg_{i,j} = 1$ if courier i carries item j, 0 otherwise. The last column represent the depot, if a courier travels then $Asg_{i,n+1} = 1$ meaning that it leaves the depot.
- Then, there's the matrix $Couples \in \{0,1\}^{(n+1)\times(n+1)}$, where $Couples_{i,j} = 1$ means that a courier travels from distribution center i to distribution center j. Last row and last column identify movement from depot and to depot respectively.
- The third structure is a vector of ordering $Ordering \in \{0,1\}^{(n+2)\times(n+2)}$, this matrix is used to define an order among the items, avoiding sub loops in the *Couples* paths. The idea of this structure is been inspired by the unary encoding analyzed in [3]
- Finally, the array d, l and the *maximum* as already introduced in Section 1, but encoded in binary form to be coherent with the SAT formulation.

3.2 Constraints

The formulation of the constraints is based on propositional logic, the model is formalized as follows:

- The model must ensure that each item must be delivered once, let's denote with c_i the i^{th} column of the matrix Asg:

$$\bigwedge_{i=1}^{n} exactly\_one(c_i)$$

- If a courier carries an item, it must move from the origin:

$$\bigwedge_{k=1}^{m} (\vee_{i=1}^{n} Asg_{k,i}) \rightarrow Asg_{k,n+1}$$

- Avoid travels with starting point equal to ending point

$$\bigwedge_{i=1}^{n+1} \neg(Couples_{i,i})$$

- This constraints guarantee that the *Orderings* columns represent items and that those are not repeated.

$$\bigwedge_{i=1}^{n+2} exactly\_one(Ordering_i)$$

  Let's denote with c_i the i^{th} column of the matrix *Orderings*:

$$\bigwedge_{i=1}^{n+2} exactly\_one(C_i)$$

- The model also need to guarantee that each movement is assigned to the same courier.

$$(Couples_{i,j} = 1) \Rightarrow (\exists k \in [1,..,m] \quad s.t. \quad (Asg_{k,i} \wedge Asg_{k,j}) = 1)$$

$$\forall i, j \in [1, ..., n+1]$$

- The following is the constraint needed to ensure that the path formed in the *Couples* matrix are correct.

$$\bigwedge_{i=1}^{n+1} \bigwedge_{j=1}^{n+1} Couples_{i,j} \rightarrow ge(o_i, o_j)$$

  where $ge(x, y)$ is a logic formulation of the greater equal between binary numbers, it returns a set of constraints which are satisfiable only whether $x \geq y$ and o_i identifies the i^{th} column of the matrix *Ordering*

- This constraints ensures coherence between the *Couples* and *Asg* matrices, for what concern the movement from the depot.

$$\sum_{i=1}^{m} Asg_{i,n+1} = \sum_{j=1}^{n+1} Couples_{n+1,j} = \sum_{k=1}^{n+1} Couples_{k,n+1}$$

- This constraint ensures that each courier respects their maximum size:

$$\bigwedge_{i=1}^{m} gt(courier\_size_k, courier\_load_k)$$

- The others constraints were mainly introduced to compute mathematical operations such as the sum of binary represented integers, the maximum in a list of binary represented integers and the *ge* between numbers.

3.3 Optimization Search Strategies

The z3 SAT solver does not provide any optimization step, so in order to derive optimal solution, the optimization procedure must be ensured by constraining iteratively the objective function. The provided approaches are the following:

- Linear search: it has been used a different approach from the classical one. In particular, this approach starts from the upper bound defined in Section 1 and iteratively searches for a solution. When a solution is found, the upper bound is updated to be one less than it.
- Binary search: the implementation of the binary search is based on the same principle of the linear search. In particular, it starts from the upper bound and lower bound defined in Section 1 and then the search space is iteratively reduced following this formulation:

  $$\frac{upper\_bound - lower\_bound}{2}$$

  When a satisfiable solution is found, the upper bound is updated; when an unsatisfiable solution is found the lower bound is updated. The search ends when $upper\_bound = lower\_bound$.

3.4 Validation

Experimantal design

The SAT model has been tested using z3 solver, while all the models were built using the python library provided by z3. The sat solver does not include the optimization step though, so finding the optimal solution required to perform a search over the solutions provided:

- The first approach has been linear search, were the search begins with an upper bound on the max, and each solution provided is reintroduced into the solver as a constraint until the unsatisifiablity is reached.
- Then the more efficient binary search is been implemented, in order to reduce as fast as possible the search space.

Experimental results

In the following table has been reported the difference between the two types of search strategies. For the instances which are not reported, the solution is unknown.

ID	Linear + SB	Linear w/out SB	Binary + SB	Binary w/out 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	225	241	183	191
8	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**

Table 2: Results obtained by SAT model using Linear and Binary search

3.5 Others Experiments

An other SAT approach has been used to model the structure of the problem. The approach is based on using two vectors, namely $start, end \in 0, 1^{mx(n+1)}$ where for each courier k, the position i of $start$ and end identify the i^{th} edge, where the starting node is $start_i$ and the ending node is end_i.
The setting of the constraints is based on creating a path for each courier, modelling the whole structure of the problem has a graph which is represented by the $start$ and end vectors.

4 SMT

The SMT (satisifiablity modulo theory) is an approach to solve optimization problems based on a collection of solvers for different theories [1]. The models constructed for solving the couriers problem exploited the linear integer theory and the array vector theory. Those tools were extremely useful in order to reduce the spatial complexity of the model and to exploit more high level structures and constraints when building the models.

4.1 Decision variables

The structure of this model is based on the same structure of the CP approach, the main structures used are:

- An ArrayVector Asg, represented as a matrix, for the first dimension the matrix has size m, while the size of the second dimension is unknown a priori. Based on the structure's usage, it can be inferred that the second dimension has a size of $n + 1$.
  This matrix represent the path of each courier, precisely $Asg_{i,j}$ means that the i^{th} courier start from j and arrive to $Asg_{i,j}$.
  The domain of Asg is $[-1, n + 1]$:

  - when the value of $Asg_{i,j}$ is -1 means that the i^th courier does not deliver item j.
  - when the value of $Asg_{i,j}$ is $n+1$ means that the i^th courier starts from position j to the origin.

- $final\_distance \in \mathbb{N}^m$ where each element at position i represents the maximum distance traveled by each courier.
- The array of *distances*, *couriers_load* and the *maximum* as already introduced in Section 1.

4.2 Constraints

The main constraints of the model are:

- Ensure that each row of Asg is a sub circuit

$$\bigwedge_{i=1}^{m}\bigwedge_{j=1}^{n+1}((Asg_{i,j} \neq -1) \wedge (Asg_{ij} \neq n)) \rightarrow is\_circuit(n, Asgi, j, 0, Asgi, n+1)$$

where:

$$is\_circuit(i,s,r,v,it) = \begin{cases} False & r = it \\ v_i \neq it \quad \wedge \quad (v_i = s \quad \vee & \\ is\_circuit(v,s,r+1,v,it)) & otherwise \end{cases}$$

- This constraint completes the previous constraint because ensures the creation of a subcircuit without sub loops.

$$\bigwedge_{i=1}^{m}\bigwedge_{j=1}^{n+1}(Asg_{i,j} \neq -1) \rightarrow (Asg_{i,v} \neq -1) \quad where \quad v = Asg_{i,j}$$

- This constraint ensures that each item can be delivered by at most one courier:

$$\bigwedge_{i=1}^{m} at\_most\_one(Asg_i = j) \quad \forall j \in [1, ..., n+1]$$

- This constraint ensures that the i^th courier cannot start and end in the same position:

$$\bigwedge_{j=1}^{n+1}(Asg_{j,i} \neq i) \quad \forall i \in [1, ..., m]$$

- This constrain ensures that either the courier must arrive to the origin or the courier does not depart:

$$\bigwedge_{i=1}^{m} (Asg_{i,j} = n \vee sum(Asg_{i,k}) = -(n+1)) \quad \forall j, k \in [1, ..., n+1]$$

- This constraint "computes" the sum of the loads of items delivered for each courier:

$$\bigwedge_{i=1}^{m} couriers\_load_i = sum(if(Asg_{i,j} \neq -1, item\_size_j, 0)) \quad \forall j \in [1, ..., n{+}1]$$

- This constraint ensures that each courier respects its max load:

$$\bigwedge_{i=1}^{m} couriers\_load_i \leq courier\_size_i$$

- The distances constraint has the same principle of the constraint to "compute" the sum of the loads for each courier.
  Finally the maximum has been added to the constraints set.

4.3 Validation

Experimental design

The creation of SMT model has been implemented through the Z3 library as a base environment to build the models. Then those have been also converted in smtlib format in order to test different solvers. Differently from z3, smtlib format does not provide the optimization step, so it has been created by means of liner and binary search. The SMT has been tested using two solvers:

- Z3
- CVC4

Experimental results

The following results are based on the using of binary search. For the instances which are not reported, the solution is unknown.

ID	Z3 + SB	Z3 w/out SB	Cvc4 + SB	Cvc4 w/out SB	Z3OPT + SB	Z3OPT w/out SB
1	**14**	**14**	**14**	**14**	**14**	**14**
2	N/A	**226**	**226**	N/A	**226**	**226**
3	**12**	**12**	**12**	**12**	**12**	**12**
4	**220**	**220**	**220**	**220**	**220**	**220**
5	**206**	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**	**322**	**322**
7	N/A	N/A	N/A	N/A	**167**	221
8	N/A	**186**	N/A	**186**	**186**	**186**
9	N/A	**436**	N/A	N/A	**436**	437
10	N/A	**244**	N/A	N/A	**244**	252

Table 3: Results obtained by SMT model using binary search and Z3 Optimizer

4.4 Others Experiments

For SMT has been tried the approach using for the SAT explained in the section 3.5 to make a comparison between two model which have the same structure but one use the propositional logic and the other one the Int and the Array theory.
Based on the observation of these two models, it can be concluded that constructing of SMT allows to have a reduction in spacial complexity due to the fact that the numbers in SAT are encoded as binary. An other observation is that the construction of SMT's constraints is easier than SAT given by the higher level structure.

5 MIP

The Mixed Integer Programming model has been used through the python library PuLP, an LP modeler which creates MIP or LP files and can call different solvers to optimize models. Also for MIP several encodings have been tried, below is reported the best one in terms of runtime efficiency and spatial complexity.

5.1 Decision variables

This model high level structure is very similar to the one used for SAT. The main idea is to use 3 structures:

- The first one is a matrix of assignment $Asg \in \{0,1\}^{m\times(n+1)}$: in particular $Asg_{ij} = 1$ if courier i carries item j, 0 otherwise. Moreover $Asg_{i,n+1} = 1$ if courier i carries at least one item, 0 otherwise, namely if a courier start a travel.
- The second matrix is couples, $Couples \in \{0,1\}^{(n+1)\times(n+1)}$: in particular $Couples_{i,j} = 1$ if a courier in his path goes from item i to item j. Last

row and last column identify respectively the first items which are carried by the couriers and the last column what are the last items carried before coming back to the depot.

- The third structure is a vector of ordering *ordering* $\in \mathbb{N}^{(n+2)}$ which has domain [1,...,n+2]. How this vector is used to define on order among the items, as explained further on.

5.2 Constraints

Let's start from the main constraints of the model:

- The first constraint imposes that all the items are carried, thus we impose an exactly one on all the columns of the assignment matrix except for the last (which indicates if a courier start or not).
  Let's denote with C_i the i^{th} column of the matrix Asg:

$$\bigwedge_{i=1}^{n}(\sum_{j=1}^{m} C_{ji}) = 1$$

- The second constraint concerns always the Asg matrix, in particular if a courier i carries at least one item then $Asg_{i,n+1} = 1$

$$(\sum_{i=1}^{n} Asg_{k,i} >= 1) \Rightarrow (Asg_{k,n+1} = 1) \quad \forall k \in [1, ..., m]$$

- The following constraints ensures that if $Couples_{i,j} = 1$ then both the item i and the item j must be carried by the same couriers:

$$(Couples_{i,j} = 1) \Rightarrow (\exists k \in [1, .., m] \quad s.t. \quad (Asg_{k,i} \wedge Asg_{k,j}) = 1)$$

$$\forall i, j \in [1, ..., n+1]$$

- Let's now impose that the ordering vector contains exactly once all the value [1,n+1], to do this we use an auxiliary matrix to impose the uniqueness of the values $Ord\_matrix \in \mathbb{N}^{(n+2)\times(n+2)}$, this matrix has domain [0,1] and exactly one value equal to 1 on all the rows and all the columns

$$\bigwedge_{i=1}^{n+2} ordering_i = (\sum_{j=1}^{n+2} Ord\_matrix_{j,i} * j)$$

  Let's remind that the product has been linearized.

- this constraints imposes the ordering between items asserting that if $Couples_{i,j} = 1$ then $ordering_i$ is greater than $ordering_j$

$$(Couples_{i,j} = 1) \Rightarrow (ordering_i \geq ordering_j) \forall i, j \in [1, .., n+1]$$

- finally we impose the constraints on the weights, in particular each courier must respect its courier size:

$$\bigwedge_{i=1}^{m}((\sum_{j=1}^{n} asg_{i,j} * item\_size_j) \leq courier\_size_i)$$

5.3 Validation

Experimental design
The model has been tested using two different solvers: PULP_CBC_CMD (based on CBC solver [2]) and GLPK. The main reasons for choosing these solvers was the known capability of solving large-scale linear programming problems

Experimental results
Below are reported the results with and without symmetry breaking for the two solvers. For the instances which are not reported, the solution is unknown.

ID	CBC + SB	CBC w/out SB	GLPK + SB	GLPK w/out SB
1	**14**	**14**	**14**	**14**
2	N/A	**226**	N/A	N/A
3	**12**	**12**	**12**	**12**
4	**220**	**220**	N/A	**220**
5	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	N/A
7	N/A	258	N/A	N/A
8	**186**	200	N/A	N/A
9	N/A	**436**	N/A	N/A
10	N/A	246	N/A	N/A

Table 4: Results obtained by MIP model

5.4 Others Experiments

Beyond the model presented, other experiments have been done, firstly testing different python interface for LP problem solving such as MIP-python library and PuLP. Secondly, different models have been tested, among them one of major interest has been through the usage of multiple matrices for each courier, where each one stored the items carried by the associated courier. This model w.r.t. the one presented above presents an higher number of variables leading to worse results.

6 Conclusions

This project has been a really complete journey through the optimization world. Several approaches and libraries has been used in order to provide satisfactory results. Some of the main issues which are common in combinatorial decision making were handled with experiments and ideas. The use of such different approaches to the problem has made possible to understand properly the main advantages and drawbacks of all them.

Some future work could be to extend the project including some variation of the problem, exploiting the modularity of the interface proposed to run all the different models and trying to use more memory efficient languages in order to construct the models, so that the spatial complexity of the model itself is kept as small as possible.

References

[1] Clark Barrett and Cesare Tinelli. *Satisfiability modulo theories.* Springer, 2018.

[2] Robin Lougee-Heimer John Forrest. *CBC User Guide.* INFORMS, 2014.

[3] Neng-Fa Zhou. *In Pursuit of an Efficient SAT Encoding for the Hamiltonian Cycle Problem.* CUNY Brooklyn College Graduate Center, 2019.