Department of Mathematics and Computer Science
University of Southern Denmark, Odense

March 21, 2020
Marco Chiarandini

DM865 - Heuristics and Approximation Algorithms

Obligatory Assignment – Part 1, Spring 2020

~~**Deadline: 31st March 2020 at noon.**~~
Deadline: April 13, 2020 at noon.

- This is the *first obligatory assignment* of Heuristics and Approximation Algorithms.
- The assignment has to be carried out in pairs or individually. Groups of larger sizes are not allowed. Individual participation is discouraged.
- The submission is electronic via `http://valkyrien.imada.sdu.dk/DOApp/`.
- You have to hand in:
  - The source code of your implementation of a heuristic solver. Submit all your files in a `.tgz` archive. Your must comply to the requirements listed in this document.
  - A report that describes the work you have done and presents the results obtained. The document should not exceed 10 pages and must be in PDF format. You cannot list source code, in case use pseudocode. You can write in Danish or in English.
- At the end of the report, add a process analysis where you explain how the group work went.
- Changes to this document after its first publication on February 26 may occur. They will be emphasized in color and if they are major they will be announced via BlackBoard.
- A starting package containing the instances and the code to read them is available at this link:

`https://github.com/DM865/CVRP`

Introduction

The aim of this assignment is to design, implement and report local search heuristic algorithms for solving the Capacitated Vehicle Routing Problem (CVRP).

Make sure you have read the whole document before you start to work.

Heuristics for Capacitated Vehicle Routing

In vehicle routing problems we are given a set of *transportation requests* and a fleet of *vehicles* and we seek to determine a set of *vehicle routes* to perform all (or some) transportation requests with the given vehicle fleet at *minimum cost*; in particular, we decide which *vehicle handles which requests in which sequence* so that all *vehicle routes* can be *feasibly* executed.
The *capacitated vehicle routing problem* (CVRP) is the most studied version of vehicle routing problems.
In the CVRP, the transportation requests consist of the distribution of goods from a single *depot*, denoted as point 0, to a given set of n other points, typically referred to as *customers*, $N = \{1, 2, \ldots, n\}$. The amount that has to be delivered to customer $i \in N$ is the customer's *demand*, which is given by a scalar $q_i \geq 0$, e.g., the weight of the goods to deliver. The *fleet* $K = \{1, 2, \ldots, |K|\}$ is assumed to be *homogeneous*, meaning that $|K|$ vehicles are available at the depot, all have the same capacity $Q > 0$, and are operating at identical costs. A vehicle that services a customer subset $S \subseteq N$ starts at the depot, moves once to each of the customers in S, and finally returns to the depot. A vehicle moving from i to j incurs the *travel cost* c_{ij}.
The given information can be structured using an undirected graph. Let $V = \{0\} \cup N = \{0, 1, \ldots, n\}$ be the set of *vertices* (or nodes). It is convenient to define $q_0 := 0$ for the depot. In the symmetric case, when the cost for moving between i and j does not depend on the direction, i.e., either from i to j or from j to i, the

Instance	K_{LB}	Construction Heuristic		Local Search	
		cost	time (sec)	cost	time (sec)
CMT01					
CMT02					
CMT03					
CMT04					
CMT05					
CMT11					
CMT12					

Table 1: The table shows the median results from 5 runs per instance of the best heuristic designed. The time limit was set to 60 seconds on a Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz with 16 GB RAM running Ubuntu 16.04.

underlying graph $G = (V, E)$ is complete and undirected with edge set $E = \{e = (i, j) = (j, i) : i, j \in V, i \neq j\}$ and edge costs c_{ij} for $\{i, j\} \in E$. Overall, a CVRP instance is uniquely defined by a complete weighted graph $G = (V, E, c_{ij}, q_i)$ together with the size $|K|$ of the of the vehicle fleet K and the vehicle capacity Q.
A *route* (or *tour*) is a sequence $r = (i_0, i_1, i_2, \ldots, i_s, i_{s+1})$ with $i_0 = i_{s+1} = 0$, in which the set $S = \{i_1, \ldots, i_s\} \subseteq N$ of customers is visited. The route r has cost $c(r) = \sum_{p=0}^{s} c_{i_p,i_{p+1}}$. It is *feasible* if the capacity constraint $q(S) := \sum_{i \in S} q_i \leq Q$ holds and no customer is visited more than once, i.e., $i_j \neq i_k$ for all $1 \leq j \leq k \leq s$. In this case, one says that $S \subseteq N$ is a *feasible cluster*. A solution to a CVRP consists of K feasible routes, one for each vehicle $k \in K$. The routes $r_1, r_2, \ldots, r_{|K|}$ and the corresponding clusters $S_1, S_2, \ldots, S_{|K|}$ provide a *feasible solution* to the CVRP if all routes are feasible and the clusters form a partition of N. Hence, the CVRP consists of two interdependent tasks:

(i) the partitioning of the customer set N into feasible clusters $S_1, \ldots, S_{|K|}$;

(ii) the routing of each vehicle $k \in K$ through $\{0\} \cup S_k$.

1 Your Tasks

Using the test instances described below, you have to submit a report and a Python program that address the following tasks:

- Determine an easy-to-calculate lower bound K_{LB} to the number of vehicles needed to satisfy the demand of all customers. Report in your final text document a table like Table 1 with the lower bounds thus found for each given instance.
- Design and implement one or more construction heuristics.
- Design and implement one or more iterative improvement algorithms. They must terminate in a local optimum.
- Undertake an experimental analysis to compare and configure the algorithms from the previous two points.
- Describe the work done in a report of at most 10 pages. The report must at least contain a description of the best algorithm designed and the experimental analysis conducted. The level of detail must be such that it makes it possible for the reader to reproduce your work.
- In an appendix of the report (that does not count in the 10 pages) report the results of the best algorithms on the test instances made available (see below) in a table like Table 1. You are welcome to report also graphical comparisons and assessment of the way your algorithms scale with respect to the size of the instance (this must be included in the 10 pages).
- Submit your best algorithm in the upload page. The programs will be run on a 64-bit machine with Ubuntu Linux, equivalent to those in the terminal room. A time limit of **60 seconds** will be imposed. If your algorithms are faster you can consider using some basic metaheuristic like random restart or neighborhood change. No other metaheuristic is allowed in this assignment.

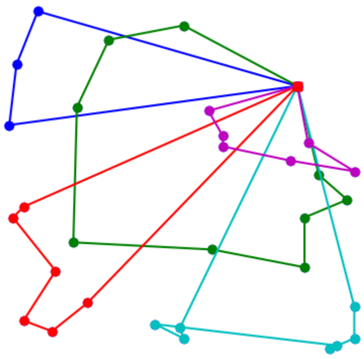

Figure 1: A solution for the `A-n32-k05` instance

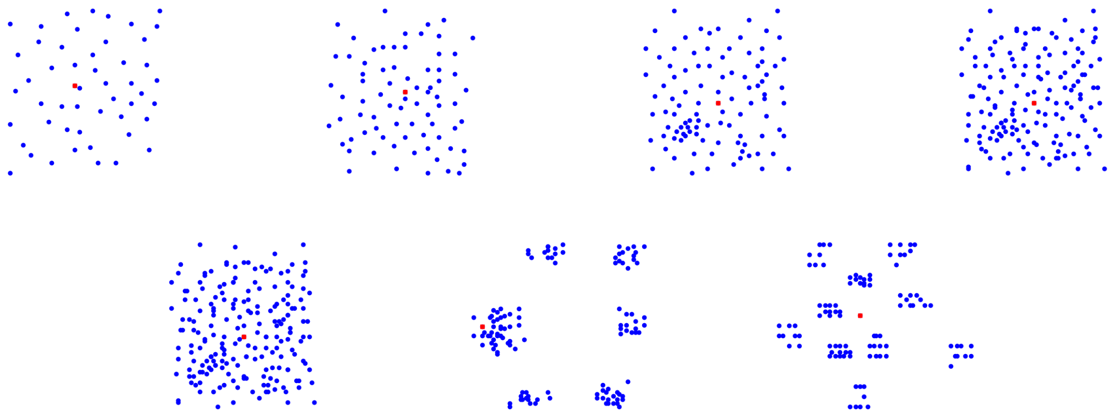

Figure 2: The CMT instances

Practicalities

Associated to this document there is a GIT repository at:

`https://github.com/DM865/CVRP`

The repository is made of a directory `data/` containing the instances, a directory `src/` containing some initial Python 3 code to read the instances, output a solution and produce a graphical view of solutions. The code provides also a framework within which to organize your implementation. The directory `tex` contains the sources of this document and can be therefore ignored.

Instances In the directory `data/` you find the instance `A-n32-k05.xml` that is a small toy instance with 32 nodes. This instance and a heuristic solution is represented in Figure 1. In the directory `data/CMT` you find the set CMT[1] of *middle size* instances with number of nodes ranging between 51 and 200, and in the directory `data/Golden` you find the set Golden[2] of *large size* instances with number of nodes ranging between 241 and 484. The displacement of the nodes in these instances is depicted in Figure 2 and 3. The best known solutions for these instances are reported in Table 2.

Source Code The Python code in the directory `src/` contains the following files:

[1] Christofides, N., Mingozzi, A., Toth, P. The vehicle routing problem. 1979. In Christofides, N., Mingozzi, A., Toth, P., Sandi, C. (Eds.), Combinatorial Optimization. Wiley, Chichester, pp. 315– 338.

[2] Golden, B. L., Wasil, E.A., Kelly, J.P., Chao, I.-M. Metaheuristics in Vehicle Routing. 1998. In T. G. Crainic and G. Laporte, eds, Fleet Management and Logistics. Boston: Kluwer, pp. 33-56.

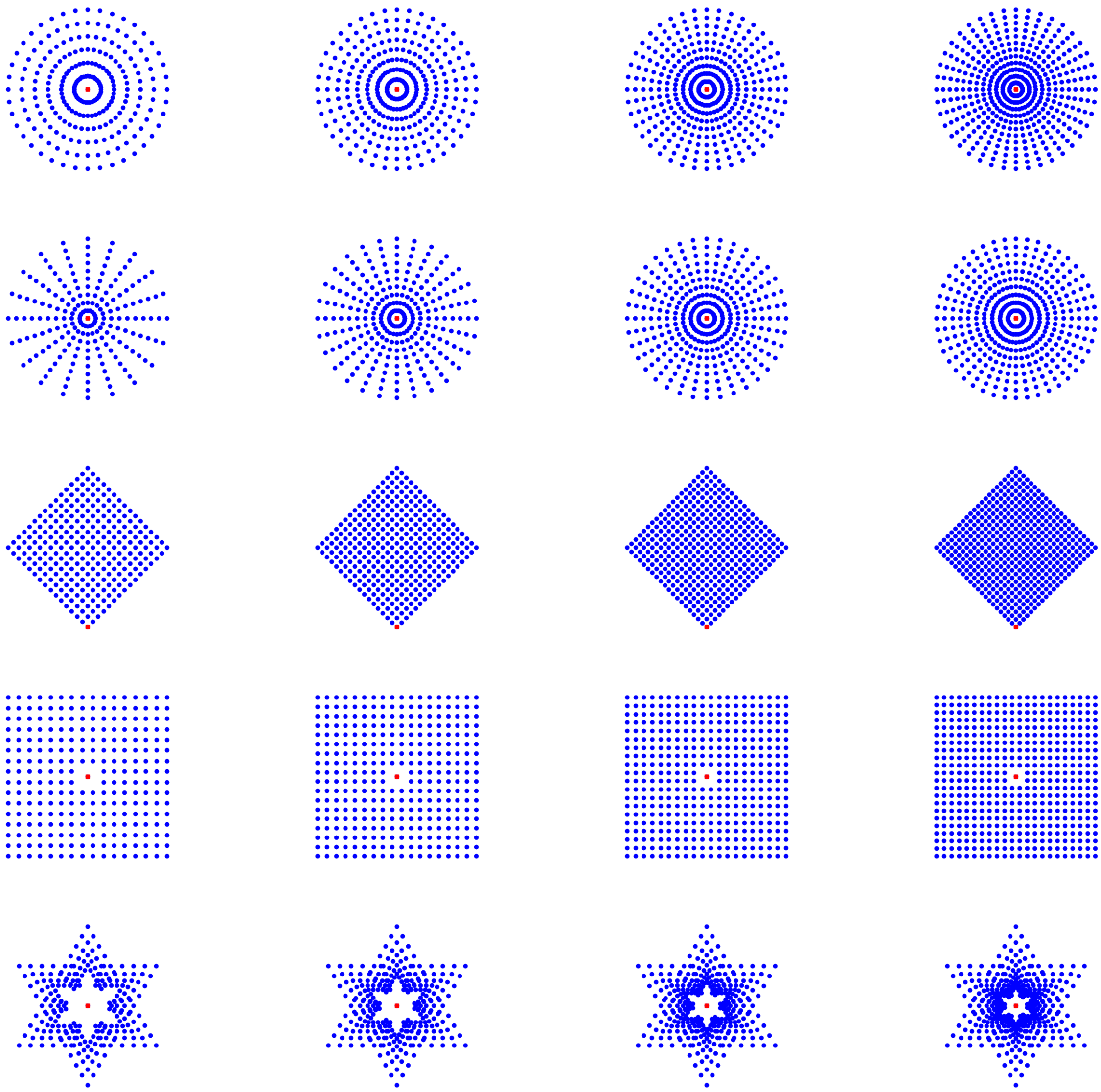

Figure 3: The Golden instances

instance	nodes	best known
CMT01	51	524.61*
CMT02	76	835.26*
CMT03	101	826.14*
CMT04	151	1028.42*
CMT05	200	1291.29*
CMT11	121	1042.11*
CMT12	101	819.56*

instance	nodes	best known
Golden_01	241	5623.47
Golden_02	321	8404.61
Golden_03	401	11036.22
Golden_04	481	13592.88
Golden_05	201	6460.98
Golden_06	281	8404.26
Golden_07	361	10102.68
Golden_08	441	11635.34
Golden_09	256	579.71
Golden_10	324	736.26
Golden_11	400	912.84
Golden_12	484	1102.69
Golden_13	253	857.19
Golden_14	321	1080.55
Golden_15	397	1337.92
Golden_16	481	1612.50
Golden_17	241	707.76
Golden_18	301	995.13
Golden_19	361	1365.60
Golden_20	421	1818.32

Table 2: The best known solutions on the set of instances CMT and Golden. A star indicates that the solution has been proven optimal.

- `data.py` that implements the class `Data` to maintain the data associated with the input instance. It contains an instance reader for the XML format. Objects of this class contain the following data that will be relevant to you: `capacity` giving the capacity of the vehicle and `nodes` that is a tuple container of the nodes of the given network. Each element from the tuple `nodes` is a dictionary with the following keys and values: `id`, the original identifier of the node from the input file, `pt`, the coordinates of the node in complex numbers notation as we saw for the TSP, `tp`, the type of customer: 0 if a depot and 1 if a customer, `rq`, the quantity demanded by the node (if it is a depot this value is 0). Nodes in the tuple `nodes` are organized in such a way that the depot is the first element followed by all others. Each node can be accessed in constant time through the index in the tuple. Hence, internally the depot has always index zero.

  The class `Data` contains also methods for printing the instance, reporting statistics, calculating distances and plotting.

- `solution.py` that implements the class `Solution` to store data relative to a candidate solution. For now it assumes to store the solution in a list of lists, called `routes`. However, this is up to changes according to your needs. It then assumes that you finish implementing the methods `valid_solution` and `cost` that determine the feasibility and the quality of the candidate solution.

  The class `Solution` contains also methods for writing the solution file and for plotting the solution. These methods assume that solutions are represented as lists of lists, where every inner list representing a route starts with the depot and ends with the depot. Note that the solution writer outputs the original identifier of the nodes and not the one used to represent them internally.

- `solverCH.py` that implements the class `ConstructionHeuristics`. Currently only a canonical construction is implemented that routes costumers in the order they are stored. The implementation might be helpful to see how to use the data from an object of class `Data`.

- `solverLS.py` that implements the class `LocalSearch`.

- `main.py` that implements the main program defining the objects and calling the methods defined in the other files. It provides a starting example that can be modified at your best convenience. It also defines the parameters to be specified when the program is run.

The program is executed as specified below:

```
$ python3 main.py -h
usage: main.py [-h] [-o OUTPUT_FILE] -t TIME_LIMIT instance_file
```

```
positional arguments:
  instance_file   The path to the file of the instance to solve

optional arguments:
  -h, --help      show this help message and exit
  -o OUTPUT_FILE  The file where to save the solution and, in case, plots
  -t TIME_LIMIT   The time limit
marco@nat-102098:~/IMADA/DM865/CVRP/src$
```

for example:

```
python3 main.py -t 30 -o A-n32-k05 ../data/A-n32-k05.xml
```

Included in the directory there is also a `Makefile` that can be used to automatize tasks. For example, the call above can be also achieved with:

```
make A-n32-k05
```

In addition, in the Makefile there is an example on how to use the code profiler: `cProfile`.
It is possible to modify all these files and to add new ones.

Submission Guidelines

The submission is done from `http://valkyrien.imada.sdu.dk/DOApp/`
You have to submit a tar gzip file. Your directory must be organized as follows:

```
ob1
  |-  doc
  |-  src
```

In the directory `doc` put the report with your full name and username. Keep it shorter than 10 pages.
In `src` put all the source code.
You can then create the tar gzip file from the directory `ob1/` as follows:

```
tar czvf ob1.tgz doc src
```

You can submit as many times as you wish, each new submission overwrites the previous one.
To be considered acceptable, your source code must satisfy the following requirements:

i) it must execute the heuristic that you chose as the best one when called as follows:

```
python3 main.py -t 30 -o [an_instance] ../data/[an_instance].xml
```

The program must solve the specified instance and halt before the specified time limit.

ii) At termination the program must write the solution in the format described below in a file whose name is the one given for the parameter `-o` plus the extension `.sol`. The starting code provided has a function `write_to_file` to do this but probably you will need to modify it if you change the solution representation. The function is called from the main file. The solution written in the file must be valid, that is, feasible.

Right after the submission the program will be tested and if it does not satisfy the requiremets above you will receive an email and the submission will be invalid.
In addition, the submission system will execute your program and compare it against your peers on a set of unspecified instances. Therefore, you should submit your best algorithm early and eventually revise your submission. Before submitting, test your implementation on the IMADA machines. If you are using additional python modules not present in setting of the Computer Lab machines, write to Marco.

Solution file The solution file must list the routes one per line. Each route is a comma separated list of nodes to be visited in the given order. The node identifier must be the original one from the input file. Routes must start with the depot and finish with the depot.
The following listing provides an example of solution file for a valid solution to the instance CMT02:

```
76,1,2,3,4,5,6,7,76
76,8,9,10,11,12,13,76
76,14,15,16,17,18,19,20,76
76,21,22,23,24,25,26,27,76
76,28,29,30,31,32,76
76,33,34,35,36,37,38,39,76
76,40,41,42,43,44,45,76
76,46,47,48,49,50,51,52,76
76,53,54,55,56,57,58,59,76
76,60,61,62,63,64,65,66,76
76,67,68,69,70,71,72,73,74,75,76
```

Solutions files have extension `.sol`.

Remarks

Remark 1 This is a list of factors that will be taken into account in the evaluation:

- quality of the final results;
- level of detail of the study;
- complexity and originality of the approaches chosen;
- organization of experiments that guarantees reproducibility of conclusions;
- clarity of the report;
- presence of the analysis of the computational costs involved in the main operations of the local search.
- effective use of graphics in the presentation of experimental results.

Remark 2 Note that a few, well thought algorithms are better than many naive ones!

Remark 3 If you search on Internet, the literature on heuristics for vehicle routing problems is vast but not every article is relevant. The following are three relevant articles:

- Clarke, G., and J. W. Wright. "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points." Operations Research 12, no. 4 (1964): 568-81. `http://www.jstor.org/stable/167703.`
- Ann Melissa Campbell and Martin Savelsbergh, Efficient Insertion Heuristics for Vehicle Routing and Scheduling Problems, Transportation Science, 38(3), 369-378, 2004, `http://dx.doi.org/10.1287/trsc.1030.0046`,
- G. Kindevater and M. Savelsbergh, "Vehicle routing 2 - handling side constraints," tech. rep., School of Industrial and Systems Eng., Georgia Institute of Tech., 1995.