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];
```

SMT Model

Summary

Our SMT model is run by the $z3$ solver (using both its Python wrapper and the console command for the solver indipendent version of the code). The solver indepent model is automatically generated in the SMTLIB2 format from the assertions of the $z3$ model.

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.

We used a lightly simplified version of one of our CLP models as our definitive SMT model. It has $n + 2 * m$ locations to visit. From 1 to n they represent customer locations, while points $f_k = n + k$ represent the first visit of vehicle k, and the points $l_k = n + m + k$ the last visit of the same vehicle. It should be noted that all the f_k and l_k points share the same coordinates: the origin coordinates.

A solution for the problem is a permutation of those $n + 2 * m$ points from which routes that satisfy the problem constraints can be derived.

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

Numerical values such as distances, weights, capacities, and operators like sum and $\leq$ as well, are encoded using the number theory format that SMT, unlike SAT, understands.

For the optimization part, we used a binary search algorithm, in order to improve the quality of solutions faster. We also tried the 'Optimize' $z3$'s solver, but this last required way too much time for the optimized solution and didn't return any intermediate solutions.

NB: In order to use this optimization step, it is necessary to use the Python wrapper code. Using the command line to run the solver independent version will not execute the Python encoded binary search, but at least it returns a solution that is not garanteed to be the optimal one.

Parameters

$V = \{1, \ldots, n + 2 * m\}$: the set of customer locations and dummy origins.
$F_{start} = n$: the dummy leaving node of the first courier.
$F_{end} = n$: the dummy leaving node of the last courier.
$L_{start} = n$: the dummy arriving node of the first courier.
$L_{end} = n$: the dummy arriving node of the last courier.
$F = \{F_{start}, \ldots, F_{end}\}$: the set of leaving dummy origins.
$L = \{L_{start}, \ldots, L_{end}\}$: the set of arriving dummy origins.
Distance Matrix: the distance matrix. In the implementation, index 0 is the depot. $D_{i,j} = \text{manhattan}(i, j)$.

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: $\text{manhattan}(P_1, P_2) = |x_1 - x_2| + |y_1 - y_2|$.
$K = \{1, \ldots, m\}$: the set of couriers.
weight_i: the weight of package demanded by customer i, $i < n$.
capacity_k: the capacity of courier number k.

Decision Variables

s : the successors array.
p : the predecessors array.
v : the vehicles array.

Routing variables and constraints

The main two array variables for this problem are p (predecessors array) and s (successors array). They are two different formulations. Channelling them should help us propagate the constraints better. Each position of p and s is contrained to be in $\{0, \ldots, V - 1\}$ and they must be all different from each other. The 'extra positions' of the couriers' trip ($p[n + 0]$ to $p[n + m]$) must be equal to $n + m + k$ in order to respect our formulation. The array v's length is equal to

V and each position represent which courier delivered the correspondant package, including the dummy packages.

The meaning and of those three arrays is encoded into the following constraints, along with the channeling:

$$s[p[i]] = i, \ \forall i \in V \setminus F \tag{1}$$

The successor of the predecessor of package i is, of course, i itself.

$$v[i] = v[p[i]], \ \forall i \in V \setminus F \tag{2}$$

The package and its predecessor are delivered by the same courier, except when considering the leaving depots.

$$p[s[i]] = i, \ \forall i \in V \setminus L \tag{3}$$

The predecessor of the successor of package i is i itself.

$$v[i] = v[s[i]], \ \forall i \in V \setminus L \tag{4}$$

The package and its successor are delivered by the same courier, except when considering the arriving depots.

$$v[n+k] = v[n+m+k] = k, \ \forall k \in m \tag{5}$$

Each courier leaves and arrives at the predetermined dummy depot nodes.

Subtour elimination

In one of our CLP models, we used the 'Circuit' constraint that automatically avoids subtours and creates a one way circuit in our s array. We coded a SMT encoded constraint that does the same job as Minizinc's 'Circuit'. With a working array variable order, we make sure that *order* registers a valid visit order (10) for the locations in our problem. It starts with the first courier leaving the depot (6) and contain all positions in V(7) exactly once.

$$\text{order}[0] = F_{start} \tag{6}$$

$$\text{order}[i] < |V| \land \text{order}[i] \geq 0, \ \forall i \in V \tag{7}$$

$$s[i] \neq i, \ \forall i \in V \tag{8}$$

$$\text{order}[i] = |V| - 1 \implies s[i] = n, \ \forall i \in V \tag{9}$$

$$(s[i] = j \land \text{order}[i] \neq |V| - 1) \implies \text{order}[j] = \text{order}[i] + 1, \ \forall i, j \in V \tag{10}$$

Now s is contrained to be a 'Circuit'.

Bin packing and Objective function

The next constraint is the 'bin packing' constraint. With this, we ensure that each courier delivers a number of packages such that the total weight doesn't exceed the capacity of the courier itself.

$$\text{load}_k = \sum_{\substack{i=1,\ldots,n \\ \text{if } v[i]=k}} \text{weight}_i, \ \forall k \in K \tag{11}$$

The logic behind this constraint is: if the courier k delivers the package i ($v[i] = k$) then sum the weight of the package i, 0 otherwise for each i in n.

$$\text{load}_k \leq \text{capacities}_k,\ \forall k \in K \tag{12}$$

Here we enforce the maximum capacity constraint.

To model the fact that we are searching for the shortest route, a measure for the total distance is needed. Unlike SAT, this time the sum is not modelled by means of partial sums between points, but it is computed directly: To get the total distance of the routes: for each position except the starting depots (F), we sum the distance between it and its predecessor.

$$\text{totDistance} = \sum_{\substack{i \in V \setminus F, \\ j=p[i]}} D_{i,j} \tag{13}$$

Search strategy

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 this kind of problem to optimality with a SMT encoding proved to be nearly impossible for the majority of proposed instances. Given that it takes a bit time to even add all the constraints to the solver, unfortunately we can provide no results within $300s$ of computing time.

Our results are summarized in the following table:

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. The model could probably improve, but since SMT (like SAT, being its 'evolution' and using array and arithmetic theories) is not well-suited for efficiently tackling this problem, we preferred modelling and encoding the problem more or less in the same way we did for CLP, focusing more into CLP and MIP, both more suitable for this kind of problem. Obviously we tried other models, like the basic 'matrix' and 'array' encoding (each courier's path is one row of the matrix, and this must start and end with the origin. In the case of the array formulation, it is the same logic, but working with just one row, where each path is subdivided with the presence of the origin) but, given that they could not return a solution in feasible time, we preferred to take inspiration from one of our best CLP models, that run good on every instances but not perfectly, hoping it will perform a bit better than the very basic models.

Instance	Obj. function value
inst01	timeout
inst02	timeout
inst03	timeout
inst04	timeout
inst05	timeout
inst06	timeout
inst07	timeout
inst08	timeout
inst09	timeout
inst10	timeout
inst11	timeout

Table 1: SMT results

References

[1] Philip Kilby, Paul Shaw, Chapter 23 - ”Vehicle Routing”, Foundations of Artificial Intelligence, Elsevier, Volume 2, 2006, Pages 801-836

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