Project 2 – Heuristic search for capacitated arc routing problem (CARP)

1 Overview

In this project you need to design and implement heuristic search algorithms for CARP, a classical NP-hard arc routing problem. An introduction to CARP (including problem formulation and solution representation) can be found in the slides we provide (CARP.pdf). Overall, CARP is a constrained optimization problem, and your algorithm needs to find high-quality feasible solutions to it. The scores you get in this project will be given according to your algorithm's performance in our test.

2 General Rules

After you submit your project, we will run some scripts to test your CARP solver on different CARP problem instances. To make this process as smooth as possible, the package you submit must satisfy the following requirements.

1) Algorithm description

A solver description (in pdf format) must be submitted, in which you should describe the core idea of your design, entail each component of your algorithm, illustrate the algorithm structure and give the pseudo code.

2) Programming aspects

In order to get rid of the operating system related issues and the execution efficiency issues of different programming languages, your algorithm must be implemented using Python 2.7 (Python 3.0+ not included) and the only allowed library is numpy.

The name of your executable CARP solver must be CARP_solver.py

3) Input and output

a) Input

In the test, we will repeatedly call your CARP solver through scripts. Specifically, the format of the solver call is as follows (the test environment is Unix-like):

python CARP_solver.py < CARP instance file > -t <termination> -s <random seed>

carp_solver.py is your executable CARP solver.

< carp instance file > −t < termination > −s < random seed > are the arguments passed to your CARP solver

< carp instance file > is the absolute path of the test CARP instance file

−t < termination > specifies the termination condition of your algorithm. Specifically, <termination> is a positive number which indicates how many seconds (in

Wall clock time, range: [60s, 600s]) your algorithm can spend on this instance. Once the time budget is consumed, your algorithm should be terminated immediately.

−s < random seed > specifies the random seed used in this run. In case that your solver is stochastic, the random seed controls all the stochastic behaviors of your solver, such that the same random seeds will make your solver produce the same results. If your solver is deterministic, it still needs to accept **−s < random seed >**, but can just ignore them while solving CARPs.

You can use multithread, but the number of thread shall not exceed 8.

In summary, your CARP solver needs to:

- handle the arguments passed from our scripts while being called as above
- measure the runtime, and terminate after the time budget is consumed. Typically, in python 2.7 you can measure runtime using time.time():

  start = time.time()

  ⋯solving carp

  run_time = (time.time() - start)

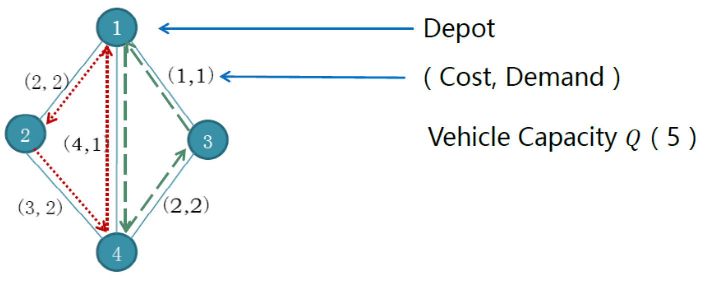
  Note here although our tests rely on your solver's internal time measurement, we will still measure your solver's runtime from external, so make sure that your solver's internal time measurement is accurate.
- use random seeds to control all the stochastic behaviors

b) Output

Your solver must print messages to the standard output. These messages will be used to evaluate this solver run. Your solver has to print two lines. Each of them, according to its first char, must belong to one of the categories described below. Lines that do not start with one of the patterns below will be considered a comment and hence ignored.

Solution line begins with a lower case "s" followed by a space (ASCII code 32). Only one such line is allowed and it is mandatory. The best solution your solver has found for the CARP problem instance must be printed in this line. The format of the solution can be found in the slides we provide (CARP.pdf). For example, for the CARP instance given below, the "s" line is:

s 0,(1,2),(2,4),(4,1),0,0,(4,3),(3,1),0



Quality line begins with a lower case "q" followed by a space (ASCII code 32). Only one such line is allowed and it is mandatory. The solution quality (i.e., the total cost) of your best found solution must be printed in this line. The "q" line of the above example is:

总Cost，这里特别注意，到4节点的最短距离不是4，而是3

q 15

4) Format of CARP problem instance

The format of the CARP problem instances can be found in the instruction we provide (CARP_format.txt), and we also provide some CARP instance files as examples (CARP_samples/).

3 Evaluation

We divide the solver evaluation into three parts:

Usability test (30%): In this test we will use some CARP problem instances to check whether your solvers are usable. Here the meanings behind usability are twofold:

a) (15%) Your solvers can satisfy the input and the output requirements (See Section 2). The failure cases include: cannot handle the inputs, the output messages are undesirable (missing 's' line or 'q' line, the format of the output solution is incorrect or the total cost of the output solution is not correctly calculated), crash.

b) (15%) The best found solution output by your solver is a feasible solution (since CARP is a **constrained** problem). Infeasible solutions violate at least one constraints of CARP.

Once your solvers have passed a test, you will get all the corresponding scores. Note only those solvers that have been through test a) will be tested in b).

Efficacy test (50%): In this test we will focus on how good your solvers are. Specifically, we will select different CARP instances with different sizes from existing CARP benchmarks to test your solvers. For each instance, we will set a common cut-off time for all participant solvers and compare their solution quality.

The scores you get in this test depend on the rankings your solvers obtain. Note only those solvers that have been through usability test will be tested in efficacy test.

Robustness test (20%): A robust solver is said to be applicable to a wide range of problem instances. Although your solvers may show good performance in the efficacy test, they may perform badly on unseen instances. In this test, for each participant solver, we will generate a fixed number of instances that are very hard for it. After this we will put all the newly generated instances into a pool, forming a test set. Finally, all the participant solvers will be tested on this set. Similar to efficacy test, we will set a common cut-off time for all participant solvers and compare their solution quality.

The scores you get in this test depend on the rankings your solvers obtain. Note only those solvers that have been through usability test will be tested in robustness test.

3 Test Environment

1) Operation System: Ubuntu
2) Server CPU：2.5GHz, 14-core
3) Python version: 2.7.12

PS：

1) A time measure method:

```
 xxxxSearch(ElapsedCpuTimer elapsedTimer)
   avgTimeTaken = 0
   acumTimeTaken = 0
   remaining = elapsedTimer.remainingTimeMillis()
   numIters = 0

   remainingLimit = RESERVED_LIMIT //An experience value
    while(remaining > 2*avgTimeTaken && remaining > remainingLimit)
        ElapsedCpuTimer elapsedTimerIteration = new ElapsedCpuTimer();

        ONE_ITERATION();

        numIters++;
        acumTimeTaken += (elapsedTimerIteration.elapsedMillis());

        avgTimeTaken  = acumTimeTaken/numIters;
        remaining = elapsedTimer.remainingTimeMillis();

    }

}
```

2) Performance reference: High Performance Python: Practical Performant Programming for Humans
3) Paper reference link: http://scholar.google.com