Excercise 4
Implementing a centralized agent

Group №76: Simon Honigmann, Arthur Gassner

November 6, 2018

1 Solution Representation

1.1 Variables

A Solution class was created to better store and manage information for each potential delivery solution. A texttttSolution object consists of a texttttHashMap of texttttArrayLists of texttttTasks where each vehicle in the initialization is a key. Each texttttArrayList contains a list wrapped version of a task, which includes information for whether the task is to be picked up or delivered, in the order in which the vehicle is supposed to perform those actions. The Solution also stores it's total cost. Finally, the texttttSolution has a method, which can convert the simplified list of pick-ups and drop-offs into a complete action list suitable for Plan generation.

1.2 Constraints

Relatively few constraints are applied to the solution representation. Constraints include the following:

1. Plans cannot result in a vehicle carrying more tasks that its capacity allows.
2. Deliveries must always after pick-ups in the generated task lists.
3. A vehicle which picks up a task must also deliver that task. There is no exchange between vehicles, within a plan.
4. All tasks provided upon initialization must be picked up and delivered in the generated solution.
5. A solution must be generated within the time-out specified in the default_settings.xml file.

1.3 Objective function

The function to be optimized is the sum of distances travelled by each vehicle for a given solution. For each generated neighbour solution, this value is compared to the current best solution. Any solution outperforming the previous best is stored as the new best solution.

2 Stochastic optimization

2.1 Initial solution

Several methods for generating an initial solution were explored. First, a naive approach which sequentially distributed tasks between each vehicle, to result in a distributed initial task allocation. Second was a greedy approach, which assigned every single task to the first vehicle in the vehicle list which could carry the task. This resulted in the first vehicle having the majority of tasks assigned to it. It is assumed for all initial solutions, that at least one vehicle can carry the largest task in the specified task list. If a task

is ever larger than the selected vehicle's capacity, it is assigned to a different vehicle until the selected vehicle has a large enough capacity.

Ultimately, our testing showed that the naive approach was the most consistent, and this initialization was used in all future experiments.

2.2 Generating neighbours

First a vehicle that has at least 1 task is randomly selected. Then for every each task carried by the chosen vehicle, for each additional vehicle, and for each possible location of the task pick-up and task delivery in the new vehicle, a neighbour is generated. This method indirectly accounts for switching the order of tasks within a vehicle, as through multiple iterations of neighbour generation a task could change back to its original carrier. With this method, it is possible to reach every possible permutation of task distribution, given enough time. Only neighbours that satisfy the constraints listed above are returned.

2.3 Stochastic optimization algorithm

Every time neighbours are generated, the local minimum is first found and compared to the current best solution. If it outperforms the current best, the current best is replaced by the local minimum. To implement the stochastic local portion of stochastic local search, two probability thresholds are defined: textttP_Upper and textttP_Lower which determine how the next iteration's base solution will be selected. Every iteration, a random number is generated between 0 and 1. If the number is below textttP_Lower, the local minimum solution is returned and used to generate the next set of neighbours. If the number is between textttP_Upper and textttP_Lower, inclusive, the previous solution used to generate neighbours is returned. If the number is above textttP_Upper, a random solution is selected from the set of neighbours and is used to generate future neighbours.

To formalize our approach, we added local-minimum loop detection. If the optimal solution stayed in the same for more than `MAX_REPEAT` times, then a random neighbour is automatically selected. This is to try to avoid getting caught at local-minima, without sacrificing performance early in the search by selecting random neighbours when the new local minima are consistently improving. After adding this, `P_Upper` was set to 1 so that random selection is only used when solution stagnation occurs.

3 Results

For all experiments, the following settings were kept constant. Plan time out was 30 s. Every cost value reported was averaged over 3 trials.

3.1 Experiment 1: Model parameters

In the first experiment we tested the impact of textttP_Lower and `MAX_REPEAT` on the final plan cost.

3.1.1 Setting

Settings for this experiment are specified in `E1_centralized.xml` and `E1_settings_default.xml`. This experiment always used 30 tasks, 4 vehicles, and the England topology. Values of `MAX_REPEAT` between 2 and 20, and textttP_Lower between 0.1 and 1 were tested. The results are included in the following section.

3.1.2 Observations

As is shown in the table, `MAX_REPEAT` and textttP_Lower strongly influence the performance of the solution generation. When textttP_Lower is too small, the search is too likely to hold on to old solutions

Table 1: Comparison of Different `MAX_REPEAT` and `P_Lower` Values

MAX_REPEAT	P_Lower			
	0.1	0.5	0.8	1.0
2	$42,815	$26,079	$19,338	$19,806
10	$24,552	$17,556	$16,087	$15,268
20	$21,051	$17,251	$15,724	$18,158

and doesn't explore enough to find optimal solutions. As texttttP_Lower increases, the solution favours local minima and stops holding onto old solutions. This proves to be advantageous in all cases, though this is likely due to our implementation of loop detection. There also appears to be an optimal number of allowed repetitions during loop checking. Too few and the search wanders in random directions too often. Too many, and the algorithm gets stuck at local minima without exploring enough to find more optimal solutions.

3.2 Experiment 2: Different configurations

3.2.1 Setting

3.2.2 Observations