University of Southern Denmark

DM852

13 April 2020

Heuristics & Approximations Algorithms

Submitted To:
Marco Chiarandini
Lene Monrad Favrholdt
IMADA
Institute of Mathematics & Computer Science

Submitted By:
Alexander Lerche Falk
Fall - Master of Computer Science

Contents

1 Introduction

The goal of this project is to show the knowledge acquired by the student in regards to Heuristic- and Local Search Algorithms. The algorithms are developed to suit the Capacitated Vehicle Routing Problem (CVRP). CVRP is a combinatorial optimization problemm, where a set of n customers has to be visited by m vehicles. Each customer can only be visited one vehicle. The sum of demands by the customers assigned to a vehicle must not overseed the capacity of the vehicle. All vehicles start and end their route at the depot (the starting point). The objective is to minimise the travelling cost and try to use as few vehicles as possible. Heuristics and Local Search algorithms are applied to CVRP, where the heuristics are laying the foundation of possible solutions of routes, and the Local Search algorithms try to improve the solutions. Three heuristic algorithms have been developed:

- Nearest Neightbour - Pick nearest customer from the depot or current customer without breaking the capacity
- Furthest Neightbour (with nearest clustering) - Pick customer furthest away from the depot, find the closest customers until the capacity is reached, and generate a route by using Nearest Neightbour
- K-Nearest Neightbour - Pick k nearest customers and randomly pick one of the k options. Proceed until no more customers without breaking the capacity

Two local search algorithms have been developed:

- Two Opt - Pick a route generated by some heuristic. Pick two edges, swap the edges, and compare the distance from before and after the swap. If better, make the swap.
- Three Opt - Pick a route generated by some heuristic. Pick three edges, swap the edges, and compare the distance from before and after the swap. If better, make the swap.

At the end of the report, comparison of the heuristics and results are presented.

2 Nearest Neightbour

The Nearest Neightbour algorithm is a simple, yet powerful, approach to tackle CVRP. The algorithm goes starts at the depot and picks the nearest customer by iterating through all of the customers. It marks the customer and visited, adds it to a vehicle, and continues until the capacity of the vehicle is reached (or nearly reached). It returns to the depot and start over with the remaining customers.

The algorithm is described in pseudocode:

Algorithm 1 Custom CVRP Heuristic - Nearest Neighbour Approach

Require: $Point_1 \dots Point_N$
Ensure: *Solution* (solution to the CVRP instance)

```
function ALGORITHM(Points[ ])
    capacity ← 0
    data ← CVRP instance
    visited ← empty array containing visited nodes
    shortestdistance ← 0
    current ← keep track of next node to visit
    for i ← 0 to length of data-1 do
        for j ← 1 to length of data do
            temp ← euclideandistance(current, j)
            if j not in visited then
                if temp ≤ shortestdistance then
                    shortestdistance ← temp
                    current ← j
                    visited ← j
▷ checking if capacity requirements are met
                    if current total capacity ≤ max capacity then
                        capacity ← capacity of current node
                    else
                        Add starting point to end of route and
                        add capacity of current node to variable(capacity)
        shortestdistance ← 0                                ▷ reset shortestdistance
    return solution
```

3 First Custom Local Search Algorithms

Having executed the heuristics, providing us with solutions to the initial problem instances, we can continue optimizing. This is where the Local Search Optimization algorithms comes into play. Given a canonical solution, we want to optimize it to reduce the total cost of the solution. We have provided three custom Local Search (LS) algorithms, trying to optimize the canonical solution. All of our localsearches optimize solution by make exchanges of visiting node between 2 routes in solution set. Thus they can be used in combination with original Traveling salesman problem's localsearches such as 2-opt and 3-opt, where improvements are done by swapping and reversing order of visiting nodes in single route.
The first LS algorithm looks into the generated routes done by the heuristics. It takes two routes and compare them with each other, checking if any swaps are possible in the two routes. A swap is possible if the distance is being reduced in at least one route, while still preserving the maximum capacity limit of both routes. The second LS algorithm looks like the first but with a minor change. It takes one route and compares it with every other route to find a better solution. It then continues doing this with every other route.

4 Performance Analysis & Boxplots

5 Process Analysis

Appendices

A

Custom Cluster Algorithm to solve CVRP

Algorithm 2 Cluster CVRP Localsearch

Require: *closest_point*$(A, points)$: returns point in points which hase minimum distance from A

Require: *elim_empty_routes*: returns list of routes where the empty routes are removed

Ensure: *routes*: Better Solution to the CVRP instance

```
function ALGORITHM(routes, maxcap)
    route_convexes is an array storing convex_hull for each route
    improvements ← True
    while improvements do
        for all i, j such that i, j are routes index combination do
            central_point_i ← average of point in convex i
            central_point_j ← average of point in convex j
            closest_from_i_j ← closest_point(mid_point_i, convex_routes[j])
            closest_from_j_i ← closest_point(mid_point_j, convex_routes[i])
            if closest_from_i_j and closest_from_j_i is Found then
                if closest_from_i_j fits in route j then
                    add move to route improvements j
                    make the swap
                else if closest_from_j_i fits in route i then
                    add move to route improvements i
                    make the swap
                else if capacity is not reached by swapping customers then
                    swap the route-points between i and j
                if improvements then
                    recalculate convex hull of route i and j
        if not improvements then
            routes = elim_empty_routes(routes) return routes
```

B

LocalSearch Algorithm - Swap if any improvement

Algorithm 3 swap_if_improvement

Require: $dist(A, B)$ returns distance from point A to B

```
function SWAP_IF_IMPROVEMENT(route_i, route_j, j_to_i, i_to_j)
    A, B, C ← route_i[i_to_j-1], route_i[i_to_j],
            route_i[i_to_j+1 mod(%) len(route_i)]
    D, E, F ← route_j[j_to_i-1], route_j[j_to_i],
            route_j[j_to_i+1 mod(%) len(route_j)]
    old_distance_i ← dist(A, B) + dist(B, C)
    old_distance_j ← dist(D, E) + dist(E, F)
    new_distance_i ← dist(A, E) + dist(E, C)
    new_distance_j ← dist(D, B) + dist(B, F)
    if old_distance_i + old_distance_j >
            new_distance_i + new_distance_j then
            swap point B of route i with point E of route j
            return True
```

C

LocalSearch Algorithm - Move if any improvement

Algorithm 4 move_if_improvement

Require: $dist(A, B)$ returns distance from point A to B

```
function MOVE_IF_IMPROVEMENT(route_i, route_j, i_to_j, j_to_i)
    A, B, C ← route_i[i_to_j -1], route_i[i_to_j],
          route_i[i_to_j+1 % len(route_i)]
    D, E, F ← route_j[j_to_i -1], route_j[j_to_i],
          route_j[j_to_i+1 % len(route_j)]
    old_distance_i ← dist(A, B) + dist(B, C)
    old_distance_j ← dist(D, E) + dist(E, F)
    new_dist_before_B ← dist(A, E) + dist(E, B) + dist(B, C)
    new_dist_after_B ← dist(A, B) + dist(B, E) + dist(E, C)
    new_distance_j ← dist(D, F)
    if new_dist_before_B < new_dist_after_B then
        if old_distance_i + old_distance_j >
          new_dist_before_B + new_distance_j then
          move point E from route j to route i before B
    else
        if old_distance_i + old_distance_j >
          new_dist_after_B + new_distance_j then
          move point E from route j to route i after B
```

D

BOXPLOT FIGURES