IFT6751: Homework 2

Gabriel C-Parent

March 21, 2015

1 Introduction

In this homework, two different approaches to solve the capacitated vehicle routing problem (CVRP) were designed. The first one is a genetic algorithm using specialized crossover and mutation operators whereas the the second one uses the Tabu Search to control local search with the λ-interchange neighbourhood.

Two greedy local search methods are used within both methods to improve the results such as the 2-opt descent and and the λ-interchange descent.

Another method based on the Clark & Wright savings is also used to initialize solutions for both metaheuristics.

Both methods were tested against problem instances without length limit, from [1] and compared to the best known solutions.

A description of all the optimization methods along with special implementation details is given. Some experimental results are then compared based on running time, implementation complexity and results quality.

Finally, a user guide is given in the supplementary section 9.1.

2 Local Search Methods

2.1 2-opt descent

First, a simple and fast optimization method for the TSP was needed to improve the path of each routes.
The local search method used to optimize individual routes is the steepest improvement method as described in [4]. Basically, it is a greedy use of the well known 2-opt method.

At each iteration, the best possible 2-opt is chosen, according to the reduction in total distance, until there isn't any possible improvement. The complexity of the procedure $O(n^2)$ on the number of edges in the route.

Although it might seem slow, usually the number of edges is quite small and the time spent optimizing routes is relatively negligible.

Simplified python code can be seen in the code section 9.4.

2.2 λ-interchange descent

Another optimization method was needed to exchange clients between routes, in this case the λ-interchange descent with λ=1 [3].

The possible transitions are the transfer of a client from a route to another and the swap of two clients between two routes. Since only feasible solutions are considered, only transitions that do not violate the capacity limit are considered.

The procedure chooses the interchange with best possible improvement and applies it. Then the 2-opt descent is applied to the modified routes and the process is repeated until it gets stuck in a local minima.

3 Random Savings Initialization

The Clark & Wright savings algorithm is a well known simple heuristic for the CVRP. Many improvements were suggested for this heuristic [2].

The one used in this work is a slight variant of the parallel savings, where instead of choosing the best the best saving and merging the corresponding routes, the k best savings are found and one is randomly chosen.

This procedure is used in both the Tabu Search and Genetic Algorithm procedures.

In the Genetic Algorithm procedure, the random savings is used to generate good initial solutions. The initialization step is costly but the quality of the initial population is great.

In the Tabu Search procedure, the random savings is used to initialize a solution, which is then locally explored until convergence is achieved.

4 Genetic Algorithm

4.1 Encoding

The solutions are encoded using the Route object. Basically, a route is a list of clients that starts at the depot and ends at the depot, where no client is repeated. The solution is a list of routes, where each client is in exactly one route.

This representation isn't really friendly to classical genetic operators but allows functions defined on routes and solutions to be shared for both the Tabu Search and Genetic Algorithm representations.

4.2 Objective Function

The objective function is the minimization of the total distance of the routes.

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{r \in solution} distance(r) \\ \text{subject to} & weight(r) \leq vehicle\ capacity \end{array}$$

4.3 Selection

The parents to the next generation are selected using the simple and well known binary tournament selection.

4.4 Crossover

The crossover used here can be explained in terms of the cluster-first route-second methods.

Since the initial population is generated using the random savings 3 with 2-opt descent 2.1 optimization applied on the resulting routes, the solutions are already good and the routes usually are pretty close to the optimal solution. However the choice of clusters, that is which clients belong together, is somewhat far from optimal.

Therefore, what was needed was a crossover that would allow for new clusters to be created while using part of the routes of the parents. To do so, routes are selected from both parents and the remaining clients have routes assigned to them using the Clark & Wright savings. The second part is where new clusters should be formed.

4.4.1 Selection Of Inherited Routes

In both parents, routes are sorted by the angle of their centroid (average of clients coordinates) relative to the depot and a quarter of the routes of each parents is selected so that overall about half the routes of the child will be inherited and the other half will be newly created in the second part.

Let n_1 be the number of routes in the first parent. A contiguous sequence $\frac{n_1}{4}$ routes in selected in the first parent. Since they are sorted by their angle to the depot, the hope is that they are all very close to each other.

In the second parents, only routes that share no clients with the selected routes from parent 1 are considered. Obviously, it is possible that much less than $\frac{n_1}{4}$ remain and of those, at most $\frac{n_1}{4}$ hopefully contiguous routes are chosen using the same mechanism as for parent 1.

Overall, at most $\frac{n_1}{2}$ routes are inherited.

4.4.2 Assigning Routes To Remaining Clients

Once the routes inherited are selected, the rest of the clients have routes assigned to them using the Clark & Wright parallel savings. The path of each resulting routes is then improved using the 2-opt descent local search.

4.5 Mutation

A n optimal λ-interchange 2.2 with λ=1 are applied. In this case, n is set to 5.

4.6 Population Swap

An elitist population replacement is implemented. A fixed percentage of the best solutions is automatically inherited from the previous generation at each iteration.

4.7 Stopping criteria

A number of iterations is given.

5 Tabu Search

With minor variations, the Tabu Search implementation was the same as described in [3].

5.1 Solution Space

Only feasible solutions are considered, that is those not violating the capacity constraint.

5.2 Objective Function

The objective function is the minimization of the total distance of the routes.

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{r \in solution} distance(r) \\ \text{subject to} & weight(r) \leq vehicle\ capacity \end{array}$$

5.3 Neighbourhood Structure

The neighbourhood of a solution is all the feasible solutions that can be reached by applying the λ-interchange, with λ=1.

5.4 Tabu List

The tabu list is implemented as a matrix keeping the expiration of tabu status for each move.
In this case, since the λ-interchange allows only exchange of clients between routes, the tabu is based on the clients implicated in the move. A *null* client is used to represent the interchanges that imply only one client. No information is kept about the routes.

A fixed tabu duration is chosen according to the simple heuristic outlined in [3], that is a value is picked in $[0.4n,\ 0.6n]$ is picked randomly, with n being the number of clients in the problem.

5.5 Asipration Criteria

5.6 Intensification

5.7 Diversification

The diversification strategy used here is to restart the Tabu Search with a new solution created by random savings.

5.8 Stopping Criteria

As with the genetic algorithm, the stopping criteria can be a set number of iterations or a time limit, depending on the type of constraints the user has.

6 Experimental Results

6.1 Random Savings

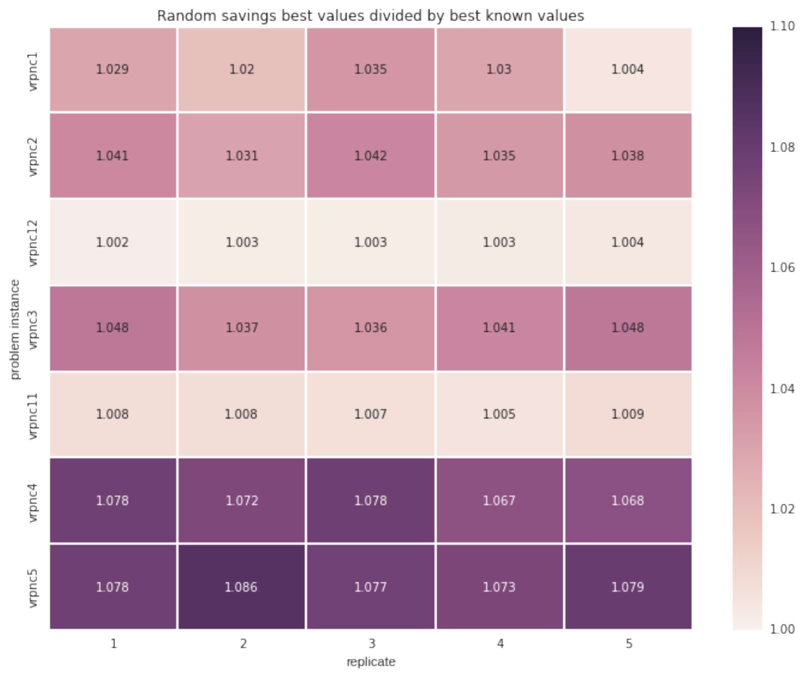

Figure 1: The random savings algorithm is used to generate solutions for each problem instance. Each of the 5 replicates is a run of 30 seconds of generating solutions in a Monte Carlo fashion, with specified depth of suboptimal choices considered (see 3 for more details). The total distance used by the best solution of each replicate is divided by the best known value of each problem instance. Seeing as each run lasts only 30 seconds, the results are surprisingly good.

6.2 Genetic Algorithm

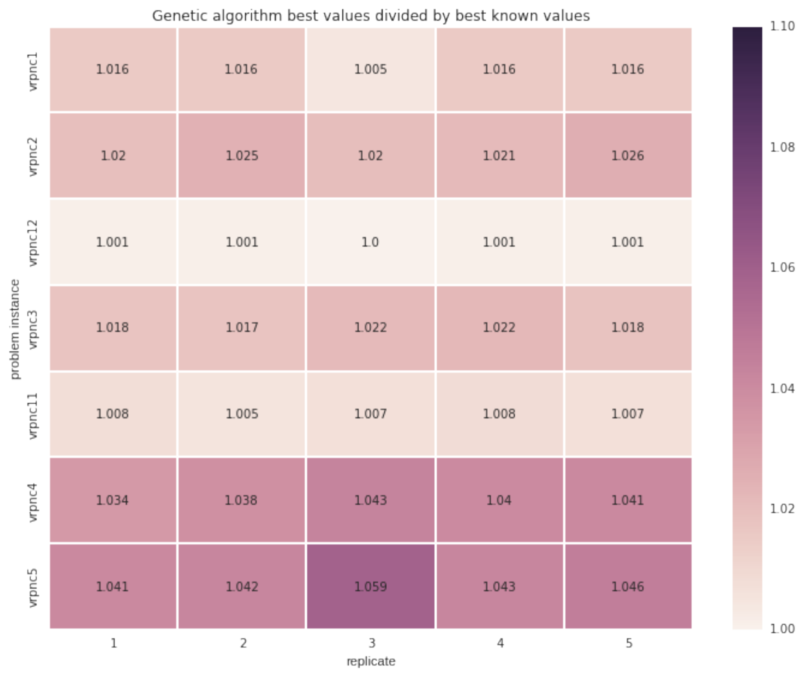

Figure 2: The genetic algorithm is used to generate solutions for each problem instance. Each of the 5 replicates is a run of 100 iterations with a population size of 200. Additional parameters are specified in the corresponding IPython notebook. With 5 replicates, the total distance used by the best solution of each replicate is divided by the best known value of each problem instance. The results are acceptable, but not as good as expected given that the random savings was much quicker and the solutions generated here are only slightly better. Then again, the last bits of performance are usually the hardest to get.

6.3 Tabu Search

7 Discussion

The CVRP is a hard problem and the experimental results are encouraging.

7.1 Random Savings: Speed And Simplicity

The Random Savings 1 had much more success than expected, especially since there was a 30 seconds time limit for solving. For vrpnc1, vrpnc12 and vrpnc11 the best results are very close to optimality. This can be explained by the location of the clients for vrpnc11 and vrpnc12 while vrpnc1 has the least amount of clients.

Overall the simplicity of implementation and speed of solving made this algorithm very interesting. That is why both the Genetic Algorithm and Tabu Search solvers use it for different roles.

7.2 Genetic Algorithm: Modularity

The Genetic Algorithm managed to improve quite a lot the initial solutions generated by the random savings. The algorithm is time consuming but the results achieved are interesting, although not as good as the best known values.

The implementation of the genetic algorithm more modular than the Tabu Search and as such allowed for many options of crossover and mutation operators to be tried.

In my opinion, this algorithm was simpler and much more fun to implement than the Tabu Search.

7.3 Tabu Search: Performance

The tabu search was quite hard to implement compared to previous algorithms. In my opinion it would be because all of the design parameters must be known beforehand to make it efficient. The choice of the local optimization dictates how the rest of the algorithm should be implemented more strictly than I would like.

However, since the best known values have been obtained using such algorithm, the idea must be good.

7.4 Non-dominated Treatments

8 Conclusion

References

[1] N Christofides, A Mingozzi, P Toth, and C Sandi. Combinatorial optimization. 1979.

[2] M D Nelson, K E Nygard, J H Griffin, and W E Shreve. Implementation techniques for the vehicle routing problem. *Comput. & Ops. Res.*, 12(3), 1984.

[3] I H Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41, 1993.

[4] Áslaug Sóley Bjarnadóttir. Solving the vehicle routing problem with genetic algorithms. Master's thesis, Technical University of Denmark, 2004.

9 Supplementary Materials

9.1 User Guide

The following section should help with verification of the results and repeatability.

The language used is a mix of python and cython, an optimising compiler that allows static typing and generates C code.

9.1.1 Working Environment

All computational results obtained in this work should be repeatable given a suitable python environment. The particular dependencies of this work are the Cython, Numpy, IPython and Seaborn along with standard python environment.

The following python environment was used:

```
CPython 2.7.9
ipython 2.2.0

numpy 1.9.2
cython 0.21
ipython 2.2.0
seaborn 0.5.1

compiler   : GCC 4.4.7 20120313 (Red Hat 4.4.7-1)
system     : Linux
release    : 3.13.0-46-generic
machine    : x86_64
processor  : x86_64
CPU cores  : 4
interpreter: 64bit
```

As of now, my personal recommendation is to use the excellent Anaconda python distribution from Continuum Analytics.

9.1.2 Running Computational Results

All computational results and figures are contained in the form of IPython Notebooks with the hope of allowing repeatability and reproducibility.
If IPython is available on the computer, an IPython Notebook service can be launched from command line using the following call.

```
ipython notebook
```

This allows viewing and recomputation of the results.

Alternatively, the IPython notebooks can be viewed online if it is reachable from a url using the nbviewer tool. This allows viewing a static version of an IPython Notebook.

9.2 Experimental Solutions

Here are the resulting solutions for each problem. The depot is represented using the index 0. All routes start and end at the depot. A list of routes is considered a solution and the total length of the routes is also given.

9.2.1 Random Savings

```
vrpnc1
distance 526.926289575
[
[0, 8, 26, 31, 28, 3, 36, 35, 20, 22, 1, 32, 0]
[0, 12, 37, 44, 15, 45, 33, 39, 10, 49, 5, 46, 0]
[0, 6, 14, 25, 24, 43, 7, 23, 48, 27, 0]
[0, 38, 9, 16, 50, 30, 34, 21, 29, 2, 11, 0]
[0, 47, 4, 17, 42, 19, 40, 41, 13, 18, 0]
]

vrpnc2
distance 860.774267149
[
[0, 47, 36, 69, 71, 60, 70, 20, 37, 5, 29, 0]
[0, 7, 11, 66, 65, 38, 26, 0]
[0, 12, 72, 39, 9, 31, 10, 58, 0]
[0, 50, 25, 55, 18, 24, 49, 16, 33, 0]
[0, 34, 46, 52, 13, 57, 15, 27, 45, 0]
[0, 48, 21, 61, 22, 62, 28, 74, 0]
[0, 51, 3, 44, 32, 40, 17, 0]
[0, 35, 53, 14, 59, 19, 54, 8, 0]
[0, 67, 0]
[0, 73, 1, 43, 42, 64, 41, 56, 23, 63, 0]
[0, 75, 68, 6, 2, 30, 4, 0]
]

vrpnc12
distance 821.030002939
[
[0, 3, 7, 8, 11, 9, 6, 4, 2, 1, 75, 0]
[0, 5, 99, 100, 97, 93, 92, 94, 95, 96, 98, 0]
[0, 13, 17, 18, 19, 15, 16, 14, 12, 10, 0]
[0, 21, 23, 26, 28, 30, 29, 27, 25, 24, 22, 20, 0]
[0, 32, 33, 31, 35, 37, 38, 39, 36, 34, 0]
[0, 47, 49, 52, 50, 51, 48, 45, 46, 44, 40, 41, 42, 43, 0]
[0, 55, 54, 53, 56, 58, 60, 59, 57, 0]
[0, 67, 65, 63, 74, 62, 66, 0]
[0, 81, 78, 76, 71, 70, 73, 77, 79, 80, 72, 61, 64, 68, 69, 0]
[0, 91, 89, 88, 85, 84, 82, 83, 86, 87, 90, 0]
]

vrpnc3
distance 855.739001735
[
[0, 6, 96, 99, 5, 60, 83, 84, 17, 45, 46, 36, 47, 48, 7, 82, 8, 18, 0]
[0, 12, 80, 68, 29, 24, 55, 25, 4, 54, 26, 58, 53, 0]
[0, 13, 97, 92, 37, 98, 85, 93, 59, 95, 94, 0]
[0, 27, 69, 70, 30, 20, 66, 65, 71, 35, 9, 51, 1, 0]
[0, 28, 76, 77, 3, 79, 78, 34, 81, 33, 50, 0]
[0, 40, 73, 22, 41, 23, 67, 39, 56, 75, 74, 72, 21, 0]
[0, 87, 42, 100, 91, 61, 16, 86, 38, 44, 14, 43, 15, 57, 2, 0]
[0, 89, 52, 88, 62, 11, 19, 49, 64, 63, 90, 32, 10, 31, 0]
]

vrpnc11
distance 1047.45609368
[
[0, 2, 1, 3, 4, 5, 6, 7, 9, 10, 11, 15, 14, 13, 12, 8, 118, 0]
[0, 17, 16, 19, 22, 24, 25, 28, 31, 27, 30, 33, 34, 36, 35, 32, 29, 26, 23, 20, 21,
    109, 0]
[0, 88, 87, 95, 102, 96, 93, 94, 97, 115, 116, 100, 99, 103, 104, 107, 106, 105,
    120, 0]
[0, 98, 68, 73, 76, 77, 79, 80, 78, 75, 72, 74, 71, 70, 69, 67, 0]
[0, 101, 40, 43, 45, 48, 51, 50, 49, 47, 46, 44, 41, 42, 39, 38, 37, 0]
[0, 110, 52, 54, 57, 59, 65, 61, 62, 64, 66, 63, 60, 56, 58, 55, 53, 0]
[0, 111, 86, 85, 89, 92, 91, 90, 18, 114, 108, 83, 113, 117, 84, 112, 81, 82, 119,
    0]
]

vrpnc4
distance 1097.15398016
[
[0, 21, 73, 72, 56, 23, 67, 39, 139, 25, 55, 4, 110, 149, 0]
[0, 132, 31, 10, 108, 131, 32, 90, 63, 126, 62, 148, 88, 127, 146, 0]
[0, 52, 18, 106, 82, 114, 8, 45, 125, 83, 147, 89, 0]
[0, 61, 17, 113, 86, 16, 141, 44, 140, 38, 14, 119, 91, 98, 0]
[0, 12, 109, 54, 130, 134, 24, 29, 121, 68, 150, 80, 76, 50, 0]
[0, 69, 101, 70, 122, 30, 20, 128, 66, 65, 136, 35, 135, 71, 103, 51, 1, 0]
[0, 6, 104, 99, 59, 92, 37, 100, 42, 142, 43, 15, 57, 145, 41, 22, 133, 75, 74, 0]
[0, 60, 118, 5, 84, 85, 93, 96, 94, 0]
[0, 112, 105, 26, 138, 28, 111, 27, 0]
[0, 116, 77, 3, 79, 129, 78, 34, 120, 9, 81, 33, 102, 0]
[0, 13, 117, 95, 97, 87, 144, 115, 2, 137, 58, 40, 53, 0]
[0, 48, 124, 46, 47, 36, 143, 49, 64, 11, 107, 19, 123, 7, 0]
]

vrpnc5
distance 1385.48640721
[
[0, 21, 73, 72, 74, 171, 133, 22, 41, 145, 115, 178, 2, 137, 0]
[0, 40, 197, 75, 56, 186, 23, 39, 67, 170, 187, 139, 0]
[0, 164, 135, 35, 136, 65, 66, 71, 161, 103, 51, 1, 0]
[0, 52, 153, 106, 194, 82, 48, 182, 148, 62, 159, 0]
[0, 28, 105, 53, 112, 156, 0]
[0, 58, 152, 144, 57, 15, 43, 142, 42, 172, 87, 97, 117, 13, 0]
[0, 88, 7, 123, 19, 107, 175, 11, 64, 49, 143, 36, 47, 168, 124, 46, 0]
[0, 89, 60, 118, 84, 173, 17, 113, 16, 61, 85, 5, 6, 0]
[0, 94, 95, 59, 92, 151, 37, 98, 93, 99, 104, 96, 183, 0]
[0, 100, 192, 14, 119, 44, 38, 140, 86, 141, 191, 91, 193, 0]
[0, 101, 20, 188, 128, 160, 131, 32, 181, 63, 126, 90, 108, 162, 0]
[0, 111, 184, 68, 150, 80, 177, 54, 109, 12, 154, 138, 0]
[0, 134, 163, 24, 29, 121, 129, 79, 185, 158, 3, 77, 116, 196, 76, 0]
[0, 166, 18, 114, 8, 174, 45, 125, 199, 83, 147, 0]
[0, 27, 132, 69, 122, 30, 70, 10, 189, 31, 190, 127, 167, 146, 0]
[0, 169, 78, 34, 120, 9, 81, 33, 157, 102, 50, 176, 0]
[0, 180, 198, 110, 155, 4, 25, 55, 165, 130, 179, 195, 149, 26, 0]
]
```

9.2.2 Genetic Algorithm

9.2.3 Tabu Search

9.3 Code Examples

9.4 2-opt descent

```python
def steepest_improvement(route, dist):
    """steepest improvement procedure, using 2-opt"""
    best_ind1 = 0
    best_ind3 = 0
    savings = 0.
    proposed_savings = 0.
    # iterate until there isn't any better local choice (2-opt)
    while True:
        savings = 0.
        for ind1 in range(0, len(route.nodes)-2):
            for ind3 in range(ind1+2, len(route.nodes)-1):
                n1 = route.nodes[ind1]
                n2 = route.nodes[ind1 + 1]
                n3 = route.nodes[ind3]
                n4 = route.nodes[ind3+1]
                actual = dist[n1][n2] + dist[n3][n4]
                proposed = dist[n1][n3] + dist[n2][n4]
                proposed_savings = actual - proposed
                if proposed_savings > savings:
                    best_ind1 = ind1
                    best_ind3 = ind3
                    savings = proposed_savings
        if savings > 0.:
                # do the 2-opt exchange
            two_opt(route, best_ind1, best_ind3)
        else:
            return
    return
```