Combinatorial Decision Making and Optimization Project Work: Multiple Couriers Planning

Benedetti Enrico
enrico.benedetti5@studio.unibo.it

Levita Luca
luca.levita@studio.unibo.it

Fantazzini Stefano
stefano.fantazzini@studio.unibo.it

Hartsuiker Jens Matthias
jens.hartsuiker@studio.unibo.it

Problem description

A set of n goods have to be delivered by a fleet of m vehicles to different customer locations. Each courier's vehicle has a maximum capacity. Also, each item has a size and needs to be transported to the right location. It is required to plan a tour for each courier, i.e. define the sequence of locations visited to deliver the items. A courier's item load cannot exceed its maximum capacity. All couriers begin and end their route at a single depot, situated at the origin o. The objective is to minimize the total tour distance.

Instance format

For each instance there is a text file containing on each line different parameters of the problem.

- m: the number of couriers available;
- n: the number of customers to visit;
- the m capacities of each courier's vehicle;
- the n weights/sizes/quantities of goods for each customers' order;
- the x and y euclidean coordinates for the customers' and the depot/base positions.

Follows an example for the test instance given in the project pdf (the files made for Minizinc models are recycled).

```
m = 3;
n = 7;
capacities = [15, 10, 7];
weights = [3, 2, 6, 8, 5, 4, 4];
Xs = [1, 2, 2, 4, 5, 5, 6, 3];
Ys = [3, 1, 5, 0, 2, 5, 4, 3];
```

SAT Model

Summary

Our SAT model is run by the $z3$ solver (using its Python wrapper). By declaring variables, boolean propositions, formulas, and running it it will hopefully output an assignment of the variables making the whole formula true, or UNSAT if there is none. In our case it mostly didn't get to the end of the search.

Our problem model has $n+m$ locations to visit. From 1 to n they represent customer locations, while points $n+1, \dots, n+m$ are additional 'dummy' depot locations. It should be noted that the depots share the same coordinates.

A solution for the problem is a permutation of those $n+m$ points from which routes that satisfy the problem constraints can be derived.

To give a broad explanation, the decision variables e model the fact that an edge between two points in the graph is used in the solution. Then, we have inserted other decision variables for the visit order (those propositions determine the order of visits for the couriers). Encoding an ordering function is needed to enforce the subtour elimination constraints. Finally, we added variables that kept track of which goods were delivered by which courier.

The objective function is the total distance travelled by the couriers, and it has to be minimized.

Numerical values such as distances, weights, capacities are SAT encoded in binary format, and arithmetic operators like sum and $\leq$ have also been SAT encoded.

In order to improve the quality of solution faster, we used a binary search approach.

Parameters

$N = \{1, \dots, n\}$: the set of customer locations.

$K = \{1, \dots, m\}$: the set of all couriers.

$B = \{n+1, \dots, n+m\}$: the set of depot locations (they are the same location).

$A = N \cup B$: the set of all locations.

D: the distance matrix. In the implementation, index 0 is the depot, and indexes in B also point to the depot index. $D_{i,j} = \text{manhattan}(i, j)$. The manhattan distance, also known as taxi distance or 1-norm, is computed using the sum of differences in the absolute values of the coordinates: $\text{manhattan}(P_1, P_2) = |x_1 - x_2| + |y_1 - y_2|$.

weight_i: the weight of package demanded by customer i, $i \in N$.

capacity_k: the capacity of courier number k.

Routing variables and constraints

$$e_{ij} = \begin{cases} 1 & \text{if the edge going from } i \text{ to } j \text{ is used} \\ 0 & \text{otherwise} \end{cases} \qquad \forall (i,j) \in A$$

Each decision variable e_{ij} encodes the possibility of going from position i to position j.

$$v_{ij} = \begin{cases} 1 & \text{if place } i \text{ is visited at timestamp } j \\ 0 & \text{otherwise} \end{cases} \qquad \forall (i,j) \in A$$

v_{ij} decision variables encode the fact that customer or depot i is the jth place visited. Also thinking about j in terms of timestamps can help.

Those two sets of propositions in a way represent the same information: the tour of couriers. They are meant to be used together in order to put in place subtour elimination constraints, which in the context of the problem are subroutes that loop without including the depot. They have been added following the unary encoding method from [**1**].

Note: the exactly-k constraint uses the sequential encoding, which was used in the lab sessions for the course.

$$\text{exactly-1}(\{e_{ij} \mid j \in A\}),\ \forall i \in A \tag{1}$$

$$\text{exactly-1}(\{e_{ji} \mid j \in A\}),\ \forall i \in A \tag{2}$$

These constraints (1) and (2) make it so every node is connected to exactly one other node in the solution.

$$\text{exactly-1}(\{v_{ij} \mid j \in A\}),\ \forall i \in A \tag{3}$$

$$\text{exactly-1}(\{v_{ji} \mid j \in A\}),\ \forall i \in A \tag{4}$$

Constraints (3) and (4) ensure that every node visited once, and at each timestamp only one position is visited. Some of them are implied by the channeling constraints added after them.

$$v_{n+1,1} = 1 \tag{5}$$

The depot associated with the route of the first courier is visited first (First of all, they leave that depot).

$$e_{n+1,i} \implies v_{i,2},\ \forall i \in A \setminus \{n+1\} \tag{6}$$

If there is an edge from the first depot to position i, then i is visited second.

$$e_{i,n+1} \implies v_{i,n+m},\ \forall i \in A \setminus \{n+1\} \tag{7}$$

If there is an edge from i to the first depot then i's position is last.

$$e_{i,j} \wedge v_{i,p} \implies v_{j,p+1},\ \forall p \in \{3, \ldots, n+m-1\},\ \forall (i,j) \in A \setminus \{n+1\}, i \neq j \tag{8}$$

These constraints ensure that if edge (i, j) is in the Hamiltonian cycle, and vertex i's position is p, then vertex j's position is $p + 1$.

To model the fact that we are searching for the shortest route, a measure for the total distance is needed. A value for the total distance is obtained by means of partial sums between points with an edge between them. To do that in propositional logic, we need to compute all possible sums across the $A \times A$ edge matrix and use the logical implication operator to say that, if there is an edge between two points, then the distance between them needs to be added to the partial sum; If not, it should stay the same. The total distance will therefore be contained in the last partial sum.

Because the problem requires to represent many numbers (and also capacities and weights have to be modelled), we implemented a binary logarithmic encoding for integers. An integer number is encoded as a bit vector in a big-endian fashion (the most significant bit is the first in the sequence). The sum operator between bit vectors of length l is implemented following and extending the course slides.

In general, to encode the fact that $a + b = d$ with carry c, we have the conjunction

$$\phi \wedge \bigwedge_{i=1}^{|a|} [\neg]a_i \wedge \bigwedge_{i=1}^{|b|} [\neg]b_i$$

where the sum equality constraints ϕ are:

$$a_i \iff b_i \iff c_i \iff d_i \text{ for } i = 1, \ldots, l;$$

$$c_{i-1} \iff ((a_i \wedge b_i) \vee (a_i \wedge c_i) \vee (b_i \wedge c_i)) \text{ for } i = 1, \ldots, l;$$

$$\neg c_0 \wedge \neg c_l.$$

Following this, to be more concise, $\text{add}(a, b, d)$ will be a shorthand for this sum implementation. Next up are the logical formulas for encoding the distance notion.

$$\bigwedge_{b=1}^{|\text{distanceSum}_1|} \neg\text{distanceSum}_{1_b} \tag{9}$$

Negate every bit of the first partial sum to set it to zero.

$$e_{i,j} \implies \text{add}(D_{ij}, \text{distanceSum}_{i+j-1}, \text{distanceSum}_{i+j}),\ \forall (i,j) \in A \tag{10}$$

The binary encoded distance matrix values are summed to the previous partial sum in case of the edge being present.

$$\neg e_{i,j} \implies (\text{distanceSum}_{i+j-1} \iff \text{distanceSum}_{i+j}),\ \forall (i,j) \in A \tag{11}$$

Otherwise, the sum remains the same. The total distance will then be encoded by the last bit vector, $\text{distanceSum}_{2(n+m)}$.

Capacity variables and constraints

In handling the capacity requirements, a similar method is applied. The sum of the weights transported by each courier must be less or equal than their capacity.

Along with the sum operator, a SAT implementation of the less or equal operator has been coded, as well as the decision variables that encode which packages are delivered by which courier.

$$\text{deliveredBy}_{ki} = \begin{cases} 1 & \text{if package } i \text{ is delivered by courier } k \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in N, \forall k \in K$$

$$\text{exactly-1}(\{\text{deliveredBy}_{ki} \mid k \in K\}),\ \forall i \in N \tag{12}$$

Ensures that each package is delivered by exactly one courier.

$$\text{deliveredBy}_{kj} \wedge e_{j,i} \implies \text{deliveredBy}_{ki},\ \forall k \in K,\ \forall (i,j) \in N, i \neq j \tag{13}$$

If package j is delivered by courier k and there is an edge between j and i, then also i is delivered by k.

$$e_{n+k,i} \implies \text{deliveredBy}_{ki},\ \forall k \in K,\ \forall i \in N \tag{14}$$

Exiting its depot ($n + k$ is the depot of courier k) marks the start of which edges are covered by courier k.

Using the same method of partial sums, we encode the total weight carried by each courier.

$$\bigwedge_{b=1}^{|\text{weightSumk}_1|} \neg\text{weightSumk}_{1_b},\ \forall k \in K \tag{15}$$

Negate every bit of the first partial sum to set it to zero.

$$\text{deliveredBy}_{ki} \implies \text{add}(\text{weight}_i, \text{weightSumk}_i, \text{weightSumk}_{i+1}),\ \forall i \in N, \forall k \in K \tag{16}$$

If a package is delivered by a courier, add its weight to that courier's load.

$$\neg\text{deliveredBy}_{ki} \implies (\text{weightSumk}_i \iff \text{weightSumk}_{i+1}),\ \forall i \in N, \forall k \in K \tag{17}$$

Otherwise, the sum remains the same.

For each courier, the load is encoded in the sequence of propositions weightSumk_{n+1}.

The last thing to add to the model is the constraint that ensures the loads of vehicles to not be exceeded. Because of this, we implemented a propositional formula that is true when a sequence of boolean values is less or equal than another (it is exactly like a precedence constraint). We started with the truth table of $a \leq b$, and constructed a formula that can be applied to the whole sequences.

$$\begin{aligned} a_i &\leq b_i \\ \bigwedge((a_i \wedge b_i) &\implies a_{i+1} \leq b_{i+1}) \\ \bigwedge((a_i \wedge b_i \wedge a_{i+1} \wedge b_{i+1}) &\implies a_{i+1} \leq b_{i+1}) \\ &\bigwedge \ldots,\ \forall i \in |a| \end{aligned}$$

Starting from the most significant bit of the bit vectors, the formula holds if it holds for first bit $a_1 \leq b_1$. In case they are equal, then it holds if the following are less or equal, and so on. In the following, lessOrEqual will be a shorthand for the above formulation.

$$\text{lessOrEqual}(\text{weightSumk}_{n+1}, \text{capacity}_k)\ \forall k \in K \tag{18}$$

Finally, we enforce the less or equal constraints on the loads and capacities for the couriers.

The solver tries to find a feasible truth assignment to all propositions in order to get a SAT formula. Since it will most probably not be the optimal one, it needs to start the search again, with the constriant that the total distance must be less or equal than the one previously found. When it returns UNSAT, the previous solution is the optimal one.

We implemented binary search, a more efficient algorithm to converge to the optimal solution. In this case, when it returns UNSAT, it means that the optimal solution is somewhere between the last SAT solution and the UNSAT lower bound+1. In our Python code, every time we get a SAT result, we add a new constraint on the value of the objective function, asking the solver to find a solution in a restricted interval (about a half of the space visited in the last search), creating a kind of checkpoint with a push before adding this constraint. If the result is again 'SAT', we iterate the process, otherwise we undo to the last checkpoint (with a pop) and add another checkpoint with the search space limited between the last SAT solution and a point after the last UNSAT bound (that could even be that bound+1).

Furthermore, we even tried to add some of the z3's Tactics [**2**], like concatenation of formula's simplifications and bounds' normalization, but no combination of them produced good result in the given amount of time, so we decided to remove them in order to keep the code cleaner and simpler.

Results

Solving the capacitated heterogeneous vehicle routing problem to optimality with a SAT encoding proved to be impossible for the majority of proposed instances. Given that it takes a considerable amount of time to even add all the constraints to the solver, unfortunately we can provide no results within 300s of computing time, apart from two instances (01 and 11).

Our results are summarized in the following table:

Instance	Obj. function value
inst01	3916
inst02..inst10	timeout
inst11	4870

Table 1: SAT results

As for the performance results, our model could not produce any optimal solution to any of the more difficult instances, probably because of the excessive complexity and quantity of our constraints. As the number of destination points increases (also considering an increase in the number of couriers adds more dummy depots), the number of constriants does too, exponentially.

That, and the fact that we didn't implement the fastest state-of-the-art helper encodings (for example, sequence and cardinality) really limits the size of problems our SAT encoding can solve. It could improve by researching more in that direction, but since SAT is not well-suited for efficiently tackling this problem, we stayed on the more intuitive modelling and encoding to focus more into the other frameworks.

References

[1] In Pursuit of an Efficient SAT Encoding for the Hamiltonian Cycle Problem, the 26rd International Conference on Principles and Practice of Constraint Programming, 585602, 2020 (N.F. Zhou)

[2] University of Tel Aviv, School of Computer Science, "Z3 Strategies", http://www.cs.tau.ac.il/ msagiv/courses/asv/z3py/strategies-examples.htm