Master In Computational and Mathematical Engineering

Final Master Project (FMP)

Metaheuristic Algorithms for solving the Multi-Depot Arc Routing Problem

Name of the Student: Patricio Page Carro
Area of the FMP: Modelización y Simulación

Name of the Tutor: Jesica de Armas Adrián
Name of the Professor in Charge of the Subject: Angel Alejandro Juan Pérez

Date of Delivery: 18/06/2017

Alternative licences (choose any of the following and substitute the one of the previous page)

To) Creative Commons:

This work is subject to a licence of Attribution-NonCommercial-NoDerivs 3.0 of Creative Commons

This work is subject to a licence of Attribution-NonCommercial-ShareAlike 3.0 of Creative Commons

This work is subject to a licence of Attribution-NonCommercial 3.0 of Creative Commons

This work is subject to a licence of Attribution- NoDerivs 3.0 of Creative Commons

This work is subject to a licence of Attribution-ShareAlike 3.0 of Creative Commons

This work is subject to a licence of Attribution 3.0 of Creative Commons

B) GNU Free Documentation License (GNU FDL)

Copyright © YEAR YOUR-NAME.

Permission is granted to copy, distribute and/*or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no **Back-Cover Texts.
To copy of the license is included in the section **entitled "GNU Free Documentation License".

C) Copyright

© (The author/to)
Reserved all the rights. It is forbidden the total or partial reproduction of this work by any half or procedure, comprised the impression, the reprography, the microfilm, the computer treatment or any another system, as well as the distribution of copies by means of rent and loan, without the permission written of the author or of the limits that authorise the Law of Copyright.

INDEX CARD OF THE FINAL MASTER PROJECT

Title of the FMP:	*Metaheuristic Algorithms for solving the Multi-Depot Arc Routing Problem*
Name of the author:	*Patricio Page Carro*
Name of the TUTOR:	*Jesica de Armas Adrián*
Name of the PRA:	*Angel Alejandro Juan Pérez*
Date of delivery (mm/aaaa):	06/2017
Degree:	*Master In Computational and Mathematical Engineering*
Area of the Final Work:	*Modelización y Simulación*
Language of the work:	*English*
Keywords	*Arc routing problem, randomized algorithms, heuristics*

Summary of the Work (maximum 250 words): *With the purpose, context of application, methodology, results and conclusions of the work.*

The main objective of the present work is to elaborate the most effective algorithm for solving the Multi-Depot Arc Routing Problem (MDARP), taking the Randomized Sharp as base algorithm and starting point, and particularly to study different alternatives for developing the edge-to-depot assignment. Concrete applications of this problem are garbage collection, electricity meter reading, mail distribution and door-to-door selling. To accomplish this several edge-to-depot allocation strategies in conjunction with variations on the Randomized Sharp algorithm were implemented in the Java language and tested against one another and using the existing benchmarks for this problem.

The results show that assigning edges to depots using a biased-randomized strategy offers the best results. Also the present work's algorithm, which combines the Randomized Sharp algorithm with a splitting search, simulated annealing and a cache strategy gives competitive results compared to current benchmarks.

Abstract (in English, 250 words or less):

The Multi-depot Arc Routing Problem (MDARP) is a combinatorial optimization problem belonging to a family of related problems that have in common the objective of finding the optimal route for a vehicle or a fleet of vehicles in order to satisfy demand located at the nodes or along the edges of a graph. When the demand is located at nodes it is called a Vehicle Routing Problem (VRP) and when it is located along the edges it is called Arc Routing Problem (ARP). For the present work, the ARP problem is studied, enriched by having multiple starting and finishing nodes, called depots. This problem is known in literature as Multi-depot Arc Routing Problem (MDARP). The aim of the present work is to study algorithms for the solution of the MDARP and some of its variants using as base the Randomized SHARP algorithm from González et al. (2012). This base algorithm is a randomized Clarke & Wright Savings heuristic (Clarke and Wright (1964)) for the construction of the solutions. Several strategies for the allocation of edges to each available depot were studied and compared in their results and efficiency.

According to the results, the assignment of edges to depots using a biased-randomized strategy combined with the Randomized Sharp algorithm, a splitting search, simulated annealing and a cache strategy gives competitive results compared to current benchmarks.

Index

1. Introduction......1
  1.1 Context and justification of the Work......1
  1.2 Aims of the Work......1
  1.3 Approach and method followed......1
  1.4 Planning of the Work......1
  1.5 Brief summary of products obtained......1
  1.6 Brief description of the others chapters of the memory......1
2. Brief description of the problem......2
3. Literature Review......4
4. Present Approach......4
5. Computational Results......9
6. Conclusions......20
7. Glossary......22
8. Bibliography......23

List of figures

Figure 1: MDARP representation

Figure 2: Splitting procedure

Figure 3: gdb6

Figure 4: gdb8

Figure 5: gdb9

Figure 6: gdb11

Figure 7: gdb12

Figure 8: gdb13

Figure 9: gdb21

Figure 10: gdb22

Figure 11: All Instances

Figure 12: Accepting only solutions improving best solution

Figure 13: Accepting solutions improving base solution

1. Introduction

1.1 Context and justification of the Work

The ARP family of problems has not been studied as exhaustively as Vehicle Routing Problems (VRP). Particulary the multi-depot versions have significantly less bibliography. The purpose of the present work is to contribute by comparing several strategies for the allocation of edges-to-depots and present a competitive algorithm which can be used to explore further variations of the problem.

1.2 Aims of the Work

- Study the effectiveness of different edge-to-depot allocation strategies for the MDARP.
- Develop an algorithm based on the most effective strategy identified.
- Compare the final algorithm with the current benchmarks.

1.3 Approach and method followed

As a starting point, the state of the art algorithms for the ARP and MDARP are reviewed. This includes similar problems that serve as a good starting point for the MDARP, such as the VRP and Multi-depot VRP. The main objective of this initial phase is to develop an understanding of the use of the CWS heuristic, its randomized variation, and the various frameworks in which they work, including multi-start, ILS, tabu-search and cache schemes. Also, the main strategies for node allocation to depots are reviewed.

Next, several node allocation strategies in conjunction with the Randomized Sharp algorithm are implemented and tested against one another and using the existing benchmarks for this problem. This serves the purpose of increasing the understanding of the way each strategy impacts the end result, and their strengths and weaknesses. The next step is to develop different modifications of the existing strategies and methodologies for node allocation to depots and route generation. The implementation of these variations of the main Randomized Sharp algorithm are done using the Java language due to the ease of modelling the language provides and its widespread use.

Having implemented several different strategies and variations on the Randomized Sharp algorithm, they are tested using the problem's benchmarks to determine the quality of the solutions each of them provides. This is performed simultaneously with some parameter-tweaking worthy of studying.

Along with the optimality of the solutions, the time cost is taken into account, not discarding strategies solely based on time performance, but including it into the final considerations of the global performance of each of the strategies.

Finally, conclusions are extracted regarding the effect of applying the various node allocation strategies, as well as variations of the Randomized Sharp algorithm and the impact of the tweaking of the parameters. Also the efficiency of the studied algorithms is analyzed. To conclude, paths for future investigations are proposed.

1.4 Planning of the Work

Task	Days	Starting date	Finishing date
End of Master Paper Realization	187	5-Dec-2016	10-Jun-2017
Work Plan preparation	16	5-Dec-2016	21-Dec-2016
Literature revision	16	5-Dec-2016	5-Jan-2016
Study of the Randomized Sharp algorithm	31	5-Dec-2016	5-Jan-2017
Formulation of improvement strategies	13	2-Jan-2017	15-Jan-2017
Implementation of improvement strategies for the algorithm	59	16-Jan-2017	16-Mar-2017
Comparative analysis of improvement strategies for the algorithm	31	17-Mar-2017	17-Apr-2017
Elaboration of conclusions	17	18-Apr-2017	5-May-2017
Composition of the preliminary report	15	6-May-2017	21-May-2017
Revision of the paper	10	22-May-2017	1-Jun-2017
Composition of the final report	8	2-Jun-2017	10-Jun-2017

1.5 Brief summary of products obtained

An algorithm for the MDARP was developed, which offers competitive results compared to benchmarks and proves to be a good starting point to explore richer versions of the MDARP.

1.6 Brief description of the others chapters of the memory

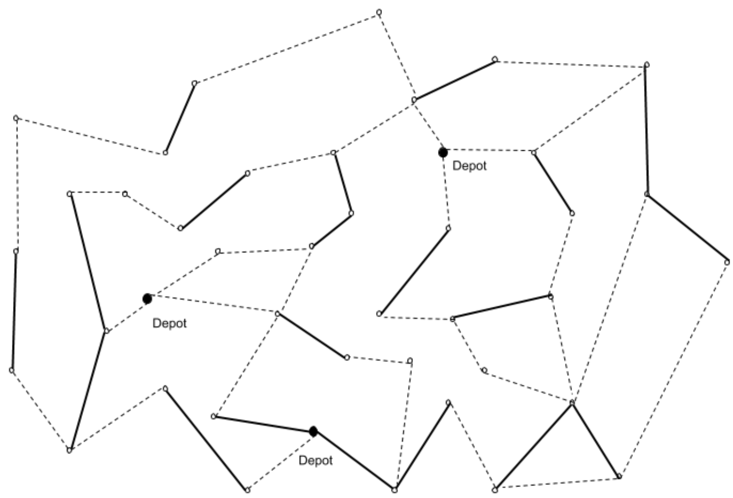
The article is structured as follows: Chapter 2 gives a brief introduction to the MDARP problem, Chapter 3 highlights some related works on the ARP and its variants. Details and implementation of the solutions analyzed in this article are given in Chapter 4. The experiments carried out and their results are described in Chapter 5. Lastly, Chapter 6 points out the key aspects of this paper and identifies the possibilities for some future research lines.

2. Brief description of the problem

This paper aims at exploring various strategies for solving the Multi-Depot Arc Routing Problem (MDARP). In this problem, there is a graph G = (N,E) (where N is the number of nodes and E the number of edges) to be traversed in any number of different routes which start and end in one of the depots. Some of this graph's edges are required to be part of a route and some are not required. Concrete problems that could be modeled this way are garbage collection, electricity meter reading, mail distribution and door-to-door selling Assad and Golden (1995)[2], Dror (2000)[5].

In Figure 1 an MDARP graph is presented, with its nodes, depot nodes and edges. The bold edges represent edges that are required and contain demand to be serviced by routes beginning and ending in one of the depots. The dashed edges are not required but may or may not be needed as part of one of the routes. The problem then is how to construct routes to service each of the required edges with the least cost, each route beginning and ending in the same depot.

Figure 1: MDARP representation



Differently from a single-depot ARP, where the problem consists in finding the best route to serve all required edges, when approaching a MDARP, a previous phase can be identified. This first phase is related to the problem of determining which depots will be serving each one of the required edges, referred to as edge allocation. This part of the problem will output as a result a submap for each depot, meaning a subset of edges from the full graph. The second phase is involved in determining the routes through the required edges for each of the depots. In this work the focus is placed mainly on the first phase, the edge allocation, for which many strategies for generating submaps are explored. Also different schemes for applying the Randomized SHARP are analyzed: the use of a cache strategy and a simulated annealing approach combined with a local search procedure. Different combinations of these strategies are tested and compared regarding the minimum and average values obtained.

3. Literature review

The amount of literature devoted to the ARP is significantly lower than that dedicated to the Vehicle Routing Problem. Nevertheless, many parallelisms can be drawn between the two types of problems and what serves as good literature for one might prove valuable for the other.
The ARP might have begun with Leonhard Euler's solution to the Königsberg bridges problem (Sachs et al. (1988)). In this problem, a connected graph G = (N,E) is given and the task is to find a closed tour that visits every edge in the graph exactly once or prove that no such tour exists. Such tours, if found, are known as Euler tours. Two algorithms were presented some years later for constructing the Euler Tour, the first one by C. Hierholzer (Hierholzer (1873)[11]) and another version, less efficient, by M. Fleury (Fleury (1883)[8]). Another famous ARP is the Chinese Postman Problem, posed by Kwan Mei-Ko (Mei-Ko (1962)[17]). It is similar to Euler's problem: Given a connected graph G = (N,E,C), where N are the nodes of the graph, E are the edges and C is a distance matrix, find a tour that traverses every edge in the graph, but does so in the least amount of time. Assad and Golden (1995)[2] state the basic methodology for solving generic ARPs, and describe several application areas. Similarly, Eiselt et al, write two papers (Eiselt et al. (1995a)[6], Eiselt et al. (1995b)[7]) to review the algorithmic methods for solving the chinese postman problem. There exist other surveys on the various methods for solving the ARP such as Dror (2000)[5], Wohlk (2008)[20], this last one more focused on the capacitated version of the ARP. Another survey of methods was published in Corberán and Prins (2010)[4] in which two important versions of the problem are discussed: the standard ARP and the capacitated ARP (CARP), in which an additional constraint is imposed on the ARP: the routes serving edges with demand have a limited capacity to satisfy that demand.
Metaheuristic approaches have been explored, some of which are used in the present work as well. For instance, the use of simulated annealing techniques has been applied to the ARP family of problems such as in Wohlk (2005)[19] and Amberg et al. (2000)[1], the latter of which also a tabu search is tested.
Many evolutionary approaches have been used for the MDARP and CARP as well, such as Hongtao et al. (2013)[12], Tiantang et al. (2014)[18], Xing et al. (2009)[21] and Kansou (2010)[14].
Finally, some Ant Colony Optimization algorithms have been used in Kansou and Yassine (2009)[15] and Kansou and Yassine (2012)[16]. The present article is strongly based on the SHARP algorithm presented in González et al. (2012)[10] which makes use of the Clarke & Wright savings heuristic from Clarke and Wright (1964)[3]. This heuristic has been succesfully applied to Vehicle Routing Problems and in their paper, González et al present a framework for applying the CWS heuristic to the ARP, and also present a biased randomized version for use in multistart algorithm.
Regarding the approach to the Multi-Depot version of the problem, the paper by Juan et al. (2014)[13], provides a good framework for the VRP, particularly for the allocation of nodes to each depot. The mentioned work provides valuable ideas and methods that can be translated into the MDARP.

4. Present approach

As stated in a previous section, the MDARP problems can be divided into an edge allocation problem and a simpler ARP problem. The first phase produces a submap for every depot in the graph, that is it establishes a relationship of "belonging" of every required edge to a depot. The second phase of solving each of these submaps using the Randomized SHARP algorithm in conjunction with other techniques will determine the most successful of these allocation strategies. This work first tries to select the best edge allocation strategy in this way, and subsequently different combinations of techniques for solving the submaps will be compared as well.

4.1 Edge Allocation Strategies

Edge allocation strategies can be divided into two groups: the savings-based strategies and strategies not based in the concept of savings. For the first group it is necessary to elaborate on the concept of savings as it is applied to edge allocation, since it differs slightly from the concept presented in Clarke and Wright (1964)[3]. In the most common sense, what is referred to as "saving" associated to an edge is how much cost is prevented if that edge is traversed, as opposed to returning to the depot from that edge's starting node and then travelling again from the depot to the edge's finishing node. In the case of edge allocation, we can see savings in the following way. When an edge is assigned to a depot, there is a certain cost of travelling from the depot to the edge's starting node, plus the cost of traversing the edge, plus the cost of returning to the depot from the edge's finishing node. For a particular edge there is going to be a different total cost depending on the depot to which it is assigned, therefore we can understand a saving associated to a depot-edge pair as the difference of cost between assigning that edge to that depot and assigning it to the closest of the remaining depots.

The following table briefly references each of the strategies tested in this work:

Table 1: Edge Allocation Strategies

Savings-Based strategies
Round-Robin
Round-Robin With Capacity
Random With Savings
Depot With Highest Saving
Edge-To-Edge Savings
Distance (or cost)-based
Edge Probability
Randomized Depot Distance
Depot And Edge Distance
Depot And Two Edges Distance
Depot And Edge Average Distance
Random Edge Distance
Two Random Edges Distance
Closest Edge In Submap
Not Savings nor Cost-Based
Random

4.1.1 Savings-Based strategies

Round-Robin: This strategy will select one depot at a time and assign an edge to it according to the savings of the edge for that depot, with some randomization given by a geometric distribution. This loop will continue assigning an edge to each depot at a time, until all edges have been assigned.

Round-Robin With Capacity: Similarly to the previous strategy, this one attempts to assign edges to depots one depot at a time. The difference between the two strategies is that this one will always assign an edge to the depot with the least amount of demand served so far in an attempt to achieve a more uniform distribution of loads among depots.

Random With Savings: Taking into account the savings for each depot-edge pair, this strategy assigns edges to depots one depot at a time, but every time a random depot is chosen among all the depots following a uniform distribution.

Depot With Highest Saving: In this case, for each edge the depot which produces the highest saving is determined and the edge is assigned to it.

Edge-To-Edge Savings: This strategy assigns the first edge of every submap according to its distance to the depot. Afterwards it iterates over every unassigned edge and calculates the savings caused by connecting that edge to every edge in each submap, finally the edge is assigned to the submap that contains the edge for which the savings are greater.

4.1.2 Distance (or cost)-based

Edge Probability: for this strategy, we first calculate the costs of assigning the edge to every depot and select the two closest least costly depots for this edge. Then a "probability" of assigning the edge to the closest depot is calculated. This is done by taking the cost of assigning the edge to the farthest depot of the two and dividing this cost by the sum of both costs. This number is then multiplied by a factor of 1.5 to increase the probability of assignment to the closest depot. During the assignment phase, for each edge, a random number with uniform distribution is obtained and if this number is less than the assignment probability of the edge, it is assigned to the closest depot, otherwise it is assigned to the second closest depot.

Randomized Depot Distance: This strategy first determines the closest depot to the nodes of an edge and assigns the edge to that depot. To determine the closest depot, the distance to each of the edge's nodes must be minimal. In the case of a depot having a closer distance to one of the nodes, this is labeled as "second closest" depot and the edge is assigned to one of these depots according to a uniform distribution. In the case that only one closest depot is found, the edge is assigned to this depot with 70% probability, the remaining 30% of the times the edge is assigned to any depot according to a uniform distribution.

Depot And Edge Distance: This strategy iterates over every edge and finds its distance to each of the depots and its distance to a randomly chosen edge already assigned to that depot. Both this distances are added, and this is done for every depot. Finally the edge is assigned to the submap for which this sum is minimal.

Depot And Two Edges Distance: Like the previous strategy, this one takes into account the distance of the edge to the depot and its distance to two random edges already assigned to that depot, assigning the edge to the submap for which the sum of these distances is minimal.

Depot And Edge Average Distance: Just like Depot And Edge Distance, with the difference that instead of taking the sum of the distances, it takes the average, and assigns the edge to the submap for which this average is minimal.

Random Edge Distance: This strategy iterates over all the edges and for each one it select a random, already assigned edge of each submap and calculates the distance between them, keeping the edge for which this distance is minimal. Finally, the edge is assigned to the same submap as this edge.

Two Random Edges Distance: Like the previous strategy, this one takes into account the distance to two edges already assigned to each submap.

Closest Edge In Submap: In this case for each edge that we need to assign, all of the currently assigned edges per submap are evaluated for distance. The edge is assigned to that submap which contains the edge that is closest to it.

Finally there is a strategy which is neither savings-based nor cost based:

Random: This strategy simply iterates over all of the edges and for each one it selects the depot with a uniform distribution.

4.2 General testing algorithm

For testing these allocation strategies an algorithm was used that combines a multistart procedure for generating several initial solutions based on the Randomized SHARP algorithm with a simulated annealing scheme which utilizes a splitting procedure and a cache of best known routes.
In Algorithms 1, 2 and 3 the main algorithm and it's most important parts are detailed.

Algorithm 1: Main algorithm

```
Map ← assignEdgesToDepots(strategy, graph)
SolPool ← solution pool of size five
Cache ← cache of best solutions
NIter ← number of iterations
while iterations < NIter do
  sol ← MDRandSHARP(Map, Cache)
  if sol.cost < highestCostInSolPool then
    addSolToPool(sol, SolPool)
    removeHighestCostSolutionFromPool
    if sol.cost < lowestCostInSolPool then
      BestOverallSol ← sol
    end if
  end if
end while
ForceImprovement ← false
for all Sol in SolPool do
  BaseSol ← splitSolution(Sol, Cache, SplitIter, ForceImprovement)
  BestSol ← BaseSol
  temperature ← Initial Temperature
  while elapsedTime < maxTime seconds and temperature > 0 do
    temperature ← decreaseTemperature(temperature)
    NewSol ← splitSolution(BaseSol, Cache, Iter, ForceImprovement)
    Delta = NewSol.cost − BaseSol.cost
    if Delta < 0 then
      temperature ← decreaseTemperature(temperature)
      if Newsol.cost < BestSol.cost then
        BaseSol ← Newsol
        BestSol ← Newsol
      else if Random < exp(delta/t) then
        BaseSol ← Newsol
      end if
    end if
  end while
  BestSol ← improveEdgesorder(BestSol)
  if BestSol.cost < BestOverallSol then
    BestOverallSol ← BestSol
  end if
end for
```

Algorithm 2: MDRandSHARP(Map, Cache)

```
for all Submap in Map do
  SubSol ← randSHARP(Submap, Cache)
  appendSubSolToSol(Subsol, Sol)
end for
```

Algorithm 3: splitSolution(Sol, Cache, SplitIter, ForceImprovement)

```
1: if ForceImprovement = true then
2:    BestSol ← Sol
3: else
4:    BestSol ← null
5: end if
6: improvements ← 0
7: BaseSol ← sol
8: while improvements < SplitIter do
9:    improvements ← improvements + 1
10:   partialRoute, remainingRoute ← removeRandomEdgesFromSol(BaseSol)
11:   for i = 0 to sharpIterations do
12:      newPartialRoute ← MDRandSHARP(partialRoute, Cache)
13:      if newPartialRoute.cost < partialRoute.cost then
14:         partialRoute ← newPartialRoute
15:      end if
16:   end for
17:   newSol ← mergeSolutions(partialRoute, remainingRoute)
18:   newSol ← improveWithCache(newSol, Cache)
19:   if BestSol = null or newSol.cost < bestSol.cost then
20:      BaseSol ← newsol
21:      BestSol ← newsol
22:   end if
23: end while
24: return Bestsol
```

4.2.1 First Phase: Multistart Algorithm

After assigning the edges to the depots according to the selected allocation strategy (Algorithm 1, line 1), the algorithm begins an initial multi-start procedure (Algorithm 1, line 5) that takes advantage of a randomized version of the SHARP algorithm González et al. (2012)[10] for multiple depots to generate many different solutions (Algorithm 1, line 6). This multi-depot version of the Randomized SHARP is succinctly detailed in Algorithm 2, where for each submap in the graph, the Randomized SHARP procedure is applied.

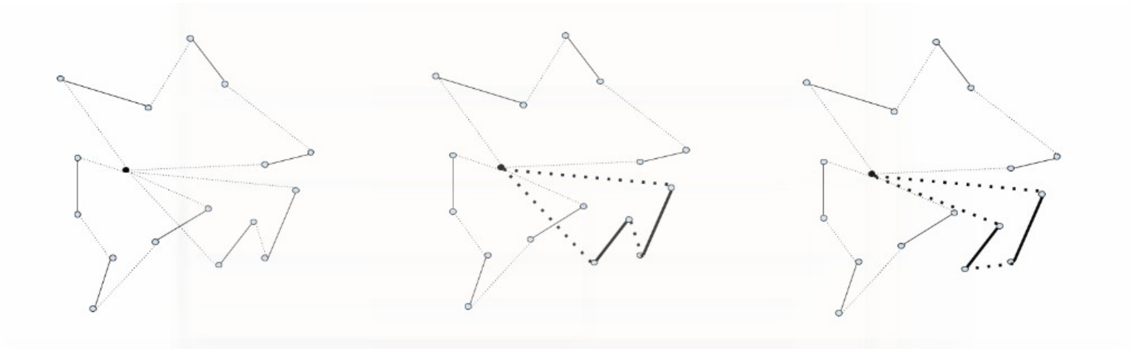
Briefly, the SHARP procedure ranks the edges in a graph according to the savings produced by traversing the edge with a single vehicle instead of visiting its nodes in two different routes. In order to construct a full route, one could simply choose those edges with highest savings and start joining them to form routes with high savings. This process, when done in a Capacitated Vehicle Routing Problem (CVRP) is known as the Clarke & Wright Savings heuristics, widely recognized to be the best heuristic for solving the CVRP. The limitation of using this heuristic is that the resulting solution is always the same. By using a guided randomization process we can obtain several different solutions, many of which might improve the original CWS solution. This ”guided randomization” consists in not simply selecting the highest saving edge when constructing a solution, but randomly select the edge following a geometric distribution. This results in the best edges being selected with higher probability, but allowing for some ”not so good” edges to be selected at times. After running this process for a number of iterations (NIter), (Algorithm 1, line 5) the best five of these solutions are kept in a ”solution pool” (Algorithm 1, lines 7-13) and a second search phase is applied to each of them (Algorithm 1, line 16-38).

4.2.2 Second Phase: Local Splitting Search with Simulated Annealing

For each solution in the solution pool generated in the previous phase, the solutions are split into the routes that compose them. Then each route in the solution is split (Algorithm 1, line 17). This means the route has a random number of routes extracted from it (Algorithm 2, line 10). The route that has been extracted is solved again iteratively using the Randomized SHARP procedure to obtain a new route (Algorithm 2, line 12). This splitting and searching is repeated until no improvements have been obtained after a number of iterations equal to splitIter. After the final route is obtained, it is merged back with the remaining routes (Algorithm 2, line 17). Using this final route, the cache is searched to attempt to improve the solution (Algorithm 2, line 18). If the final solution improves the best known solution, the new solution is accepted as best solution (Algorithm 2, line 19-22).

Figure 2 illustrates this splitting procedure: starting from a graph with three routes, one of them is selected, and solved through the Randomized SHARP algorithm, before being merged back into the solution.

Figure 2: Splitting procedure



The splitting procedure can be set to enforce improvements or not. In the case where improvements are enforced the best known solution is set to the initial solution (Algorithm 2, line 1-2), so that the newly obtained solutions will only be accepted if they improve the initial solution. Otherwise, if the procedure is set not to enforce improvements, the best known solution is set to null (Algorithm 2, line 4), which results in accepting the best of the generated solutions whether it improves the initial solution or not. This results in a starting solution for a simulated annealing search (Algorithm 1, line 19-33), which runs until the elapsed time reaches the maximum time set or the temperature parameter reaches zero. In this iterative process the solution is split again (Algorithm 1, line 22) and accepted according to a simulated annealing-based acceptance criterion (Algorithm 1, line 23-32). At each iteration of the process, temperature is decreased(Algorithm 1, line 21), and the difference of cost between the new solution and the current solution evaluated as Delta (Algorithm 1, line 23-24). If Delta is less than zero, temperature is again decreased (Algorithm 1, line 24-25), if the cost of new solution is less than that of the best known solution, both the best known solution and the best known solution are updated with the new value (Algorithm 1, line 26-28), otherwise if a uniformly random number is less than e (delta/t) , the base solution is updated with the new solution, the best known solution remains unchanged(Algorithm 1, line 29-31).

With the solution obtained from this process, the order of the edges is analyzed in search for ”knots” (Algorithm 1, line 34). This means in practice that every three consecutive edges in a route, a different ordering is analyzed and if the cost diminishes in any other ordering, the solution is updated with this new order.

This process is repeated for every solution in the pool and the best solution is kept as a result(Algorithm 1, line 35-37).

4.2.3 Use of a memory cache

At all moments during these search, a cache of best found routes for servicing the edges with demand is kept and constantly updated with improving routes. This cache is perused throughout the algorithm (Algorithm 1, line 6, 17, 22; Algorithm 2, line 2; Algorithm 3, line 12, 18), always comparing the present route with routes previously found for a given list of edges with demand.

5. Computational Results

This algorithm was coded in the Java language and tested on a Core i3 CPU @ 2.4GHz and 4GB RAM.

For the computational experiments, the gdb set proposed in Golden et al. (1983)[9] were used. These instances contain dense and sparse networks of small to medium size (from 10 to 50 edges). All of the edges contain required demand. In every instance the depots have been set to the first and last nodes of the graph.

During the multistart procedure, the number of iterations is set to 100.000 (NIter = 100.000; Algorithm

1, line 4), and the size of the solution pool is set to 5 (Algorithm 1, line 2). In the first splitting search, the search is performed until no improvements have been made for 10 iterations of the splitting procedure (SplitIter = 10; Algorithm 1, line 17), keeping the best solution. The simulated annealing search is performed until the elapsed time reaches 5 seconds (maxTime = 5), the following splitting searches are performed until reaching 30 non-improving iterations (Iter = 30; Algorithm 1, line 22), also keeping the best solution. Within the splitting procedure, the Randomized SHARP algorithm is executed on the extracted routes for 30 iterations (sharpIterations = 30; Algorithm 3, line 12). Regarding the simulated annealing parameters, the initial temperature is set to 15.000 (Algorithm 1, line 19). It is decreased in every iteration by a uniformly random amount between 0 and 10 (Algorithm 1, line 21), and when an improving solution is found the temperature is decreased by a uniformly random number between 0 and delta × 2 , delta being the difference of cost between the new solution and the old solution (Algorithm 1, line 25).

First set of experiments

In this first set of experiments, the edge allocation strategies are considered, using the base algorithm described previously. Each strategy is used to generate the edge allocation map for each gdb instance, with fifteen different runs associated with fifteen different seed numbers for the random number generator. Both the minimum cost attained by the strategy and the average cost are taken into account for deciding which is the optimal strategy.

Table 2: Compare Minimum Cost of Strategies

	Instance																							
Strategy	**gdb1**	**gdb2**	**gdb3**	**gdb4**	**gdb5**	**gdb6**	**gdb7**	**gdb8**	**gdb9**	**gdb10**	**gdb11**	**gdb12**	**gdb13**	**gdb14**	**gdb15**	**gdb16**	**gdb17**	**gdb18**	**gdb19**	**gdb20**	**gdb21**	**gdb22**	**gdb23**	**Total Cost**
Closest Edge In Submap	337	353	301	322	388	321	356	352	314	284	405	**430**	536	104	58	129	**91**	168	55	121	158	200	235	**6018**
Closest Edge In Submap With Probability	316	337	267	287	375	298	**325**	348	302	**275**	395	**430**	536	100	58	127	**91**	162	55	121	156	200	235	**5796**
Depot And Two Edges Distance	315	329	271	286	383	302	333	346	284	287	391	438	548	98	58	127	**91**	160	55	133	158	200	233	**5826**
Depot And Edge Average Distance	**300**	329	269	295	379	**285**	**325**	340	**283**	281	**387**	447	536	98	58	**125**	**91**	160	55	121	156	200	233	**5753**
Depot And Edge Distance	315	337	267	288	373	290	333	340	286	**275**	395	466	544	106	58	127	**91**	160	55	121	156	200	233	**5816**
Randomized Depot Distance	**300**	336	279	286	377	293	330	359	303	283	405	454	544	98	58	**125**	**91**	160	55	121	160	198	231	**5846**
Depot With Highest Saving	344	357	309	369	465	343	390	399	338	322	447	695	617	107	58	132	**91**	162	83	132	175	202	245	**6782**
Edge Probability	**300**	**321**	**259**	**266**	**361**	296	**325**	**334**	286	283	**387**	447	536	98	58	**125**	**91**	162	55	121	156	198	233	**5698**
Edge-To-Edge Savings	337	342	305	310	435	321	366	372	345	292	439	525	582	115	62	133	93	181	71	131	168	208	241	**6374**
Two Random Edges Distance	315	329	271	286	370	304	326	346	307	277	**387**	450	550	98	58	**125**	**91**	160	55	121	158	197	235	**5816**
Random	308	337	285	**266**	386	301	326	380	322	284	411	466	558	**96**	58	127	**91**	160	55	130	158	200	233	**5938**
Random Edge Distance	315	329	271	**266**	**361**	300	**325**	343	287	**275**	391	444	**532**	98	58	**125**	**91**	158	55	121	159	198	235	**5737**
Random With Savings	306	337	281	272	400	307	331	377	317	287	433	517	552	98	**56**	**125**	**91**	160	55	130	158	198	234	**6022**
Round Robin	314	339	280	294	405	301	331	380	310	292	421	531	558	98	**56**	127	**91**	158	61	130	160	198	231	**6066**
Round Robin With Capacity	306	337	287	294	388	315	333	383	321	292	432	543	558	98	58	127	**91**	160	69	130	158	198	231	**6109**
Minimum Cost	300	321	259	266	361	285	325	334	283	275	387	430	532	96	56	125	91	158	55	121	156	197	231	5698

Table 3: Compare Average Cost of Strategies

	Instance																							
Strategy	**gdb1**	**gdb2**	**gdb3**	**gdb4**	**gdb5**	**gdb6**	**gdb7**	**gdb8**	**gdb9**	**gdb10**	**gdb11**	**gdb12**	**gdb13**	**gdb14**	**gdb15**	**gdb16**	**gdb17**	**gdb18**	**gdb19**	**gdb20**	**gdb21**	**gdb22**	**gdb23**	**Total Cost**
Closest Edge In Submap	337	353	301	322	388	321	356	354	317	284	405	**430**	**536**	104	**58**	129	**91**	168	**55**	**121**	158	200	235	**6023**
Closest Edge In Submap With Probability	316	348	277	299	381	302	329	348	305	**279**	399	**430**	**536**	100	**58**	**127**	**91**	163	**55**	**121**	**157**	200	235	**5857**
Depot And Two Edges Distance	316	342	279	292	395	308	339	351	292	**292**	402	460	556	100	59	130	**91**	164	62	136	163	201	237	**5965**
Depot And Edge Average Distance	**301**	335	277	295	379	304	**326**	346	293	284	392	448	547	**98**	**58**	128	**91**	**161**	**55**	**121**	**157**	201	235	**5833**
Depot And Edge Distance	316	350	287	310	380	**301**	337	344	291	**279**	400	476	554	107	**58**	129	**91**	163	**55**	128	159	200	236	**5951**
Randomized Depot Distance	316	345	290	300	389	313	337	372	313	298	419	483	556	105	59	128	**91**	165	56	129	164	201	237	**6064**
Depot With Highest Saving	344	357	309	369	465	343	390	400	340	322	447	695	617	107	**58**	132	**91**	162	83	132	175	202	247	**6787**
Edge Probability	310	**333**	**274**	**272**	**375**	307	332	**342**	**292**	284	**391**	454	**536**	101	**58**	**127**	**91**	163	**55**	**121**	160	200	236	**5815**
Edge-To-Edge Savings	337	371	306	329	435	321	368	372	356	306	439	572	598	115	62	133	93	182	71	131	168	208	242	**6515**
Two Random Edges Distance	316	339	274	304	378	311	333	359	319	305	393	474	561	102	**58**	128	**91**	171	55	130	162	200	239	**6002**
Random	319	351	293	300	412	320	342	399	334	301	426	532	572	106	60	129	**91**	167	68	136	164	203	238	**6265**
Random Edge Distance	316	343	275	273	380	307	333	350	303	281	400	452	539	100	**58**	128	**91**	162	**55**	**121**	164	200	237	**5868**
Random With Savings	323	350	291	317	425	320	341	392	328	309	441	551	571	103	**58**	128	**91**	162	62	136	163	200	237	**6301**
Round Robin	329	354	295	319	425	319	344	393	324	306	439	570	573	102	59	128	**91**	**161**	67	137	164	200	**234**	**6334**
Round Robin With Capacity	324	348	297	314	423	328	345	390	327	306	446	573	570	102	59	128	**91**	162	74	134	162	**199**	**234**	**6338**
Minimum Average Cost	301	333	274	272	375	301	326	342	292	279	391	430	536	98	58	127	91	161	55	121	157	199	234	5815

Using both minimum cost and average cost metrics the lowest cost is obtained through the Edge-probability strategy. Other strategies have come to results close to this, particularly ”Depot And Edge Average distance” and this strategies might become more relevant in different setting, like larger networks with more depots.

To better visualize the difference among strategies, a sample of all the instances were selected to be represented by box-plots in the following figures.

Figure 3: gdb6

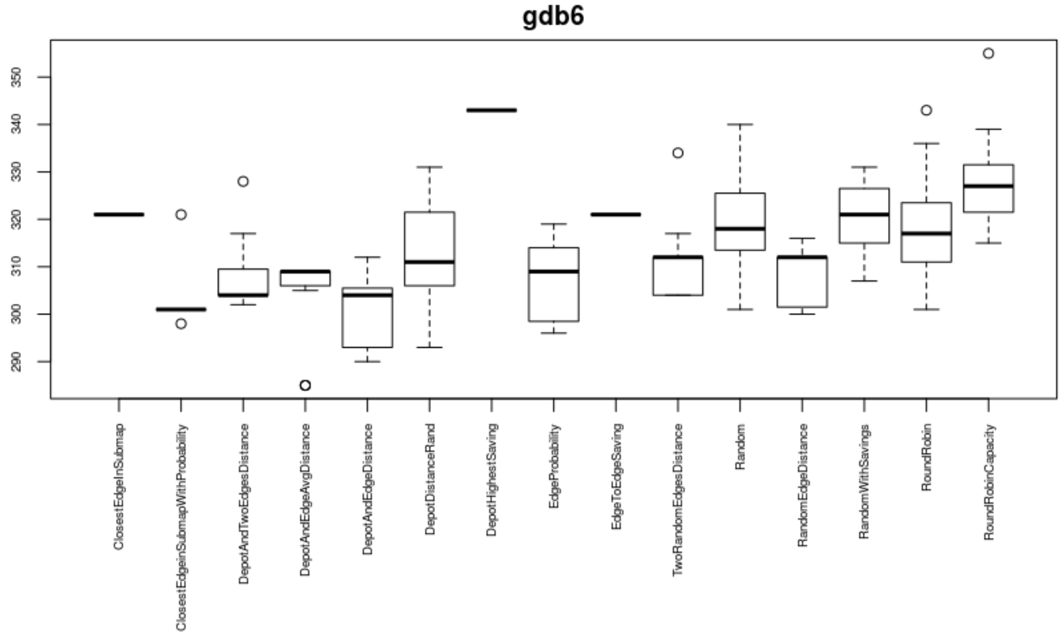


Figure 4: gdb8

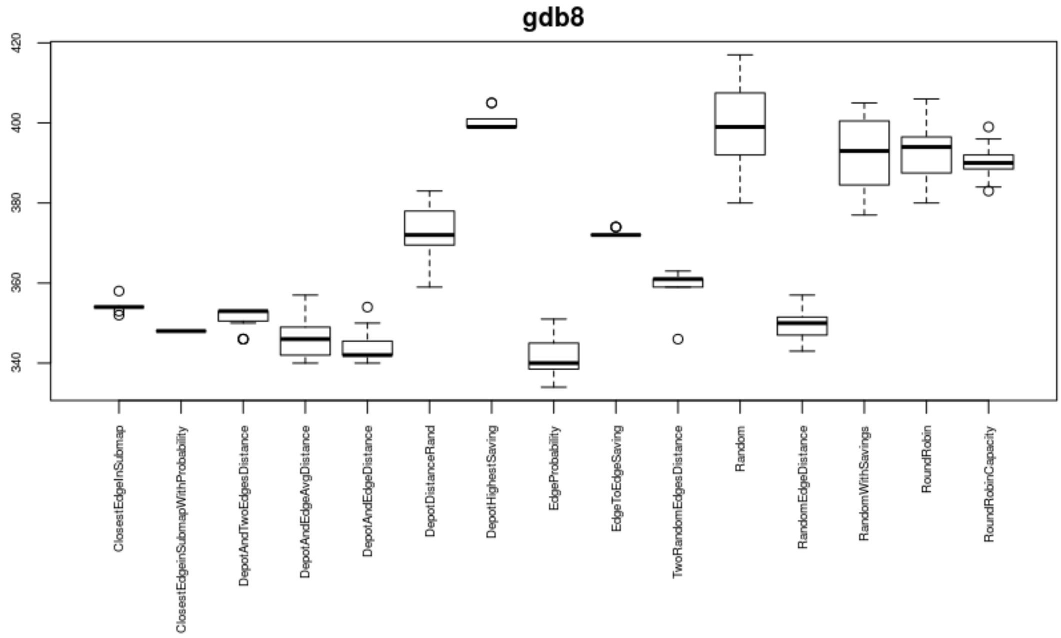


Figure 5: gdb9

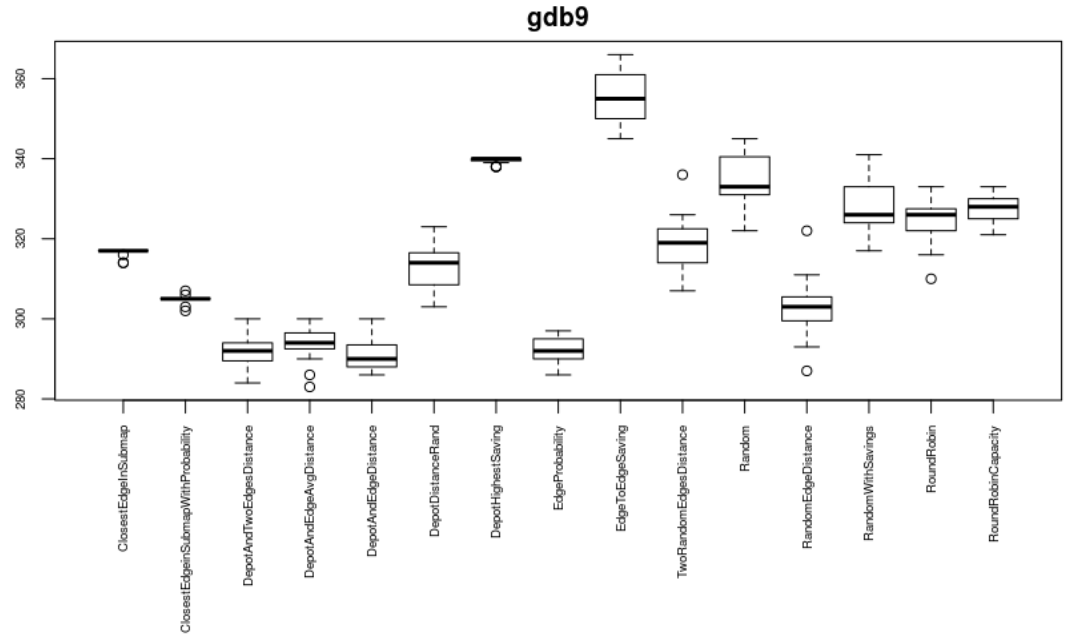


Figure 6: gdb11

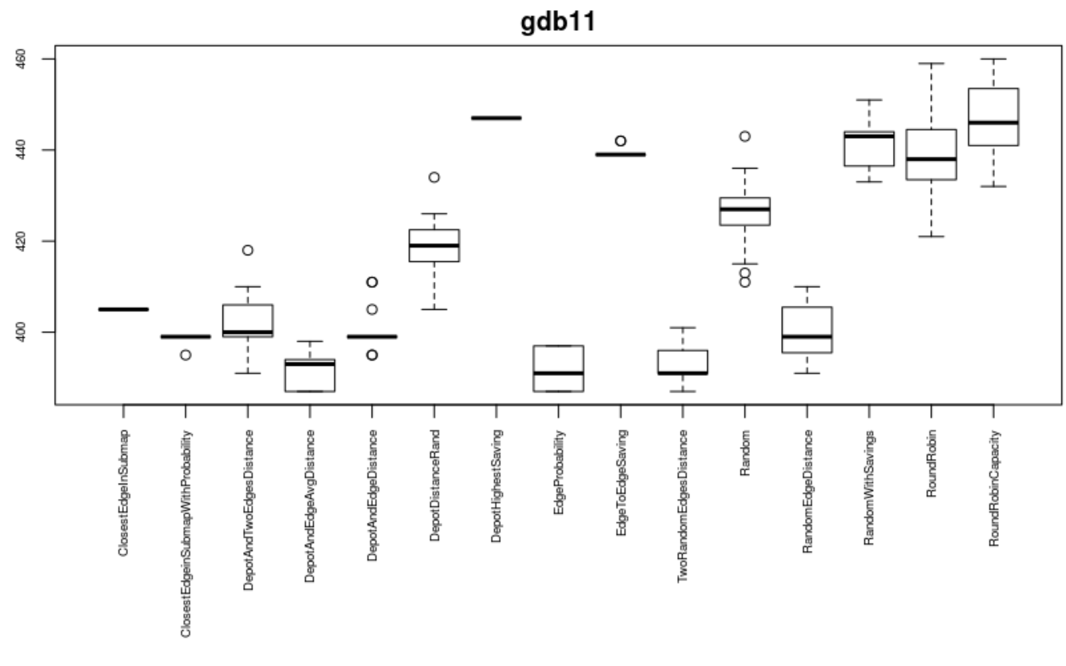


Figure 7: gdb12

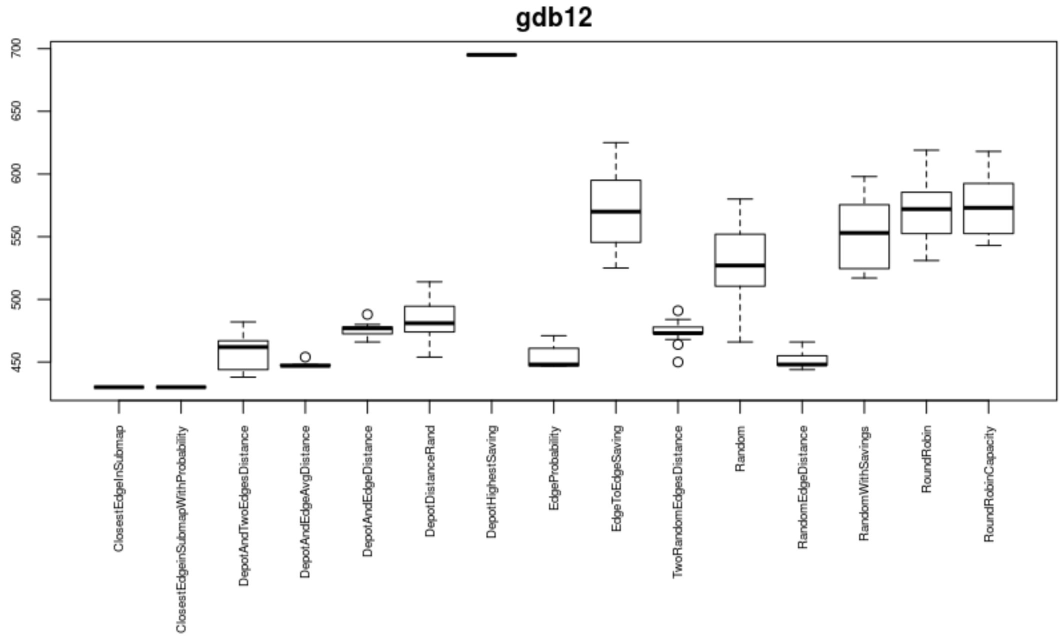
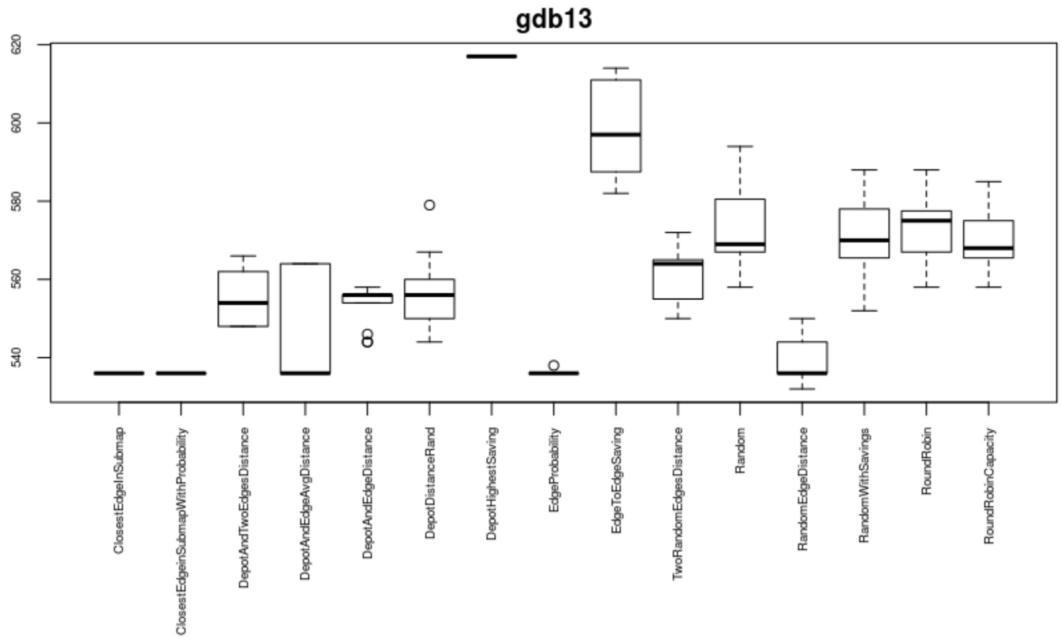


Figure 8: gdb13



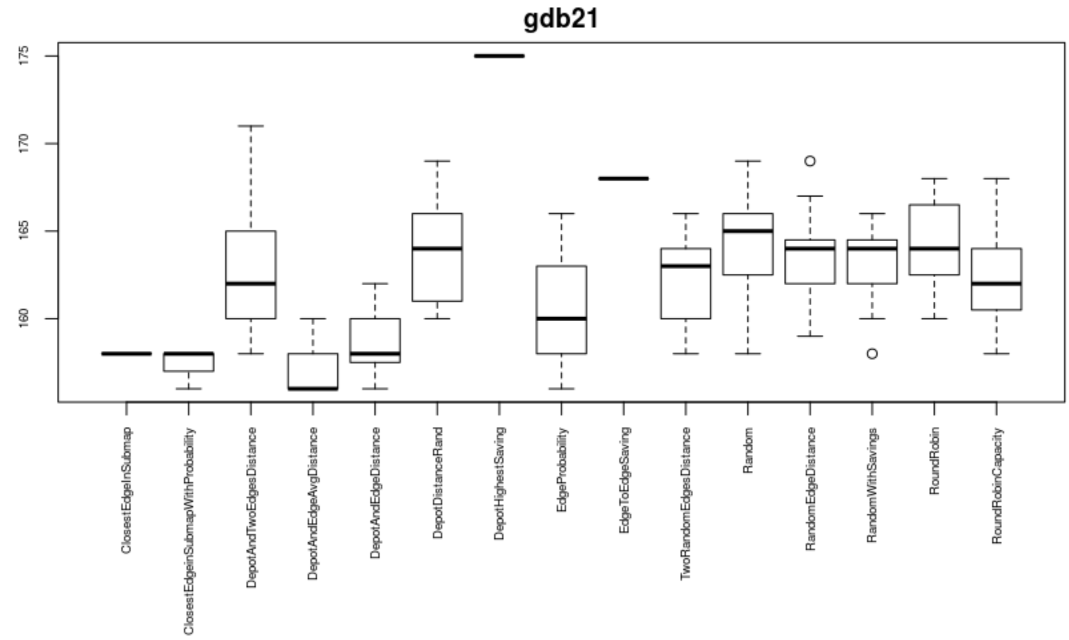
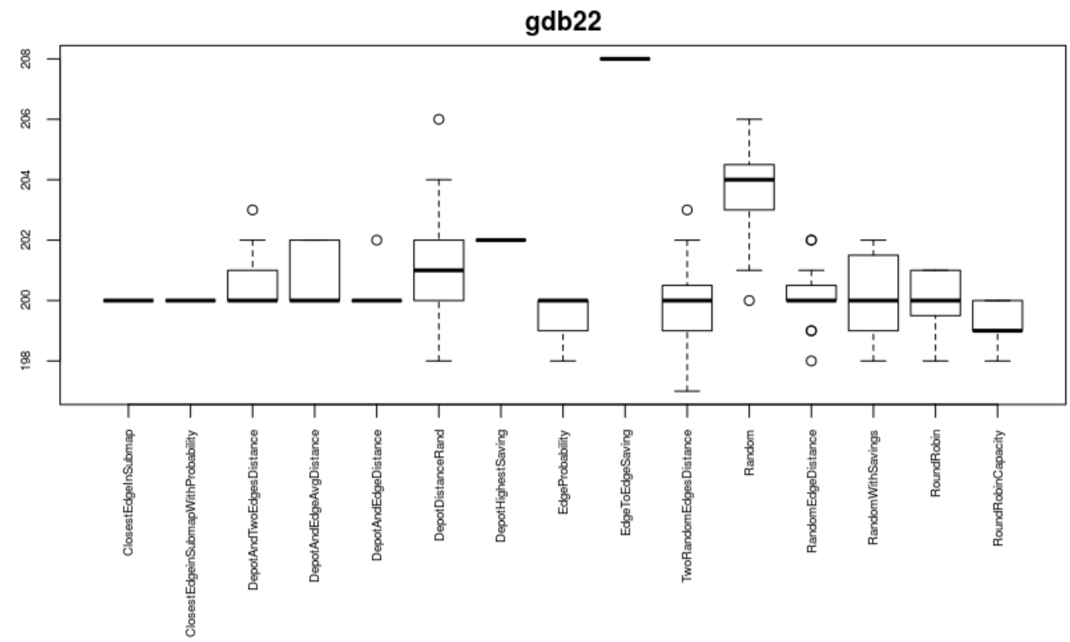
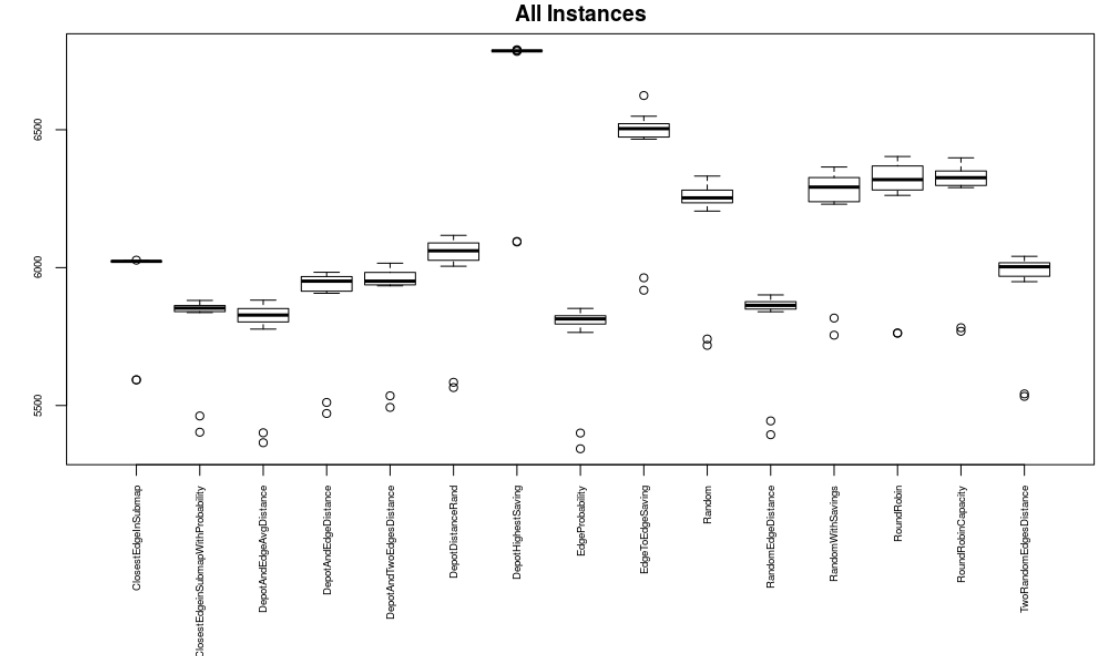


Figure 9: gdb21
Figure 10: gdb22



It is noticeable how for some instances one strategy greatly outperforms another. For example, in instance gdb6 the strategy ”DepotAndEdgeAvgDistance” is significantly better than ”EdgeProbability”. In gdb22, ”EdgeProbability” is clearly better between the two, but is again outperformed by ”ManyRandomEdges Distance”.
By analyzing the cost of each strategy to run an all instances, a broader conclusion can be made. The following chart shows this comparison.

Figure 11: All Instances



From this chart it is clearer that ”EdgeProbability” performs better on average, but some other strategies, like ”DepotAndEdgeAvgDistance”,”ClosestEdgeInSubmapWithProbability” and ”RandomEdgeDistance” seem to be very close both on average and on the minimum values obtained.
To analyze the statistical significance of the difference between these strategies, a multiple comparison was made using Friedman’s test in which each strategy is assigned a rank within each instance. Table 4 shows the statistics of the ranks for each strategy.

Table 4: Friedman's test Ranks

Strategy	Sum Of Ranks	Range	Std. Dev.	Min	Max
ClosestEdgeInSubmap	181.5	7.89	4.06	1.5	13.5
ClosestEdgeinSubmapWithProbability	92.5	4.02	1.86	1.5	8
DepotAndTwoEdgesDistance	168	7.30	2.89	2	13
DepotAndEdgeAvgDistance	80	3.48	2.27	1	11
DepotAndEdgeDistance	140.5	6.11	3.20	1	13
DepotDistanceRand	187.5	8.15	2.31	3	12
DepotHighestSaving	305.5	13.28	3.30	4	15
EdgeProbability	77	3.35	2.16	1	8.5
EdgeToEdgeSaving	313	13.61	1.62	9	15
TwoRandomEdgesDistance	158	6.87	3.61	1	14
Random	259.5	11.28	1.86	7	14
RandomEdgeDistance	115.5	5.02	2.16	2	10
RandomWithSavings	227.5	9.89	2.79	3	14
RoundRobin	234.5	10.20	3.64	1	15
RoundRobinCapacity	219.5	9.54	3.70	1	14

Through Friedman's test strategies are grouped. Each group is represented by a letter, each group contains strategies that are not significantly different. Strategies in different groups are significantly different. Table 5 shows the groups obtained.

Table 5: Friedman's Groups

Strategy	Sum of Ranks	Group
EdgeProbability	77.00	a
DepotAndEdgeAvgDistance	80.00	a
ClosestEdgeinSubmapWithProbability	92.50	a
RandomEdgeDistance	115.50	ab
DepotAndEdgeDistance	140.50	bc
TwoRandomEdgesDistance	158.00	cd
DepotAndTwoEdgesDistance	168.00	cd
ClosestEdgeInSubmap	181.50	de
DepotDistanceRand	187.50	de
RoundRobinCapacity	219.50	ef
RandomWithSavings	227.50	fg
RoundRobin	234.50	fg
Random	259.50	g
DepotHighestSaving	305.50	h
EdgeToEdgeSaving	313.00	h

We see that RandomEdgeDistance, ClosestEdgeinSubmapWithProbability, DepotAndEdgeAvgDistance and EdgeProbability are not significantly different between themselves. Nevertheless, since EdgeProbability is the lowest ranked strategy of the group, it is selected as the top strategy for the remainder of this work.

Second set of experiments

In this second set of experiments we maintain the "Edge Probability" strategy fixed and test different settings to the main algorithm. To reduce the amount of tests to run, instead of running the whole combinations of settings, they have been organized in "phases". In each phase, every combination of a reduced set of settings is studied and the optimal settings are kept for the subsequent phases.

In the first phase, we test the use of a randomized versus a greedy version of the SHARP algorithm, along with the use of the cache and the use of the solution pool. The solution pool setting can be: A) add every solution that improves the worst solution in the pool (in other words, keeping the top five solutions), B) add only solutions that improve the best solution in the pool or C) not use the pool at all.
The combination of these settings can be summarized in Table 6. It is clear that when the greedy version of the SHARP algorithm is used, there is no need of a pool of solutions, since every time the algorithm is run, the solution is going to be the same.

Table 6: First Phase

Settings	1-a	1-b	1-c	1-d	1-e	1-f	1-g	1-h
Use Randomized Solve	Yes	Yes	Yes	Yes	Yes	Yes	No	No
Use Cache	Yes	Yes	Yes	No	No	No	Yes	No
Pool (A/B/C)	A	B	C	A	B	C	C	C

The results of this tests are exposed in the following tables.

Table 7: Phase 1: Compare Minimum Cost of Settings

	Instance																							
Test	gdb1	gdb2	gdb3	gdb4	gdb5	gdb6	gdb7	gdb8	gdb9	gdb10	gdb11	gdb12	gdb13	gdb14	gdb15	gdb16	gdb17	gdb18	gdb19	gdb20	gdb21	gdb22	gdb23	Total Cost
1a	**300**	**321**	**259**	**266**	**361**	**285**	**325**	**334**	**286**	**281**	**387**	**447**	**528**	**96**	**56**	**125**	**91**	**162**	**55**	**121**	**156**	**198**	**233**	5673
1b	**300**	**321**	**259**	**266**	**361**	**285**	**325**	**334**	**286**	283	**387**	**447**	536	**96**	**56**	**125**	**91**	**162**	**55**	**121**	**156**	**198**	235	5685
1c	310	339	267	**266**	382	309	335	358	306	284	399	448	540	102	58	129	**91**	165	59	**121**	161	202	238	5869
1d	**300**	**321**	**259**	**266**	**361**	**285**	**325**	**334**	**286**	**281**	**387**	**447**	528	**96**	58	**125**	**91**	**162**	**55**	**121**	**156**	**198**	**233**	5675
1e	**300**	**321**	**259**	**266**	**361**	**285**	**325**	**334**	**286**	283	**387**	**447**	536	**96**	58	**125**	**91**	**162**	**55**	**121**	**156**	**198**	**233**	5685
1f	310	339	267	**266**	382	309	335	358	306	284	399	448	540	102	58	129	**91**	165	59	**121**	161	202	238	5869
1g	**300**	335	271	**266**	**361**	299	330	351	297	295	407	448	540	98	60	**125**	**91**	**162**	63	123	161	201	237	5821
1h	**300**	335	271	**266**	**361**	299	330	351	297	295	407	448	540	98	60	**125**	**91**	**162**	63	123	161	201	237	5821
Minimum Cost	**300**	**321**	**259**	**266**	**361**	**285**	**325**	**334**	**286**	**281**	**387**	**447**	**528**	**96**	**56**	**125**	**91**	**162**	**55**	**121**	**156**	**198**	**233**	**5673**

Table 8: Phase 1: Compare Average Cost of Settings

	Instance																							
Test	gdb1	gdb2	gdb3	gdb4	gdb5	gdb6	gdb7	gdb8	gdb9	gdb10	gdb11	gdb12	gdb13	gdb14	gdb15	gdb16	gdb17	gdb18	gdb19	gdb20	gdb21	gdb22	gdb23	Total cost
1a	310	**329**	**271**	**266**	**376**	**298**	**330**	340	**291**	**284**	**388**	**449**	**537**	**100**	**57**	**126**	**91**	**162**	**55**	**121**	**158**	**200**	**236**	**5774**
1b	312	334	274	**266**	**376**	305	332	**339**	292	**284**	389	450	**537**	101	**57**	127	**91**	163	**55**	**121**	**158**	**200**	**236**	**5801**
1c	329	356	292	275	395	323	349	366	316	295	414	477	557	108	59	136	92	168	60	123	169	205	245	**6109**
1d	310	**329**	**271**	**266**	**376**	**298**	**330**	340	**291**	**284**	**388**	**449**	**537**	**100**	58	**126**	**91**	**162**	**55**	**121**	**158**	199	**236**	**5776**
1e	312	334	274	**266**	**376**	305	332	340	292	**284**	389	450	**537**	101	58	127	**91**	163	**55**	**121**	**158**	**200**	**236**	**5802**
1f	329	356	292	275	395	323	350	366	316	295	414	478	557	108	59	136	92	168	60	123	169	205	245	**6111**
1g	**309**	341	273	273	382	312	337	373	306	298	413	453	557	102	60	127	**91**	164	63	123	164	202	241	**5965**
1h	**309**	341	273	273	382	312	338	373	306	298	413	453	557	102	60	127	**91**	164	63	123	164	202	241	**5965**
Minimum Average Cost	**309**	**329**	**271**	**266**	**376**	**298**	**330**	**339**	**291**	**284**	**388**	**449**	**537**	**100**	**57**	**126**	**91**	**162**	**55**	**121**	**158**	**199**	**236**	**5774**

Table 9: Phase 1: Compare Average Time of Settings

	Instance																							
Test	gdb1	gdb2	gdb3	gdb4	gdb5	gdb6	gdb7	gdb8	gdb9	gdb10	gdb11	gdb12	gdb13	gdb14	gdb15	gdb16	gdb17	gdb18	gdb19	gdb20	gdb21	gdb22	gdb23	Average Time
1a	16.8747	27.5899	24.8254	5.4292	22.8632	22.8204	23.482	25.6649	18.357	14.8767	18.4185	25.5037	11.6926	23.46	6.9286	24.8429	4.4087	14.1833	8.2526	9.4344	20.8743	17.6321	27.0283	**18.0627**
1b	14.1714	18.831	16.128	5.4162	16.1834	16.1458	16.8553	19.75	12.5112	14.2973	13.0675	26.8751	11.1661	12.7876	5.7833	19.5114	4.598	12.1931	8.4525	9.4433	22.235	18.9411	19.013	**14.5372**
1c	0.2344	0.0808	0.0581	0.053	0.0785	0.0393	0.0782	0.1381	0.1375	0.0556	0.1515	0.1687	0.065	0.0426	0.0284	0.0551	0.0894	0.1341	0.0471	0.0571	0.1198	0.107	0.1813	**0.0956**
1d	16.7273	27.2527	24.7029	5.3457	22.7044	22.6091	21.9895	24.7735	17.4458	14.7351	18.0816	24.7021	12.4358	22.6865	2.8305	26.0346	4.1282	14.0362	7.293	8.9451	20.5753	18.7149	25.4125	**17.5722**
1e	14.0459	19.991	15.9929	5.3463	15.8574	16.0301	16.7022	19.4483	12.1092	14.1435	12.7507	25.9271	10.4959	13.355	3.7701	15.3635	4.1412	12.0345	7.2891	8.9773	22.0096	18.058	18.7339	**14.0249**
1f	**0.0577**	0.0444	0.0172	0.0161	**0.0158**	**0.008**	0.012	0.0383	**0.0296**	**0.0072**	0.0294	0.0143	0.009	**0.0058**	0.0057	**0.0082**	0.0071	**0.0113**	0.0028	0.0043	0.0118	**0.0102**	**0.0211**	**0.0168**
1g	0.2655	0.1018	0.0768	0.0778	0.2468	0.2163	0.1404	0.1279	0.2501	0.1123	0.1182	0.1037	0.0951	0.1162	0.016	0.1354	0.063	0.1542	0.0009	0.0025	0.0917	0.1308	0.2759	**0.1269**
1h	0.0592	**0.0436**	**0.0168**	**0.0058**	0.0197	0.0097	**0.0111**	**0.0352**	0.0418	0.0124	**0.0172**	**0.0085**	**0.0081**	0.0094	**0.0017**	0.0092	**0.0068**	0.0174	**0.0003**	**0.0007**	**0.01**	0.0105	0.0215	**0.0163**

It is clear from the results exposed in the previous table that using the randomized version of the SHARP algorithm along with the cache and a pool of solutions accepting every solution improving the worst solution in the pool is the best combination of settings. It is also worthy to notice that the use of cache hasn't improved the solution very much, but the penalty in processing time for using it seems to be too small to discard its use. It is likely that for larger instances, the cache might gain much more relevance. As stated before, this settings are kept constant for the following phases.
In the second phase we test the use of the "unknotting" of the routes by the use of the function "ImproveEdgesOrder". The options are simple, either use the function or not use it. For that reason there

is no need for a table to detail the tests that were performed.
In the following tables the results from this phase are exposed.

Table 10: Phase2: Compare Minimum Cost of Settings

	Instance																							
Test	**gdb1**	**gdb2**	**gdb3**	**gdb4**	**gdb5**	**gdb6**	**gdb7**	**gdb8**	**gdb9**	**gdb10**	**gdb11**	**gdb12**	**gdb13**	**gdb14**	**gdb15**	**gdb16**	**gdb17**	**gdb18**	**gdb19**	**gdb20**	**gdb21**	**gdb22**	**gdb23**	**Total Cost**
2a	**300**	**321**	**259**	**266**	**361**	**285**	**325**	**334**	**286**	**281**	**387**	**447**	**528**	**96**	**56**	**125**	**91**	**162**	**55**	**121**	**156**	**198**	**233**	**5673**
2b	**300**	325	**259**	**266**	**361**	**285**	**325**	**334**	**286**	**281**	**387**	**447**	**528**	**96**	**56**	**125**	**91**	**162**	**55**	**121**	**156**	**198**	**233**	**5677**
Minimum Cost	**300**	**321**	**259**	**266**	**361**	**285**	**325**	**334**	**286**	**281**	**387**	**447**	**528**	**96**	**56**	**125**	**91**	**162**	**55**	**121**	**156**	**198**	**233**	**5673**

Table 11: Phase2: Compare Average Cost of Settings

	Instance																							
Test	**gdb1**	**gdb2**	**gdb3**	**gdb4**	**gdb5**	**gdb6**	**gdb7**	**gdb8**	**gdb9**	**gdb10**	**gdb11**	**gdb12**	**gdb13**	**gdb14**	**gdb15**	**gdb16**	**gdb17**	**gdb18**	**gdb19**	**gdb20**	**gdb21**	**gdb22**	**gdb23**	**Total Cost**
2a	**310**	**329**	271	**266**	**376**	**298**	**330**	340	**291**	**284**	**388**	449	537	**100**	**57**	**126**	**91**	**162**	**55**	**121**	**158**	200	**236**	**5774**
2b	311	**329**	**269**	**266**	**376**	**298**	**330**	**339**	**291**	**284**	389	**447**	**536**	**100**	**57**	**126**	**91**	**162**	**55**	**121**	**158**	**199**	**236**	**5771**
Minimum Average Cost	**310**	**329**	**269**	**266**	**376**	**298**	**330**	**339**	**291**	**284**	**388**	**447**	**536**	**100**	**57**	**126**	**91**	**162**	**55**	**121**	**158**	**199**	**236**	**5771**

Table 12: Phase 2: Compare Average Time of Settings

	Instance																							
Test	**gdb1**	**gdb2**	**gdb3**	**gdb4**	**gdb5**	**gdb6**	**gdb7**	**gdb8**	**gdb9**	**gdb10**	**gdb11**	**gdb12**	**gdb13**	**gdb14**	**gdb15**	**gdb16**	**gdb17**	**gdb18**	**gdb19**	**gdb20**	**gdb21**	**gdb22**	**gdb23**	**Average Time**
2a	16.8747	**27.5899**	**24.8254**	5.4292	22.8632	22.8204	23.4820	25.6649	**18.3570**	14.8767	18.4185	25.5037	**11.6926**	23.4600	**6.9286**	**24.8429**	4.4087	**14.1833**	8.2526	**9.4344**	**20.8743**	**17.6321**	27.0283	**18.0628**
2b	**13.1680**	28.1645	28.1315	**5.0816**	**21.1629**	**20.1322**	**21.1383**	**22.3324**	20.3410	**10.1377**	**14.2664**	**25.1566**	13.6195	**21.1213**	7.3938	30.1980	**4.2685**	15.1976	**8.1730**	10.1168	23.2102	21.2525	**19.3189**	**17.5254**

In this second phase we get very similar results between using and not using the ”unknotting” function, only in instance gdb2 we can see an improvement. It is to be noticed that in the average cost table, not using the function provided the best results. This could be due to the algorithm of the function actually producing knots in the routes while undoing other knots. Still, since we are more concerned with the minimum values than the average, the use of the unknotting function is kept for the next phase.

In this last phase there are three modifications to the simulated annealing algorithm that we test. The splitting search used both to generate the starting solution for the SA algorithm and the one used within this algorithm can be set to only return solutions that improve the original solutions, or to return solutions that don’t necessarily do. This gives modifications to test. Furthermore, the direct acceptance of solutions can be set to only apply for solutions improving the best solution overall, or for solutions improving only the base solution used within the simulated annealing algorithm, not caring if it improves the best overall solution. The following flowcharts clarify this, each flowchart details each one of the setups described previously.

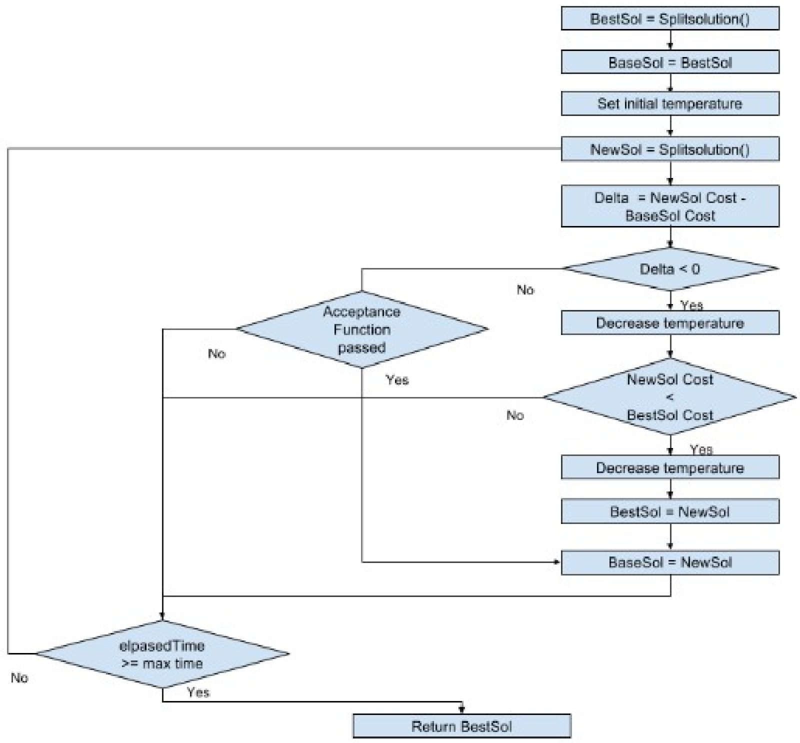


Figure 12: Accepting only solutions improving best solution

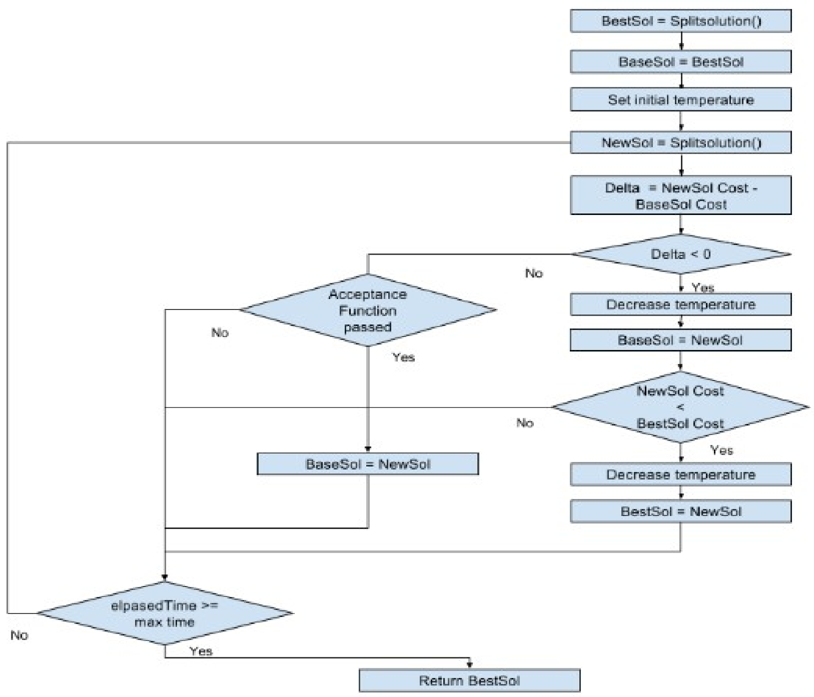


Figure 13: Accepting solutions improving base solution

The following tables detail the tests that were performed and the results of those tests.

Table 13: Third Phase

Phase 3 (SA Algorithm variations)	3-a	3-b	3-c	3-d	3-e	3-f	3-g	3-h
Outer MultiSplit With improvement	No	No	No	No	Yes	Yes	Yes	Yes
Inner MultiSplit With improvement	No	No	Yes	Yes	No	No	Yes	Yes
Accept solutions improving base but not best (baseSol = newSol)	No	Yes	No	Yes	No	Yes	No	Yes

Table 14: Phase3: Compare Minimum Cost of Settings

	Instance																							
Test	gdb1	gdb2	gdb3	gdb4	gdb5	gdb6	gdb7	gdb8	gdb9	gdb10	gdb11	gdb12	gdb13	gdb14	gdb15	gdb16	gdb17	gdb18	gdb19	gdb20	gdb21	gdb22	gdb23	Total Cost
3a	300	321	259	266	361	285	325	334	286	281	387	447	528	96	56	125	91	162	55	121	156	198	233	5673
3b	300	321	259	266	361	285	325	334	286	281	387	447	528	96	56	125	91	162	55	121	156	198	233	5673
3c	300	321	259	266	361	285	325	334	286	281	387	447	528	96	56	125	91	162	55	121	156	198	233	5673
3d	300	321	259	266	361	285	325	334	286	281	387	447	530	96	56	125	91	162	55	121	156	198	233	5675
3e	300	325	259	266	361	285	325	334	286	281	387	447	528	96	56	125	91	162	55	121	156	198	233	5677
3f	300	325	259	266	361	285	325	334	286	281	387	447	528	96	56	125	91	162	55	121	156	198	233	5677
3g	300	325	259	266	361	285	325	334	286	281	387	447	528	96	56	125	91	162	55	121	156	198	233	5677
3h	300	325	259	266	361	285	325	334	286	281	387	447	528	96	56	125	91	162	55	121	156	198	233	5677
Minimum Cost	300	321	259	266	361	285	325	334	286	281	387	447	528	96	56	125	91	162	55	121	156	198	233	5673

Table 15: Phase3: Compare Average Cost of Settings

	Instance																							
Test	gdb1	gdb2	gdb3	gdb4	gdb5	gdb6	gdb7	gdb8	gdb9	gdb10	gdb11	gdb12	gdb13	gdb14	gdb15	gdb16	gdb17	gdb18	gdb19	gdb20	gdb21	gdb22	gdb23	Total Cost
3a	310	329	271	266	376	298	330	340	291	284	388	449	537	100	57	126	91	162	55	121	158	200	236	5774
3b	311	328	269	266	376	298	330	339	291	284	389	447	536	100	57	126	91	162	55	121	158	199	235	5769
3c	310	328	271	266	376	298	330	340	291	283	389	449	537	100	57	126	91	162	55	121	158	199	236	5774
3d	310	329	271	266	376	298	330	340	291	284	389	449	537	100	57	126	91	162	55	121	158	199	236	5774
3e	311	329	269	266	376	298	330	339	291	284	389	447	536	100	58	126	91	162	55	121	158	199	235	5771
3f	311	329	269	266	376	298	330	339	291	284	389	447	536	100	57	126	91	162	55	121	158	199	236	5770
3g	311	329	269	266	376	298	330	339	291	283	389	447	536	100	57	126	91	162	55	121	158	199	235	5770
3h	311	329	269	266	376	298	330	339	291	284	389	447	536	100	57	126	91	162	55	121	158	199	236	5771
Minimum Average Cost	310	328	269	266	376	298	330	339	291	283	388	447	536	100	57	126	91	162	55	121	158	199	235	5769

Table 16: Phase 3: Compare Average Time of Settings

	Instance																							
Test	gdb1	gdb2	gdb3	gdb4	gdb5	gdb6	gdb7	gdb8	gdb9	gdb10	gdb11	gdb12	gdb13	gdb14	gdb15	gdb16	gdb17	gdb18	gdb19	gdb20	gdb21	gdb22	gdb23	Average Time
3a	16.8747	27.5899	24.8254	5.4292	22.8632	22.8204	23.4820	25.6649	18.3570	14.8767	18.4185	25.5037	11.6926	23.4600	6.9286	24.8429	4.4087	14.1833	8.2526	9.4344	20.8743	17.6321	27.0283	18.0628
3b	13.1911	28.1685	28.1359	5.0850	21.1649	20.1287	22.1400	22.3275	18.3636	10.1340	14.2673	25.1718	13.6055	21.1321	5.7170	30.1841	4.2801	15.1830	8.1995	10.1118	23.2159	21.2595	22.3669	17.5449
3c	16.8484	27.5030	24.7911	5.4130	22.8404	22.8049	22.8087	25.6682	19.0134	18.3825	16.2778	25.4915	12.8887	23.4617	6.1179	23.4982	4.4537	14.1842	8.2831	9.4385	20.8837	17.5963	25.6847	18.0145
3d	16.8260	27.4970	24.7913	5.4111	22.8352	22.7936	23.4705	25.6796	19.0289	14.8474	16.2858	25.4548	12.9302	23.4579	5.6584	24.8352	4.4423	14.1762	8.1922	9.4360	20.8821	17.5860	27.0228	17.9800
3e	13.1755	28.1799	28.1858	5.0879	21.1707	20.1352	21.1469	22.3352	20.3507	10.1422	14.2668	25.1632	13.6055	21.1303	3.8698	30.1884	4.2618	15.1941	8.1682	10.1137	23.2221	21.2530	22.3419	17.5082
3f	13.1814	28.1668	28.1461	5.0836	21.1821	20.1321	22.1468	22.3264	20.3767	10.1399	14.2885	25.1502	11.6228	21.1222	5.8947	30.1809	4.2916	15.1982	8.3012	10.1170	21.2052	19.2674	22.3526	17.3858
3g	13.1904	28.1631	28.1372	5.0895	21.1632	20.1309	21.1318	22.3179	20.3334	15.9760	14.2857	25.1482	13.5919	21.1278	5.3855	28.1764	4.3518	15.1845	8.1890	10.1078	23.2158	21.2752	22.3445	17.7399
3h	13.1961	28.1743	28.1396	5.0833	21.1727	20.1297	21.1347	22.3360	21.4096	10.1422	14.2754	25.1593	13.5756	21.1221	4.8819	30.1824	4.2567	15.2154	8.5229	10.1116	23.2099	21.2417	22.3447	17.6095

From these results we can conclude that it is better to not force the splitting procedure to only return improving solutions. It is coherent with the random nature of a simulated annealing approach, which needs of some non-improving solutions to function as it is intended. Regarding the acceptance of solutions improving only the base solution, this also seems to offer better results as can be seen in the results for the average cost. It even reaches the solution in less time than its counterpart.

Table 17 shows the results of the present algorithm and those of Hongtao et al. (2013)[12], Kansou (2010) [14] and Kansou and Yassine (2012)[16].

Table 17: Comparison of results

	Kansou 2012				Kansou 2010		HongTao		Present Algorithm					
Instance	**HACO**	**Time**	**DACOS**	**Time**	**MDMA**	**time**	**HGAP**	**Time**	**Result**	**Time**	**Gap (HACO)**	**Gap (DACOS)**	**Gap (MDMA)**	**Gap (HGAP)**
gdb1	300	<1	300	<1	300	<1	300	<1	311	13.19	3.67	3.67	3.67	3.44
gdb2	331	<1	329	1.2	321	<1	321	<1	328	28.17	**-0.91**	**-0.30**	2.18	2.19
gdb3	267	<1	273	<1	263	<1	259	<1	269	28.14	0.75	**-1.47**	2.28	3.86
gdb4	266	<1	269	<1	266	<1	266	<1	266	5.08	0.00	**-1.12**	0.00	0.00
gdb5	369	<1	364	1.3	361	<1	361	<1	376	21.16	1.90	3.30	4.16	3.96
gdb6	358	<1	358	<1	291	<1	282	<1	298	20.13	**-16.76**	**-16.76**	2.41	5.50
gdb7	325	<1	325	<1	325	<1	325	<1	330	22.14	1.54	1.54	1.54	1.40
gdb8	351	1.5	359	1.8	350	1.7	328	4.3	339	22.33	**-3.42**	**-5.57**	**-3.14**	3.27
gdb9	314	1.7	314	2	309	2.1	279	4.7	291	18.36	**-7.32**	**-7.32**	**-5.83**	4.16
gdb10	275	<1	275	<1	275	<1	275	<1	284	10.13	3.27	3.27	3.27	3.06
gdb11	407	<1	407	1.7	403	<1	387	1.9	389	14.27	**-4.42**	**-4.42**	**-3.47**	0.51
gdb12	450	<1	454	<1	440	<1	420	<1	447	25.17	**-0.67**	**-1.54**	1.59	6.10
gdb13	540	<1	540	1	540	<1	528	<1	536	13.61	**-0.74**	**-0.74**	**-0.74**	1.49
gdb14	98	<1	98	<1	96	<1	96	<1	100	21.13	2.04	2.04	4.17	4.19
gdb15	56	<1	56	<1	56	<1	56	<1	57	5.72	1.79	1.79	1.79	2.10
gdb16	127	<1	127	<1	127	<1	125	<1	126	30.18	**-0.79**	**-0.79**	**-0.79**	0.48
gdb17	91	<1	91	<1	91	<1	91	<1	91	4.28	0.00	0.00	0.00	0.00
gdb18	160	<1	160	1.5	158	<1	158	<1	162	15.18	1.25	1.25	2.53	2.47
gdb19	55	<1	55	<1	55	1.1	55	<1	55	8.20	0.00	0.00	0.00	0.00
gdb20	122	<1	123	<1	121	1.5	121	<1	121	10.11	**-0.82**	**-1.63**	0.00	0.00
gdb21	158	<1	158	<1	158	1.8	154	2.8	158	23.22	0.00	0.00	0.00	2.65
gdb22	202	1.3	202	1	201	2.6	196	4.7	199	21.26	**-1.49**	**-1.49**	**-1.00**	1.61
gdb23	235	1.6	236	1.5	235	3.2	229	6.4	235	22.37	0.00	**-0.42**	0.00	2.68

From the results we see that the algorithm gives good results, and in some instances it equals the results of Hongtao et al. (2013)[12], but it seems clear that there are many improvements to be made to the algorithm to achieve it's values. Particularly the running time of the algorithm should be improved.

6. Conclusions

In this article several strategies for approaching the Multi-depot ARP. Particular attention has been placed on the strategies to decide which depot will be serving which edge, but also some variations on a main algorithm were tested to determine the value of pursuing the refinement of these variations. The computational experiments show that the best way to assign edges to depots is by assigning a probability of assignment of this edge to the two closest depots according to the cost associated to traveling from each depot to the edge, and then randomly making the assignment with this probabilities. Other assignment strategies have returned good solutions and might even perform better on larger networks, with more depots and edges without required demand. Regarding the variations on the main algorithm, a randomized version of the SHARP has proven to offer better results, since from the variety of solutions it can return, many search algorithms can be applied, and pairing these with a simulated annealing approach might prove to be a most valuable use of these various solutions. The use of a cache mechanism has improved the time it takes the algorithm to achieve the minimum, but it hasn't significantly decreased the cost of the solutions. Again this might be more valuable in larger networks. Also, the use of a pool of solutions obtained from the initial multi-start procedure gives the possibility of exploring more solutions and not get trapped in local optima. Like it has been proven with the splitting search, accepting non improving solutions in thecontext of a simulated annealing framework is desirable. Finally, the "unknotting" procedure hasn't proved to be definitely favorable or unfavorable. Although theoretically it should never increase the cost of a solution, the results show that this might be the case for some instances. One thing to analyze is whether in the process of "unknotting" a subset of three edges, it is creating a new knot in previously "unknotted" edges.

Some ideas for future work are: (i) run these full suite of tests to larger networks with more depots and including edges without required demand, since the allocation strategies might perform very differently in these networks; (ii) further analyze the "unknotting" algorithm, isolated from the other strategies; (iii) test in isolation the cache mechanism in larger networks since its value might actually reside in those kinds of networks; (iv) analyze different alternatives to reduce the time performance of the algorithm (v) add capacity restrictions to the depots; and (vi) add restrictions on time-capacity to the problem.

7. Glossary

ARP: Arc Routing Problem
VRP: Vehicle Routing Problem
MDARP: Multi-Depot Arc Routing Problem
MDVRP: Multi-Depot Vehicle Routing Problem
CARP: Capacitated Arc Routing Problem
CVRP: Capacitated Vehicle Routing Problem
SHARP: Savings-based Heuristic for the ARP
CWS: Clarke And Wright Savings

8. Bibliography

[1]Amberg, A., W. Domschke, and S. Voß. 2000. “Multiple center capacitated arc routing problems: A tabu search algorithm using capacitated trees”. European Journal Of Operational Research.

[2]Assad, A., and B. Golden. 1995. “Arc routing methods and applications”. Handbooks in operations research and management science.

[3]Clarke, G., and J. Wright. 1964. “Scheduling of vehicles from a central depot to a number of delivery points”. Journal Of Operations Research.

[4]Corberán, A., and C. Prins. 2010. “Recent results on Arc Routing Problems: An annotated bibliography”. Networks.

[5]Dror, M. 2000. Arc routing: theory, solutions, and applications. Springer.

[6]Eiselt, H., M. Gendreau, and G. Laporte. 1995a. “Arc routing problems, part I: The Chinese postman problem”. Operations Research.

[7]Eiselt, H., M. Gendreau, and G. Laporte. 1995b. “Arc routing problems, part II : The Rural Postman Problem”. Operations Research.

[8]Fleury, M. 1883. “Deux problemes de geometrie de situation”. Journal de mathematiques elementaires.

[9]Golden, B., J. Dearmon, and E. Baker. 1983. “Computational experiments with algorithms for a class of routing problems”. Computers & Operations Research 10 (1): 47 – 59.

[10]González, S., A. A. Juan, D. Riera, Q. Castellà, R. Muoz, and A. Prez. 2012. “Development and assessment of the SHARP and RandSharp algorithms for the arc routing problem”. AI Communications.

[11]Hierholzer, C. 1873. “ber die Mglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren”. Mathematische Annalen VI.

[12]Hongtao, H., L. Tangtao, Z. Ning, Z. Yiting, and M. Dequan. 2013. “A hybrid genetic algorithm with perturbation for the multi-depot capacitated arc routing problem”. Journal of Applied Sciences.

[13]Juan, A., I. Pascual, D. Guimarans, and B. Barrios. 2014. “Combining biased randomization with iterated local search for solving the multidepot vehicle routing problem”. International Transactions in Operational Research.

[14]Kansou, A. 2010. “New upper bounds for the multi-depot capacitated arc routing problem”. International Journal of Metaheuristics 1:81–95.

[15]Kansou, A., and A. Yassine. 2009. “A two ant colony approaches for the multi-depot capacitated arc routing problem”. Technical report, Laboratoire de Mathematiques Appliquees du Havre (LMAH), Universite du Havre, France, 1nstitut Superieure d’Etudes Logistiques (ISEL), Universite du Havre, France.

[16]Kansou, A., and A. Yassine. 2012. “Splitting algorithms for the multiple depot arc routing problem: application by ant colony optimization”. International Journal of Combinatorial Optimization Problems and Informatics 3 (3): 20 – 34.

[17]Mei-Ko, K. 1962. “Graphic Programming Using Odd or Even Points”. Chinese Mathematics.

Sachs, H., M. Stiebitz, , and R. J. Wilson. 1988. “An historical note: Eulers Knigberg letters”. Journal Of Graph Theory.

[18]Tiantang, L., J. Zhibin, and G. Na. 2014. “A genetic local search algorithm for the multi-depot heterogeneous fleet capacitated arc routing problem”. Flexible Services And Manufacturing Journal.

[19]Wohlk, S. 2005. Contributions to Arc Routing. MSc. thesis, University of Southern Denmark.

[20]Wohlk, S. 2008. “A Decade of Capacitated Arc Routing”. In The Vehicle Routing Problem; Latest Advances and New Challenges. Springer.

[21]Xing, L., P. Rohlfshagen, Y. Chen, , and X. Yao. 2009. “An Evolutionary Approach to the Multidepot Capacitated Arc Routing Problem”. IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION.