Singapore Management University
School of Computing and Information Systems
2021/2022 Semester 2
CS606: AI Planning and Decision Making
Assignment 2 (Due 13 March 2022)

Maximum marks = 30.

Submission Instructions

You are to submit two files to eLearn (in the Assignment 2 folder):
(1) a Word or PDF document containing the written answer for Question (a)
(2) python codes (named `evrp.py and alns_main.py`) for Question (b).
We will test your code by the following command line. Please make sure your code can be executed as: `python alns_main.py datafile.xml randomseed`

These files should be zipped and uploaded to the eLearn Assignment folder Assignment 2. To ease handling, please name your file `A2_YourName.zip`. Multiple submissions are permitted up to the due time, but only the last submission will be saved and graded.

Problem Description

In this assignment, we will explore the application of ALNS to solve the E-VRP-NL (i.e. Electric vehicle routing problem with nonlinear charging function), as given in the DIMACS Implementation Challenge for VRP: http://dimacs.rutgers.edu/programs/challenge/vrp/evrp/

Read the following paper for the problem description and their solution approach:

> Montoya et al. 2017: The electric vehicle routing problem with nonlinear charging function. *Transportation Research Part B Methodological*.

Unlike the standard VRP, E-VRP involves both sequencing decisions as well as charging decisions. This paper proposes a metaheuristic approach based on a sequence-first-charge-second strategy. To make sequencing decisions, an iterated local search algorithm was proposed. To make charging decisions, a mathematical programming model that solves the fixed route vehicle charging problem (FRVCP) was proposed, along with a greedy heuristic.

In this assignment, we simplify the E-VRP-NL problem by assuming that every customer can always be served by no one more than one visit to a charging station.

The full data set of instances is given in the file `filtered-montoya-et-al-2017_instances.zip` (which was extracted from the dataset given in the DIMAC page).

Question

(a) (10 marks) Present your ALNS design for E-VRP-NL, i.e. describe the construction heuristic, destroy methods, repair methods, and the weights adjustment strategy.

Answer:

The ALNS-design for the algorithms I choose is implementing the following heuristisc:

The design idea is to search for a better solution at each iteration by destroying a part of the current solution and by reconstructing it in a different way. The principle is hereby to remove and reintroduce vehicles and customers into the solution to find a better possible route and more optimal solution.

Hereby, firstly I am reconstructing a weighted matrix of all the potential tours within the TSP-tour, whereas I am setting all the potential subtours $r_{v_i,v_{i+n_r}}$, which exceed the maximum travel time $t_{r_{v_i},v_{i+n_r}}$ of the vehicle to infinite, such that they are never chosen. Furthermore, the splitting procedure builds a directed acyclic graph, whereas throughout the directed acyclic graph only the shortest path routes are chosen through topological sorting.

At each node, I am using the closest neighbour heuristic, such that all the closest neighbours are being considered. Hereby, for split routes I am using a extreme greedy-heuristic, such that only the best greedy combinations are being considered.

Moreover, at each node, the heuristic shall do two feasibility checks, whereas the maximum time needed, as well as the battery level needs to checked.

If the car can't reach the next node and successfully go back to the respective depot, it shall first go back to the depot. Furthermore, the heuristic checks at each move, whether it can go to the next node and go followingly still go home, thus ensuring that at least it can go to the next node and still be able to go back to the depot.

If the time time check is successful, but the battery check fails, the heuristic shall reroute the car to the closest charging station between the current node and the next target node and recharge back to full.

I am implementing two destroy-methods: the first destroy method is deleting the worst edges, which are the furthest away from the destruction-parameter. The second destroy method is removing edges randomly. In regards to the repair function, I am implementing a greedy-repair function, which implements the cheapest insertion position given all the unserved requests, whereas it considers the two nearest nodes greedily.

After each iteration, the weights associated with the applied operators are adjusted. This is finally repeated until the termination criteria of 1000-ALNS iterations are met. I am using HillClimbing() from the ALNS-package and I have set the omega-values to be slowly decreasing starting from 0.4 and assigning the same weights to values to solutions, which are accepted and rejected.

(b) (20 marks) Implement your ALNS algorithm presented in (b) in Python based on the code template provided (see Annex A below for details).

Marking Criteria

Part (a):

(3 marks) construction heuristic,
(3 marks) destroy methods,
(3 marks) repair methods, and
(1 mark) weights adjustment strategy.

Part (b):

Your code will be graded based on the quality of solutions and run time efficiency, which are measured by running against selected test instances in `filtered-montoya-et-al-2017_instances.zip`. The details of the input and expected output format are given in **Annex B** below.

For grading purposes, we will only be running your code on instances **with 20 or less customers**, and that your code will be executed with **1000 ALNS iterations**. For each instance, we expect improvement on the objective value of the solution over the initial solution obtained by your construction heuristic; and the run time is expected to be less than 2 hours.

More precisely, you will get 5 marks if your code produces feasible solutions, and 0 mark if your code failed to produce feasible solutions. You will get an additional 5 marks if the objective value of your solution (after 1000 ALNS iterations) is strictly better than that of your initial solution. The remaining 10 marks will be awarded on a curve comparing your results with the results obtained by your classmates on both the objective values obtained and run time.

References

Some helpful code repositories in GitHub:

1. https://github.com/N-Wouda/ALNS
   This is the repository from which our code template was generated. It also contains implementation for TSP and other examples.
2. ALNS for standard VRP
   Explanation: https://programmer.group/6131160e75dbe.html
   Code: https://github.com/PariseC/Algorithms_for_solving_VRP

Annex A: Explanation of Code Template

2 skeleton python files are provided in the `code_skeleton` folder, along with the `alns` package given in https://github.com/N-Wouda/ALNS[1]. The following is a brief explanation of the two skeleton codes. Refer detailed description in the comments in the python files.

1. alns_main_skeleton.py – main program. Change the file name to `alns_main.py`

Functions to generate outputs (do not modify):

- save_output – save the solution
- create_graph – for visualization
- draw_evrp – for visualization
- generate_output – generate the output file

Functions to be modified:

- destroy_1 – implement your destroy functions with same parameters and return
- repair_1 – implement your repair functions with same parameters and return

Lines to be modified:

- save_output('YourName', evrp, 'initial') – change 'YourName' to your full name
- alns.add_destroy_operator(destroy_1) – change destroy_1 to your destroy methods. You can add more lines if you wish to include more destroy methods
- alns.add_repair_operator(repair_1) – change repair_1 to your repair methods. You can add more lines if you wish to include more repair methods
- criterion = ... – you may choose from HillClimbing, RecordToRecordTravel or SimulatedAnnealing. Refer to README in the alns github for more detail
- weights adjustment strategy

$$\psi = \begin{cases} \omega_1 & \text{if the new solution is the global best,} \\ \omega_2 & \text{if the new solution is better than current one,} \\ \omega_3 & \text{if the new solution is accepted,} \\ \omega_4 & \text{if the new solution is rejected} \end{cases}$$

omegas = [...] – a list of 4 scores for $[\omega_1, \omega_2, \omega_3, \omega_4]$

lambda = ... – a number between 0 and 1, the decay parameter that controls how sensitive the weights are to changes in the performance of the destroy and repair methods

[1] Note that the `ALNS.py` in the `alns` package is a slightly modified version from the GitHub package, allowing you to show a progress bar when ALNS is being run. You should not modify this package.

- save_output('YourName', solution, 'solution') – change 'YourName' to your full name

2. evrp_skeleton.py – builds solution state for EVRP. Change the file name to `evrp.py` (for `alns_main.py` to call)

Classes

- Parser – get the information in xml file and reformat it into correct class **(do not modify)**
- Node – the base class for nodes, keeps track of the id, type and position **(do not modify)**
- Depot – inherits from Node **(do not modify)**
- Customer – inherits from Node, keeps track of the service time as well **(do not modify)**
- ChargingStation – inherits from Node, keeps track of the charging status for vehicles
  - Charging functions are provided, you may choose to use those functions, or create your own helper functions. **Do not modify the existing functions in this class.**
- Vehicle – keeps track of the status for vehicles. You may create your own helper functions. **Do not modify the existing functions in this class.**
- EVRP – the solution state. **This is the class for you to implement.**
  - `split_route`: refer to the split function presented in section 3.3 of the paper that constructs a feasible solution from a given TSP tour (i.e. construction heuristic). You can implement your own scheme for ensuring feasibility (i.e. not following the paper). Hints are provided in comments in the python file. You may create helper functions in this class.

Annex B: **Explanation of Datasets and Output**

`filtered-montoya-et-al-2017_instances.zip` contains a set of test instances in xml format. The following is reproduced from the `readme` file:

```
Instances are named using the following convention: tcAcBsCcDE, where:
- A is the method used to place the customers (i.e., 0: random uniform,
1: clustered, 2: mixture of both)
- B is the number of customers
- C is the number of the CSs,
- D is 't' if the CSs are located using a p-median heuristic and 'f' if
the CSs were randomly located
- E is the number of the instance for each combination of parameters
(i.e., E={0,1,2,3,4}).

<network>
- the node with type=0 is the depot
- the nodes with type=1 are the customers
- the nodes with type=2 are the charging stations (CSs)
- coordinates are given in km
- nodes with type=2 define the type of charging station (slow, normal,
fast) in tag <cs_type>
- computations must be done using double precision (14 decimal)
Euclidean distances

<fleet>
- There is just one type of electric vehicle in the 120 instances
- Routes start and end at node 0 (the depot)
- <speed_factor> is given in km/h
- <consumption_rate> defines the energy consumption in wh/km
- <battery_capacity> defines the total capacity in wh
- <function cs_type="X"> defines the charging function of the electric
vehicle when charged at a station of type X
- The charging functions are piecewise linear functions with 3 break
points (plus point 0,0)
- The break points are given in 2D coordinates
(X:<charging_time>,Y:<battery_level>)
   - <battery_level> is given in wh
   - <charging_time> is given in h

<requests>
- Each customer has 1 request
- Each customer has a service time
```

Running the program:

Run in command line

```
python alns_main.py <instance> <random_seed>
```

For example,

```
python alns_main.py ./../sample_instances/tc0c10s2cf1.xml 606
```

Note again that for grading, we will only be using the test instances with 10 and 20 customers.

Sample instances and output:

There are 5 sample instances provided for your testing in the `sample_instances` folder, with the respective results in the `sample_output` folder. For each sample instance, there are 4 corresponding files to show the results, which can be generated via the skeleton code described in Annex A:

1. The file ending with `inital.txt` – initial solution
2. The file ending with `initial.jpg` – visualization plot for initial solution
3. The file ending with `solution.txt` – solution after 1000 ALNS iterations
4. The file ending with `solution.jpg` – visualization plot for solution after 1000 ALNS iterations

jpg files:

The files contain your name, instance's name, objective value and the routes as described below:

- blue dot represents depot
- red dot represents a customer
- green dot represents a charging station

e.g. for the instance tc0c10s2cf1, the edges show the route depot -> charging station 12 -> customer 6 -> charging station 12 -> depot. The path can be found in the corresponding txt file.

txt files:

The files contain your name, instance's name, objective value and the routes.

It shows the order, the type and the id of the nodes visited of each route, e.g. for the following route, the vehicle travels from depot 0 to station 12 and charges 10789.94339464934 of energy, then travels to customer 6, then station 12 and charges 12026.684523383823 of energy, and back to depot.

```
Route 0:
depot 0
station 12 Charge (10789.94339464934)
customer 6
station 12 Charge (12026.684523383823)
depot 0
```