Finding an optimal delivery plan: CSP model

Prisca Aeby, Alexis Semple

The problem this lab aimed at solving was finding an optimal solution for the pickup and delivery problem, where there is one company owning several vehicles. The company tries to use all its resources in the best way, so as to maximise the revenue of all the available tasks.

We modelled our solution according to the description given in the provided paper, and adapted it to allow the vehicles to not pickup and deliver the tasks sequentially, i.e. to pickup a task and then to deliver it at an arbitrary later point. This also allows vehicles to carry several tasks at once.

1 Description of the algorithms

Our implementation differs from the one described in the paper on several points:

- We call a task either a pickup or a delivery and the move of the vehicle to that location.
- We track the sequence of tasks by keeping time arrays for pickup and delivery for each task in the map separately. These arrays contain an entry for each task ID, and store the time value of the execution of that task.
  - To obtain the order of tasks for a vehicle, we use an iterator over the time arrays, which looks for the minimum time value of all tasks in the vehicle's task set. In order to do this, we keep track of all tasks attributed to any vehicle.
- We transform the `nextTask()` to keep track only of the first pickup of every vehicle.
- Our implementation requires two additional constraints:
  - For a task with ID t_i, $pickupTime(t_i) < deliverTime(t_i)$
  - $\forall t,\ 0 \leq t < 2 \cdot taskSize(v),\ v \in Vehicles,\ load(v,t) < capacity(v)$, where $load(x,y)$ computes the load of vehicle x at time y. The function $taskSize(v)$ returns the size of the set of tasks attributed to vehicle v. There are two time entries for each task in the set, one for pickup and one for delivery.

- We compute the cost of a plan by summing the costs of each individual vehicle. This is done by iterating over the tasks in sequence, starting from the home city, and computing the cost for the total driven distance.

Globally, our SLS algorithm works similarly to the one described in the paper. However, we keep track at all time of the best plan found in the iterations, so that if we end up with a solution inferior to the best plan found overall, we can return the best solution instead. In fact, the randomness of the SLS allows for neighbours with sub-optimal cost to be explored.

Initialisation For the initial plan, the tasks are distributed to all vehicles, in approximately equal quantity. This performs better than giving all tasks to a single vehicle, as was described in the paper. The pickup and delivery for each task is done sequentially, to begin with, in order to not violate our additional constraints on `load` and `time`

Function `chooseNeighbours` is done in two stages, as described in the paper, through `changeVehicle` and `changeTaskOrder`.

In `changeVehicle(v1, v2)` , we give the first task of v1 to v2 and insert its pickup at time $t = 0$ in v2's sequence. We generate a new neighbour for each delivery possibility of the new task, starting from time $t = 1$ until a violation of the load constraint. This distribution of deliveries makes it more likely for such a neighbour to be chosen in the `localChoice` function.

In `changeTaskOrder(t1, t2)` , we exchange any two tasks in the set of a vehicle with each other, where a task is either a pickup or a delivery (i.e. an exchange can happen between two pickups, two deliveries or a delivery and a pickup). The neighbour is only created if our additional time and load constraints aren't violated.

Function `localChoice` chooses among the set of neighbours the one with the best cost.

2 Testing

We ran our agent for different settings in order to evaluate its performance. In every case, the vehicles started alone in their home city. For each setup, the values are given in order of **cost**, **task distribution** and **computation time**

10 tasks :

3 vehicles : 11'001, (0, 6, 4), 4'378ms

4 vehicles : 15'056, (3, 2, 0, 5), 2'714ms

6 vehicles : 16'898, (2, 0, 0, 2, 6, 0), 3'170ms

10 vehicles : 15'294, (0, 0, 5, 3, 2, 0, 0, 0, 0, 0), 3403ms

30 tasks :

3 vehicles : 27'861, (11, 7, 12), 33'060ms

4 vehicles : 36'460, (4, 10, 8, 8), 22'584ms

6 vehicles : 35'056, (8, 0, 4, 6, 7, 5), 18'012ms

10 vehicles : 25'247, (5, 9, 0, 8, 4, 0, 0, 0, 0, 4), 25'457ms

60 tasks :

3 vehicles : 50'825, (16, 17, 17), 149'984ms

4 vehicles : 62'553, (13, 17, 14, 16), 157'636ms

6 vehicles : 58'418, (7, 9, 15, 8, 11, 10), 104'803ms

10 vehicles : 46'020, (11, 0, 10, 8, 13, 9, 0, 0, 9, 0), 109'882ms

From this we can see that the number of vehicles doesn't influence the time complexity of the generation of the plan much. However for a greater number of tasks in the world, the complexity grows faster than linear growth. This is because to exchange the order of n tasks, we use $n!$ operations, whereas to change the vehicle of a task, for n vehicles, we use only n operations.

Having more vehicles does not necessarily mean a lower cost for the agent. In the cases where we have 10 tasks in the world, for example, the 3 vehicle setup gives the bets cost. This can be because in such a situation, the exchange of the order of tasks is more often explored than giving the task to another vehicle. As the task number grows however, the setups with more vehicles (10, for 60 or 30 tasks) yield better results, because a greater geographical reach allows them to minimise distance for the large task set.

In some cases, there are vehicles that don't get any tasks attributed to them, because it is more efficient to use a subset of the set of vehicles, and for example leave the geographically remote ones unused.

3 Conclusion

The centralized agent model allows for a great number of vehicles to operate efficiently in a world and not be in conflict with each other, as they were for the deliberative case. For bigger task sets, the computation of a plan can be cumbersome and the algorithm needs to make sure it doesn't get stuck in local minima and thus lose efficiency. However, this model only works when there is a central organism that decides for all vehicles, and there is no competing between them.