Erasmus University Rotterdam

Bachelor thesis in Quantitative Logistics

Evolutionary Algorithms for Waiting Strategies: the influence of Selection Methods

Author

Marja van der Wind (498306mw)

Supervisor

Y.N. Hoogendoorn

Date

July 3rd, 2021

Second assessor

dr. P.C. Bouman

Abstract

Problems on pickup and delivery of goods and persons have many applications. In the dynamic variant of the Vehicle Routing Problem, new information can become available during the day. Therefore, routes are constructed in advance using known information but may be adapted while driving to insert new customer locations in the route. To obtain solutions, it is important to anticipate possible future requests, which can be done by applying waiting strategies. Waiting strategies are precepts that prescribe decisions on which spots of the route drivers should wait a while. Thus, this thesis contributes to research on the Waiting Drivers Problem. The scope of this problem is to apply waiting strategies to routes to increase the chances of inserting new customers. This is done by implementing simple heuristics and an Evolutionary Algorithm, based on results from Branke et al. (2005). Moreover, a comparison of multiple parent selection methods used in the Evolutionary Algorithm was performed. The heuristics were found to perform similarly as in Branke et al. (2005). The parent selection methods from Back (1994) perform slightly better than the Linear Ranking Selection method used in Branke et al. (2005), but the advantages are minimal. A newly introduced selection method named Golden Ratio sometimes performs very well but is not reliable over different instances.

Contents

1 Introduction

Obtaining optimal schedules for routing problems has been a challenging task for decades. Recently, more and more attention is being paid to the emergence of digital communication that makes it possible to change routes while they are being driven, as was done in Reinartz et al. (2019). It is even possible to depart without a route available and inform the driver about the next destination upon arrival at each customer. Also, with the development of communication via smartphones, customers change their behavior towards more demanding expectations. All parties dealing with delivery and traveling need to change their operational strategies to meet these expectations and the focus of research should be shifted to dynamic or adapting problems.

To contribute to this research field, this thesis is dedicated to the so-called Waiting Driver Problem (WDP) defined in Branke et al. (2005). This problem occurs when drivers planned a fixed number of appointments in a certain order, but an unexpected new request shows up during the day. In some cases, these new appointments are emergencies and are thus preferred to be scheduled on the same day. Then, drivers insert the new appointment in their planned route. To increase the chances of being able to serve the new customer the same day, the driver may choose to wait at certain locations in the hopes of decreasing the detour of visiting a new customer.

The problem studied in this research thesis is stochastic and dynamic. The problem can be seen as stochastic because the customer arrivals can be described by a random variable with a known probability distribution. Also, it is dynamic because it is concerned with new customers arriving over time, other than the similar static problem in which all customer requests would be known while scheduling. This makes the problem a variant of the Dynamic Vehicle Routing Problem. The main goal of this thesis is to research which waiting strategy methods are empirically the best to implement in WDPs. First, we implement the simple heuristics and the Evolutionary Algorithm (EA) proposed in Branke et al. (2005), limiting the scope of this thesis to the case that only one beforehand unknown customer request arrives each day. The EA is further elaborated on by implementing variations on the selection method of the EA to analyze the impact of choices on implementation.

The structure of this report is as follows. First, literature regarding our problem will be covered in Section 2. Then, a description of the problem will be given in Section 3. In Section 4, the implementation of multiple methods is explained and in Section 5, the obtained results are presented. Lastly, the discussion and conclusions can be found in Section 6.

2 Literature review

In this section, literature research on the Dynamic Vehicle Routing Problem (DVRP) and on the Waiting Drivers Problem (WDP) is evaluated. Then, the Evolutionary Algorithm is elaborated on.

2.1 Dynamic Vehicle Routing Problem

The Vehicle Routing Problem (VRP) first appeared in Dantzig and Ramser (1959) as a generalization of the well-known Travelling Salesman Problem introduced in Flood (1956). The VRP is a combinatorial optimization and integer programming problem which aims to find the optimal set of routes to visit a given set of customers by use of a fleet of vehicles. One of many variants on the VRP is the Dynamic Vehicle Routing Problem (DVRP), which is first referenced in Wilson and Colvin (1977). In this paper, the destinations of the trips appear dynamically. Their approach uses insertion heuristics that perform well with low computational effort.

Later, in Psaraftis (1988) the concept of an immediate request was introduced: a customer requesting service always wants to be serviced as early as possible, requiring immediate replanning of the current vehicle route. Finally, the dynamic aspect of the DVRP was first explicitly formulated in Psaraftis (1988), where it is no longer seen as a direct adaption of a static VRP, but includes specific characteristics such as (1) essentially of a time dimension, (2) unknown or imprecise future information, (3) emphasis on soon events and (4) necessity of fast computation times. Later, in Bianchi (2000), two conditions were stated which make an optimization problem dynamic, namely that (1) information on the problem must be time-dependent in the sense that information is updated and made known as time goes on, and (2) solutions must be found while time proceeds, which means that no prior solutions can be found, but strategies can be found beforehand.

For every static VRP, it is possible to think of many DVRPs by revealing dynamically the parameters that define the instance, such as information on customer demand for services as in Bertsimas and Van Ryzin (1991) or for goods as in Mitrovic-Minic et al. (2004). In Bertsimas and Van Ryzin (1993) the customer's position is dynamic in their so-called Dynamic Traveling Repairman Problem, a specific variant of the DVRP. In Mitrovic-Minic et al. (2004) the Pick-up and Delivery Problem was comprised, where goods have to be picked up and delivered in specific amounts at the vertices within dynamic time windows. Also, travel time can form the dynamic part of the DVRP as it does in taxi cab problems, which has been taken into account in Fleischmann et al. (2004). In Novoa (2005) dynamically revealed demands for a set of known customers and vehicle availability are considered, in which case the source of dynamism is the possible breakdown of vehicles, as elaborated on in Montemanni et al. (2005).

Then, the Waiting Drivers Problem, another variant of the DVRP, is introduced in Branke et al. (2005) and is dynamic in the sense that arrival time and position are unknown in advance. As noted in a research review in Pillac et al. (2013), this is also the most common source of dynamism in vehicle routing. Later, in Vonolfen and Affenzeller (2016) the dynamic components of the Pickup and Delivery Problem and the Waiting Drivers Problem were combined to one optimization problem. To the best of the author's knowledge, no other papers on the Waiting Drivers Problem have been published.

2.2 Evolutionary Algorithms

An Evolutionary Algorithm is a metaheuristic that optimizes a (constrained) objective in a large space of possible solutions. Fogel (1965) was the first to use evolutionary programming (EP), used to solve numerical optimization problems. Then, in Holland (1973), genetic algorithms (GA) were presented, inspired by the genetic code found in natural life, to solve combinatorial problems. Classically, the representation of real-world solutions is done by means of binary solution coding. The main search operator to traverse the possible solution space is the crossover operator, which recombines several solutions into new ones. Mutation is only a background operator whose major responsibility is to prevent the algorithm from being trapped in a local optimum, by making minimal arbitrary changes in any solution.

Evolution Strategies (ES) as proposed in Huning (1976) were motivated by engineering problems and thus mostly used a real-valued representation. Later, genetic programming was introduced in Koza and Koza (1992) as a form of GA that is used for inductive programming, a special area in automatic programming. In this area, computers learn declarative and recursive programs from incomplete specifications. As is explained in Necker (1998), ES operate on the level of the phenotype, whereas genetic algorithms operate on the level of the genotype. ES thus have a higher degree of abstraction and are often used to optimize continuous decision variables only. Both GA and ES are forms of evolutionary algorithms. The execution of an evolutionary algorithm is memory-consuming, which is first elaborated on in Branke (1999).

Another issue that researchers stumbled upon is that in most real-life problems restrictions are involved. Unfortunately, these are not straightforward to handle, because the mutation and recombination operators are mostly *blind* to constraints, as explained in Chambers (2000). To handle this, he comes up with different approaches such as elimination or repairing infeasible candidates. Since the introduction of these algorithms, both GA and EA have been used for a wide range of problems. For instance, in Branke and Schmeck (2003), EA is applied on various dynamic optimization problems.

Moreover, many comparisons and reviews are published, such as Eiben and Schoenauer (2002) and Eiben, Smith, et al. (2003). For instance, an EA was applied on the DVRP in Zhou et al. (2003). Then, Branke et al. (2005) applied an EA on a variant of the DVRP, namely on a Waiting Drivers Problem. Later, the use of EA on DVRPs was extended in Escuin et al. (2016) for research on cooperation strategies in routing in docility of Branke et al. (2005) as the first work that has touched cooperative strategies.

Selection methods in EAs A specific step that truly defines the outcome of the EA is the selection of parents in the generation step, which is further elaborated on in Section 4.2.2. These parents are later used to obtain children from. A linear assignment of probabilities in the selection of parents in the generation step was used as in Branke et al. (2005). The selection method used in an Evolutionary Algorithm plays a major role since it determines the direction of search and has an impact on the exploration-exploitation trade-off. To evaluate selection methods in terms of this trade off, the concept of *selection pressure* was introduced in Back (1994). High selective pressure is defined as a strong emphasis on selecting the best individuals, where a low selective pressure means the opposite, namely a weak emphasis on selecting the best individuals. More formally, this measure is quantified by the number of generations that is required to fill the complete population with copies of the single best individual. Since a high selective pressure emphasizes more on selecting good individuals, the convergence speed of methods with high selective pressure is also high. This has the consequence of higher risks of developing into a local optimum instead of the global optimum we search for. A low selective pressure, however, emphasizes more on diversity in solutions and leads to better convergence reliability.

3 Problem definition

In this section, we define the WDP as in Branke et al. (2005). First, a customer set $J = \{0, 1, 2, ..., n\}$ is given with $j = 0$ as the depot. Customer locations are known and represented in a Euclidean plane. A fleet of vehicles is available, represented in a vehicle set I with vehicle $i \in I$ and m the total number of vehicles. The vehicles drive different tours visiting all customers exactly once, which are pre-obtained from solving a static VRP. The set of tours R, contains a tour r_i for each vehicle and is represented as follows: $(0, c_{i1}, c_{i2}, ..., c_{in_i}, 0)$ where c_{ij} is the j^{th} customer that vehicle i visits and n_i equals the number of customers visited on route r_i. Each customer must be visited in exactly one route. We assume that the vehicles execute the customer visits in the order implied by the tour, except for cases where extra customers are serviced along the way.

Let $d_{ij} \geq 0$ with $0 \leq i < j \leq n$ be defined as the Euclidean distance between customers. Conform Branke et al. (2005) it is assumed that vehicles drive on a straight line from one customer to the next with a constant speed, which we set to one distance unit per time unit without loss of generality, such that travel times equal travel distances. All vehicles leave the depot at time 0 or later and must be back by a predetermined time T. We assume that service times are negligible so that the total length of a tour r is defined by its traveling time t_r. Then, the slack time of the tour can be defined as $w_r = T - t_r \geq 0$. The slack time is used as waiting time following a waiting strategy. This waiting strategy consists of a tour-specific waiting strategy for each $r \in R$, which in turn consists of waiting times at customer locations visited in tour r. So the waiting strategy is a planning design to distribute slack time over all customer locations as waiting times.

A day is defined from $t = 0$ to $t = T$, in which the in advance unknown customer request emerges, assumed to follow a uniformly distributed random time $0 \leq t \leq T$ and at a uniformly distributed random location within a defined rectangle. As in Branke et al. (2005), we assign the new customer to the vehicle which current tour requires the smallest detour inserting the new customer if the situation occurs where multiple vehicles can serve the extra customer. Yet if none of the vehicles can incorporate the extra customer, the request will be refused. This policy leads to situations in which locations cannot be visited at certain moments because all vehicles do no longer have enough slack time left to make detours to specific locations in the plane. The decision problem that arises here, is to find an optimal waiting strategy that maximizes the probability of servicing the new customer, wherever and whenever its request will pop up. This is defined in Branke et al. (2005) as the Waiting Drivers Problem (WDP) and is proven to be NP-complete.

4 Methodology

In this section, some simple heuristics and the implementation of the Evolutionary Algorithm (EA) are described, all following the approach of Branke et al. (2005). Then, other implementations of the EA selection method are proposed.

4.1 Simple heuristics

In Branke et al. (2005) six simple waiting time heuristics are presented, which will be reproduced in this thesis. In this paragraph, the implementation is described.

- *NoWait* is defined as not waiting at all. This strategy is called and proven in Branke et al. (2005) to be the best strategy for the one-vehicle-case.
- *Depot* is defined as waiting at the depot as long as possible, completely using the slack time, and then driving the route in one go without any delay.
- *MaxDist* is defined as waiting at the customer the farthest from the depot until the customer request arrives or until the slack time has elapsed, so that the full waiting time w_r is assigned to the furthest customer.
- *Location* is defined as dividing the slack time equally over all locations, thus making a compromise between *NoWait* and *Depot*, so that each customer is assigned waiting time $\frac{w_r}{n_r}$.
- *Distance* is defined as assigning waiting times proportionally to the driven distances. Here, distances are used to divide up waiting times where customers were in the *Location* heuristic. Thus, every customer c_i is assigned waiting time $w_r \cdot \frac{d_{c_i,c_{i-1}}}{\sum_{j=1}^{n_r} d_{c_j,c_{j-1}}}$.
- *Variable* is defined as distributing the available waiting time proportionally to leftover distance, when the remaining distance to the depot is equal to the slack time. This is done by taking the proportion of the distance to the next customer in relation to all distances, using the same procedure as the one for the full tour the *Distance* heuristic. If the requirement to distribute the waiting time is met between two customers, we first drive to the next customer.

These heuristics and explanations can also be found in Branke et al. (2005) and it is proven there that the *Variable* heuristic is optimal for two vehicles with customers on a straight line.

4.2 Evolutionary Algorithm

First, a short overview of the algorithm as explained in Branke et al. (2005) is presented, which is then explained step by step in Algorithm 1. To implement the EA, three main steps must be taken: firstly, a population of initial solutions must be available and if needed transformed into a data structure that we can work with. Secondly, a method to construct new generations from the population is constructed. This part works iteratively, creating a whole new generation of solutions in each iteration. Thirdly, the solutions of the obtained generation are evaluated and selection of the good solution takes place. The chosen solutions will then be used for reproduction in the next iteration. After presenting the algorithm, the reader is provided with more understanding of the implementation of this EA, by discussing all three parts thoroughly in the following paragraphs.

```
Algorithm 1: Evolutionary Algorithm
  Data: Customer locations and routes
  Result: Best waiting strategies
  Initialization: as a first generation, generate 100 random waiting time solutions
  for generation $g \leftarrow 0$ to 100 do
      Set selection probabilities by use of selection method
      for child $i \leftarrow 0$ to 100 do
          Select two parents from the population
          Construct a child solution by combining two parents: Algorithm 2
      end
      Select the best 100 solutions to obtain a new generation
      for test customer $j \leftarrow 0$ to 100 do
          if test customer j fits in then
              $CustomerFits_j \leftarrow CustomerFits_j + 1$
          end
      end
      Construct new population;
          add the best solution of the old generation
          add 99 best children based on the CustomerFits measure
          sort the 100 solutions from best to worst
      end
```

To construct a child, which is a crucial part of the EA, Algorithm 2 is implemented.

Algorithm 2: Child construction

```
Data: Two parents
Result: A child solution
Construct a child solution; combining two parents:
    Substring length $k \leftarrow Random\ int$(1, n)
    Starting point $p \leftarrow Random\ int$(1, n - k +1)
    Create child by crossover:
        Copy elements 0 to $p$ from parent 1
        Copy elements $p$ to $p + k$ from parent 2
        Copy elements $p + k$ to $n$ from parent 1
    Change the solution by mutation:
        forall $elements : child$ do
            $element \leftarrow element + random\ uniform\ double$(0,1)
        end
    Normalize child
```

4.2.1 Population

To stay in the evolutionary sphere, candidate solutions that form a population are also called chromosomes (Escuin et al., 2016). A solution chromosome is represented as a string of $n + 2m$ real values for a problem with m tours and n customers. The strings consist of one value for the waiting time at each customer and two additional values for each tour for the waiting times at the depot at the beginning and at the end of the tour. These real values represent fractions of the slack time for the corresponding tour that are used for waiting at particular locations. Thus, if at any point in time the values of a waiting time solution do not add up to 1, we normalize the solution by dividing each value by the sum of all waiting times. The population consists of a set of yet existing solutions that we aim to improve. In Branke et al. (2005), the heuristics are also combined with the EA in the sense that the six waiting strategies obtained from the heuristics were inserted in the initial population of the EA. The other 94 members were generated randomly. To distinguish between the EAs in this paper, we refer to the original Evolutionary Algorithm as EA1 and to the adjusted population with EA2 as is done in Branke et al. (2005). From these solutions, new solutions will be generated iteratively, as described in the second step.

4.2.2 Construction of a new solution

Secondly, a new generation is constructed by randomly selecting two so-called parent solutions from a population of size λ, and then construct a new solution from these, which is then called a child solution. As is outlined in Branke et al. (2005), the construction of a child solution is done by performing cross-over and mutating the obtained solution slightly. The two parent solutions are selected from the population.

To select the parents, the use of a Linear Ranking Selection method (LRS) is recommended in Branke et al. (2005). To use such a method, first the individuals in the populations are ordered with rank λ assigned to the best element, and rank 1 is assigned to the worst individual. The ranking is done by use of evaluating the solutions as described in Section 3 so that the maximal probability of serving an unknown customer is achieved. For selection purposes, linearly decreasing probabilities are assigned, so that we give the best current solutions the highest chance of being chosen. In the literature, many mapping functions for assigning linearly decreasing probabilities are available, and it is not clear which one is used in Branke et al. (2005). Among the many methods explained and compared in Back (1994), a linear ranking method for the computation of selection probabilities is mentioned that was first proposed in Baker (1985). The selection probability of each solution i can be computed as follows:

$$s_i = \frac{1}{\lambda}\left(\eta^- + (\eta^+ - \eta^-)\frac{i-1}{\lambda-1}\right) \tag{1}$$

where η^+ and η^- are called the maximum and minimum expected value respectively. These determine the slope of the linear function. Here λ is the rank assigned to the best element, which thus can also be understood as the number of elements ranked. This way, $\frac{\eta^-}{\lambda}$ represents the selection probability of the worst individual whereas $\frac{\eta^+}{\lambda}$ exhibits the selection probability of the best element. The restriction $\sum_{i-1}^{\lambda} s_i = 1$ requires that $1 \leq \eta^+ \leq 2$ and that $\eta^- = 2 - \eta^+$, which was used in Baker (1985) to find that a value of $\eta^+ = 1.1$ is most desirable to counteract both undesirable convergence and low percentages of involvement of old solutions to new ones.

To select two parents, it is also possible to use other selection methods. In Back (1994), four selection methods are evaluated on the selective pressure criterion as explained in Section 2, namely proportional selection, linear ranking, tournament selection, and (μ, λ)-selection, here sorted increasingly on selective pressure. In Zhang and Kim (2000), these methods are also reviewed, together with ranking selection, Genitor selection, simulated annealing, and hill-climbing. The experimental results of this paper show that ranking selection and tournament selection perform better than proportional selection. The Genitor selection performed similarly but was very slow. However, selection methods for ranking do not necessarily have to be linear functions. The best-known nonlinear ranking method is presented in Michalewicz (2013), proposing the expression $p_i = c(1-c)^{i-1}$. However, as is explained in Back (1994), the probabilities obtained from this expression do not sum to one necessarily, which must be accounted for when choosing a value of c. Also worth mentioning is the idea in Cuevas et al. (2018), where assigning probabilities to parents is being based on the Golden Section. The here introduced Golden Ratio Selection method is inspired by the Golden Section method.

Proportional Selection With the Proportional Selection method (PS) as described in Holland (1975), individuals are assigned selection probabilities proportionally, based on the number of inserted customers. This results in equal selection probabilities if multiple solutions can insert the same number of customers, thus preventing unfair advantages for equally performing solutions when being ranked. Since the probabilities are order-independent, ranking the individuals as was done in the Linear Ranking Selection method (LRS) in Section 4.2.2, is unnecessary here. Thus, for each solution i we present the number of inserted customers as a proportion p_i of the total number of inserted customers of all solutions:

$$p_i = \frac{c(inserted)}{c(totalinserted)} \tag{2}$$

Note that the probabilities sum up to one. Thus, no normalization is needed here.

Tournament Selection In the Tournament Selection method (TS) as described in Goldberg et al. (1989) as many tournaments are organized as there are solutions in the population because each tournament brings forward one solution for the new population. For each tournament, a group of q individuals is selected randomly. From these tournaments, winners are selected as parents. The winner is the best solution (i.e. with the highest number of inserted customers). Since $q = 2$ is chosen mostly, we will use this too. For higher values of q, selection pressure will increase. Without loss of generality, the solutions are sorted decreasingly in order of the number of customers that could be inserted in the solutions. As is proven in Goldberg et al. (1989), execution of TS assigns each individual i a selection probability of

$$p_i = \lambda^{-q} \cdot ((\lambda - i + 1)^q - (\lambda - i)^q) \tag{3}$$

(μ, λ)-Selection Using the (μ, λ)-Selection method ($\mu\lambda$) as introduced in Schwefel (1981), the original λ-sized population is reduced to a μ-sized offspring population. To ensure a fair comparison with the other selection methods, this method is also tested for 100 generations, also starting with 100 solutions. Thus, we decrease the population size by 1 every generation, choosing $\mu = \lambda - 1$. Since the best μ solutions are chosen each generation, this method is completely deterministic. However, it could be changed into a probabilistic method by assigning equal selection probabilities to the μ best solutions applying the LRS-method, keeping the population size constant, such that the probability p_i for individual i is given as:

$$p_i = \begin{cases} \frac{1}{\mu} & \forall 1 \leq i \leq \mu \\ 0 & \forall \mu < i \leq \lambda \end{cases} \tag{4}$$

Golden Ratio Selection Lastly, a new selection method named the Golden Ratio Selection method (GRS) is introduced in this thesis. This method is inspired by the Golden Section method proposed in Cuevas et al. (2018). In this paper, the Golden Section method divides the population into subgroups, assigning probabilities to each group and choosing groups as a whole from which parents are then taken. First, the Golden Ratio concept is described and then the GRS method is introduced to be used in an EA. The Golden Ratio, also known as ϕ, is a proportion of two values equal to the ratio of their sum to the larger value, which can be algebraically described as $\phi = \frac{A}{B} = \frac{A+B}{A}$. If one would solve this equation for $B = 1$, the obtained exact value for A equals $\frac{1+\sqrt{5}}{2}$. The GRS-method that is introduced here, is based on the Golden Ratio. The selection probability of solution $i + 1$ is such that the selection probability of solution i and solution $i + 1$ is divided between them corresponding to the Golden Ratio, so that the selection probability of individual i is:

$$p_i = \begin{cases} \frac{1}{\phi} \cdot (1 - \frac{1}{\phi})^i & i = 1, ..., n-1 \\ (1 - \frac{1}{\phi})^i & i = n \end{cases} \tag{5}$$

which is basically the nonlinear ranking method as in Michalewicz (2013) with a choice for c of $\frac{1}{\phi}$, where p_n is altered by truncating the geometric series lying under these probabilities, to account for them to sum up to one.

In each iteration, we randomly select 100 pairs of parents using one of the above-explained selection methods, yielding 100 children. To compare the impact of the use of different selection methods with the impact of choosing a different starting population, all selection methods are implemented into EA1. EA2, on the other hand, is only being tested on the standard selection method LRS as was used in Branke et al. (2005). The PS, TS, $\mu\lambda$ and GR selection methods on EA1 and the population construction method of EA2 are thus all compared to the base case of the LRS method on EA1, by use of relative performances. For each pair of parents, we obtain a child as in Branke et al. (2005) by conducting a two-point crossover in which a random substring of the string from one parent is replaced by the corresponding substring of the other parent. We choose a random substring of the chromosome of one parent and supply the other values of the other parent. The method of choosing the random substring is given in Appendix A.

Then, to apply mutation, a random value is added to each value of the newly obtained child solution, to change the solution slightly and prevent the EA from getting stuck in a local minimum. As proposed in Luke (2015), it is possible to apply Gaussian convolution, which means drawing these values from a Gaussian distribution, which involves an extra step of choosing and finetuning a mu (moving mean) and sigma (moving standard deviation). However, the values are normalized afterward, so it does not make sense to implement this method. Instead, we draw the random values independently from a standard normal distribution. In any approach, the crossover and mutation operators may result in values with a sum other than one, so the new solution is always normalized afterward as is explained to be required in Branke et al. (2005).

4.2.3 Child selection

Thirdly, the 99 best children along with the best solution of the last generation form the new population and replace the old population so that it is no longer used. To obtain such a new generation an evaluation method is needed. Evaluating solutions is tricky because it is not clear how to calculate the probability of being able to include an additional customer into one of the tours. To estimate this probability, we follow their simulation method of creating one set of 100 customers with which we evaluate all solutions, to guarantee fair comparisons. Then, each solution is evaluated individually by checking all these customers for insertion feasibility. The percentage of customers that could be inserted is taken as an estimate of the solution quality, where a higher percentage is then equivalent to an expectedly better solution. To move on to the next generation, the 99 best-evaluated solutions are selected into the new generation, together with the best solution until now.

5 Results

In this section, the results for the implemented methods to obtain waiting time strategies will be evaluated. The methods are tested on instances obtained from the OR library (Beasley (n.d.)), namely c50, c75, c100, c100b, c120, c150, and c199. As in Branke et al. (2005), the tours of an optimal solution are used as given tours for the WDP and the maximum allowed travel time T was defined as the travel time required for the longest tour. Also, the service region is defined as the rectangle limited by maximum and minimum values of x and y coordinates of the locations of known customers.

First, the results of the heuristics and EAs as in Branke et al. (2005) are presented and compared to the results in Branke et al. (2005). Then, the results of the various implemented selection methods for the EA are displayed and commented on. All final solutions are evaluated on a fixed set of 1,000 customers. For each waiting strategy, 20 test runs were performed and the results were averaged.

First, the results of the heuristics are presented in Table 1 for all instances. The table provides a mean and standard error for the number of customers that could **not** be inserted, using the corresponding waiting strategy on the instance solutions. Since this is a minimization objective, lower values are better. All results are compared relative to the strategy *NoWait*, thus meaning that relative performances of less than 100% show better approaches than the *NoWait* heuristic.

Table 1: Customers that could **not** be inserted for the waiting strategies at all instances

		NoWait	Depot	MaxDistance	Location	Distance	Variable
c50	Cust. not ins.	383.8 ± 6.0	754.9 ± 10.1	418.6 ± 6.6	431.1 ± 6.9	434.5 ± 6.8	376.9 ± 6.0
	Rel. perf.	100	196.7	109.1	112.3	113.2	98.2
c75	Cust. not ins.	305.0 ± 1.6	473.3 ± 3.5	318.9 ± 3.4	284.0 ± 3.4	305.8 ± 3.2	256.9 ± 3.4
	Rel. perf.	100	251.5	104.6	93.1	100.3	84.2
c100	Cust. not ins.	267.8 ± 2.0	426.8 ± 1.5	299.1 ± 3.0	288.4 ± 2.1	264.2 ± 1.9	311.9 ± 2.0
	Rel. perf.	100	159.4	111.7	107.7	98.7	116.5
c100b	Cust. not ins.	427.6 ± 2.9	600.9 ± 2.3	475.0 ± 2.2	400.9 ± 2.2	442.8 ± 2.4	394.2 ± 2.2
	Rel. perf.	100	140.5	106.9	93.8	103.5	92.2
c120	Cust. not ins.	344.2 ± 1.8	552.9 ± 1.6	356.7 ± 1.6	339.3 ± 1.8	326.2 ± 1.6	322.1 ± 1.2
	Rel. perf.	100	160.6	103.6	98.6	94.8	93.6
c150	Cust. not ins.	428.9 ± 2.0	570.1 ± 2.3	469.8 ± 2.2	386.5 ± 2.4	413.8 ± 2.4	410.4 ± 2.2
	Rel. perf.	100	132.9	109.6	90.1	96.5	95.7
c199	Cust. not ins.	342.8 ± 21.2	454.0 ± 19.6	377.6 ± 20.8	325.6 ± 17.3	324.8 ± 19.0	264.9 ± 19.7
	Rel. perf.	100	132.4	110.1	95.0	94.7	92.7

For all instances, the results differ for a maximal value of 100 in magnitude from Branke et al. (2005) in absolute terms, implying different choices in implementation. Since many choices had to be made during the implementation and the evaluation is dependent on random numbers, it is not surprising that the results differ. Although the absolute differences are substantial, the relative values mostly follow the same trends and yield the same conclusions as in Branke et al. (2005). It is evident from these results that the solution that uses the full slack time at the Depot performs worst. Furthermore, *MaxDistance* also performs worse than *NoWait* for every instance. The other heuristics perform differently over instances. Heuristic *Location*, for instance, mostly performs better than *NoWait*, except for instances c50 and c100. However, it has only absolute preference in instance c150, since *Variable* does better for most other instances. *Variable* performs best on all instances, except for c100 where it performs relatively badly. For instance c100, *Distance* performs best, whilst the other heuristics all perform worse compared to *NoWait*. Most of the above findings are in line with the findings in Branke et al. (2005). The realizations that the findings of relative performances are so similar can be backed by the observation that heuristics with high objective values mostly also differ more from Branke et al. (2005), maintaining the same magnitudes.

Now, the results of the Evolutionary Algorithm (EA) are presented in Table 2 for all instances. This table also provides a mean and standard error for the number of customers that could **not** be inserted out of the 1,000 test customers while using certain selection methods. Here, all performances are compared to the basic EA1 method as described in Branke et al. (2005).

It is interesting how the final results of the LRS, PS, TS, and $\mu\lambda$ perform similarly, only showing minor variances. It can be concluded from this that the difference in selection pressure as was reported in Back (1994) does not have much impact on our results. The GR method, however, performs very differently than the other selection methods. For instances c50, c120, and c150 it performs significantly worse than the other four methods, while it accomplishes great results in the instances c75, c100, c100b, and c199. This is likely due to the fact that the selection pressure of the GR method is incredibly high, assigning a selection probability to the best solution of approximately 0.618. Since the initial solution has a high chance of survival, the starting conditions of the GR method influence the performance considerably. For instances c50 and c120, where the GR method performs rather poorly, good alternatives are available in the PS, TS, and $\mu\lambda$ methods. For instance c150, however, the LRS, TS, and $\mu\lambda$ methods also perform better than the GR method but differ only slightly. Although the GR method sometimes performs up to 9.1% better than the common LRS method, its reliability is not so high. To obtain relatively steady results irrespective of the

Table 2: Customers that could **not** be inserted for the selection methods of EAs at all instances

		EA1 LRS	EA2 LRS	EA1 PS	EA1 TS	EA1 $\mu\lambda$	EA1 GR
c50	Cust. not ins.	416.3 ± 1.3	385.0 ± 0.1	414.4 ± 1.2	415.8 ± 1.7	417.7 ± 1.3	490.1 ± 1.3
	Rel. perf.	100	92.5	99.5	99.9	100.3	117, 7
c75	Cust. not ins.	240.1 ± 0.7	240.6 ± 0.6	241.3 ± 0.8	241.0 ± 0.4	241.1 ± 1.0	220.1 ± 0.8
	Rel. perf.	100	100.2	100.5	100.4	100.4	91.7
c100	Cust. not ins.	252.6 ± 0.5	257.5 ± 0.4	250.3 ± 0.3	250.5 ± 0.3	250.0 ± 0.5	229.6 ± 0.4
	Rel. perf.	100	101.9	99.1	99.2	99.0	90.9
c100b	Cust. not ins.	412.8 ± 0.6	392.0 ± 0.1	404.4 ± 0.7	410.9 ± 0.5	409.7 ± 0.5	382.9 ± 0.7
	Rel. perf.	100	95.0	98.0	99.5	99.2	92.7
c120	Cust. not ins.	371.8 ± 0.3	360.0 ± 0.1	367.3 ± 0.3	367.9 ± 0.3	369.9 ± 0.3	421.9 ± 0.2
	Rel. perf.	100	96.8	98.8	99.0	99.5	113, 5
c150	Cust. not ins.	482.5 ± 1.4	490.6 ± 0.2	482.9 ± 1.0	480.5 ± 1.0	480.2 ± 1.3	527.9 ± 1.2
	Rel. perf.	100	101.7	100.1	99.6	99.5	109.7
c199	Cust. not ins.	384.3 ± 2.5	386.0 ± 0.1	385.0 ± 2.5	385.7 ± 2.9	385.4 ± 2.1	363.4 ± 2.1
	Rel. perf.	100	100.4	100.2	100.4	100.3	94.6

instance chosen, would be best to take the TS or PS method.

To make more general claims on how these methods perform, the average relative performances on both heuristics and EA selection methods are presented in Table 3. Note that the heuristics are compared to the base case of the *NoWait* heuristic, while the EA selection methods are compared to the base case of the LRS-method for EA1 that was used in Branke et al. (2005).

Table 3: Average relative performances of heuristics and selection methods

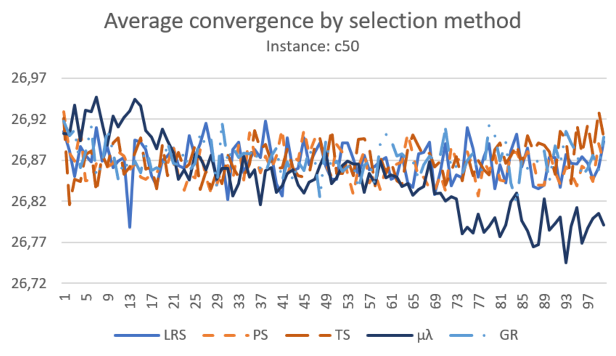
Waiting Strategies	Relative performance
NoWait	100.0
Depot	154.0
MaxDistance	107.9
Location	98.7
Distance	100.2
Variable	96.2

Selection Method	Relative performance
EA1 LRS	100.0
EA2 LRS	99.0
EA1 PS	99.5
EA1 TS	99.7
EA1 $\mu\lambda$	99.8
EA1 GR	101.5

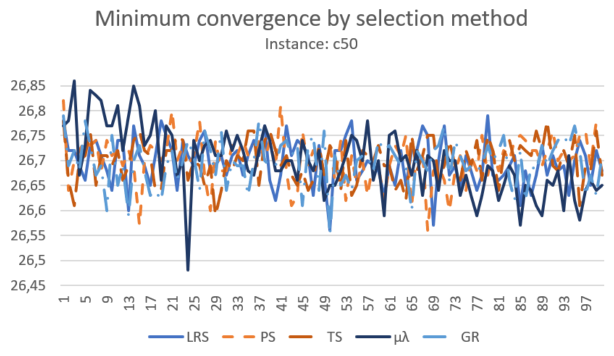
As can be seen from Table 3, the heuristics *Location* and *Variable* perform on average better than *NoWait*. In Branke et al. (2005), these heuristics are also the best performing, but there

Variable also performs better than *NoWait*. Also, *MaxDistance* performs worse, while it does not in Branke et al. (2005). Moreover, we do not have such a benchmark for the selection method. It is clear that the PS, TS, and $\mu\lambda$ perform slightly better than the LRS method that was used in Branke et al. (2005), but the benefit is minimal. The GR method that was introduced in this thesis, performs on average slightly worse than LRS. However, its performances fluctuate more than the other methods. As we have seen in Table 2, the GR method sometimes performs significantly better than the others, making it a risky but valuable method. Another interesting observation from Table 3 is that EA2 performs best, suggesting that the influence of the initial population of the EA is higher than the selection method that influences the choice of parents for the next generation. Lastly, preliminary results have shown that running the selection methods other than LRS on EA2 do not provide unexpected engaging results.

Other than considering the final solutions only, it is also of interest if and how the convergence of the EA solutions is achieved. To study this question, all solution evaluations were stored, generating 20 rows of 100 values for each selection method. To summarize this data, the progress of the average and minimum of the 100 emanated realizations of all 20 simulations are depicted in line graphs. All line graphs can be found in Appendix B.



(a) Average convergence



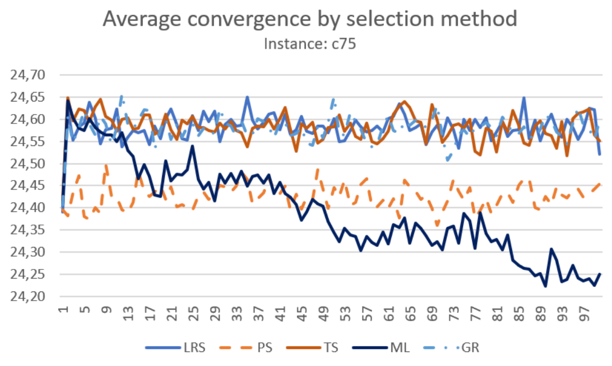
(b) Minimum convergence

Figure 1: Convergence by selection method - instance c50

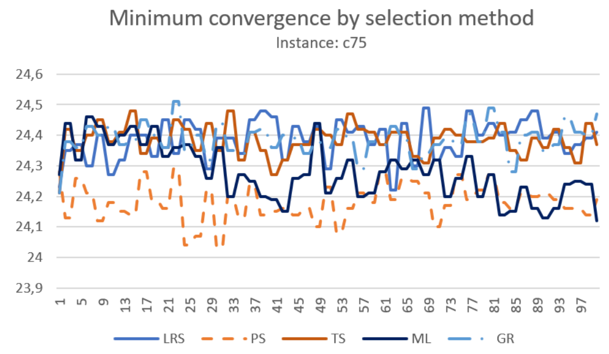
In Figure 1a, much information on the average performance of selection methods for instance c50 is released. The most remarkable observation is that all methods follow an average constant trend, not improving much over time, except for the $\mu\lambda$-method. This $\mu\lambda$-method outperforms the other methods by far in terms of development results. The other selection methods perform fairly similarly and move up and down between 26.82 and 26.92 customers. In that sense, it seems strange that this method does not perform much better on the set of 1,000 test customers as presented in

Table 2 with which we evaluate all solution methods. However, this observation can be clarified by looking at Figure 1b. There, we see the evaluation of the minimal (and thus best) performance of all methods over the generations.

Here, it is seen that the minimum values fluctuate over the generations for all methods, but no method outperforms the others. Since the final evaluation of each simulation is done on the best (i.e. minimal) solution, it is hereby visible that the $\mu\lambda$ method performs similarly to the other methods. However, the other but best solutions of the $\mu\lambda$ generations are improving over time, decreasing the average performance. This difference is most likely caused by the reduction in population size over the generations, because the $\mu\lambda$ method does not keep storing bad solutions as the other methods do, influencing the averages. In the other methods, the bad solutions are penalized with low probabilities, such that the chances of them influencing the final outcome (i.e. minimal values) are low, but they still influence the average values. Thus, for the other methods, both the average and minimal results perform similarly over time, not booking significant improvements. As can be seen in Figure 7 and 8 in Appendix B that the same conclusions hold for instance c100. Instance c120 can also be included in this group, as can be seen from Figure 11 and 12; and c150 as from Figure 13 and 14. Lastly also c199 fits in well, as can be seen from Figure 15 and 16.



(a) Average convergence



(b) Minimum convergence

Figure 2: Convergence by selection method - instance c75

The convergence results for instance c75 are presented in Figure 2a and 2b and are analyzed here. It is clearly visible that some results in these pictures differ substantially from c50. In Figure 2a, we can see that the LRS, GR and TS methods again perform similarly and move up and down between two bounds, following a constant trend. The PS method also follows a constant average trend but does so on a significantly lower, and thus better level. The cause of this seems to be that this is the only method that does not jump up 0.25 customers in the first iteration. Unfortunately, no explanation was found for this significant jump of all the other methods. The $\mu\lambda$-method again shows an obvious downward slope, ending up even lower than the PS method, although it also jumps up high in the first iteration.

In Figure 2b we can see that the minimal values converge similar to the average values. The graph implies that methods $\mu\lambda$ and PS perform similarly and do better than the other methods. However, these results do not agree with those in Table 2, which shows that all methods perform similarly and LRS does best. A possible explanation for this is that the differences are not substantial in both cases. In Figure 2b, the objective values differ only 0.2% in the 100th generation. In Table 2, we also see marginal differences of 0.1% between the methods. It is thus very likely that the differences in findings are caused by the randomness of the evaluation method, which generates a new set of 100 customers for each generation and uses another set of 1,000 customers to evaluate the final solution. These conclusions can also be applied to instance c100b to a great extent, because of the similarity of the c75 graphs and the c100b graphs presented in Figure 9 and 10.

6 Conclusion

In this thesis, a specific dynamic variant of a VRP was investigated, in which a route is performed and the truck driver waits at several customers to increase the chance of being able to serve an extra customer that appears at an in advance unknown place at an unknown time. This problem is documented in the literature as the Waiting Drivers Problem. Aiming to find a fitting waiting strategy, first, the work of Branke et al. (2005) was implemented and commented on. In his work, several heuristics and two EAs were suggested. Even though the results did not match perfectly, the overall trends were similar. This thesis contributed to research of the problem, implementing and comparing several selection methods for the EA.

Summarizing, the heuristics mostly perform similarly as in Branke et al. (2005). The heuristic *Variable* functions best overall, but is overruled by *Distance* for one instance. The heuristics *Depot* and *MaxDistance* are observed to have the worst evaluation statistics. It was also found that EA2 performs better than the wide variety of selection methods applied on EA1, meaning that the initial population influences the final performance of an EA more than the selection method applied to choose parents. Moreover, the selection methods mutually demonstrate very similar results. On average, the PS, TS, and $\mu\lambda$ method perform slightly better than the LRS method. From these results, it can be concluded that the selection pressure of parents for an EA as introduced in Back (1994) does not have much influence on our problem instances. However, it was shown that the $\mu\lambda$ method displayed its selection pressure in a coherent way, getting rid of the worst solutions in every iteration. Unfortunately, this did not lead to significantly better results. The GR method performs worse than LRS on average, but has a bigger scope of solutions, making it a bad method for a part of the instances and a good method for the other instances.

Lastly, the behavior of selection methods over time is noisy, but the trend is constant, except for the $\mu\lambda$ method which slopes downward, probably because of the reduction in population size over time. On the whole, it can be concluded from these empirical results that there lie advantages in choosing other waiting strategies than the one proposed by the reference strategy *NoWait*, because it was shown that the number of customers that could not be served can be reduced as much as 20%, for instance in the case of EA1 in c75. Mostly a reduction of 3 to 7% can be achieved, which is still a beneficial amount.

For further research, several ideas lie yet open to explore, which are out of the scope of this thesis. Firstly, the EA2 as described in Branke et al. (2005) could be more extensively tested with the selection methods other than the Linear Ranking Selection method. As we found very low standard errors for the simulations of EA2 for each instance, we have concluded that the heuristics used as an initial population here, are quite influential since the results are quite different from EA1. EA2 with LRS was shown to perform better than EA1 in combination with other selection methods. Secondly, in this research, we focused on implementing 1 customer per day, as was done in Branke et al. (2005). However, it would be interesting to see how results differ if we implement more than one customer every day, having waiting times and appearance times of customers influence each other. Thirdly, more research on the performance of the GR method is suggested. As we have seen in Section 5, its performance fluctuates heavily. It would be interesting to have some kind of performance measure for this method to know in advance whether it would be profiting to apply that method on a certain problem instance. Unfortunately, this type of further research was out of scope in this thesis. Fourthly, more research on the convergence and other behavior of selection probabilities or evaluation measures through the generations could lead to interesting results, since in this thesis the line graphs show that most EAs yet reach convergence in a few iterations. Lastly, a more decent understanding of the influence of adding randomness to solutions would probably be very relevant, to control more consciously for the genotypic diversity, which holds a trade-off with selection pressure. Although much research is yet to be done, this research has contributed to the field of EA applications. Within the scope of this research, it was empirically concluded that selection methods do not influence the EA results much in all cases, but can be very prominent, like the Golden Ratio method.

acknowledgements The thesis writer thanks Y.N. Hoogendoorn for supervising this thesis in a very pleasant and approachable way. Also, she would like to thank dr. T.A.B. Dollevoet for organizing the thesis course in such a clear and structured way, and also for being compassionate. Lastly, thanks to the second assessor dr. P.C. Bouman, who spent time validating the thesis result.

References

Back, T. (1994). Selective pressure in evolutionary algorithms: A characterization of selection mechanisms, 57–62.

Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. *101*, 111.

Beasley, J. (n.d.). Distributing test problems by electronic mail [j1. *Journal of the Operationail Reseach Society. 1993I4l: l069—1072.*

Bertsimas, D. J., & Van Ryzin, G. (1991). A stochastic and dynamic vehicle routing problem in the euclidean plane. *Operations Research*, *39*(4), 601–615.

Bertsimas, D. J., & Van Ryzin, G. (1993). Stochastic and dynamic vehicle routing in the euclidean plane with multiple capacitated vehicles. *Operations Research*, *41*(1), 60–76.

Bianchi, L. (2000). Notes on dynamic vehicle routing–the state of the art.

Branke, J. (1999). Memory enhanced evolutionary algorithms for changing optimization problems. *3*, 1875–1882 Vol. 3. https://doi.org/10.1109/CEC.1999.785502

Branke, J., Middendorf, M., Noeth, G., & Dessouky, M. (2005). Waiting strategies for dynamic vehicle routing. *Transportation science*, *39*(3), 298–312.

Branke, J., & Schmeck, H. (2003). *Designing evolutionary algorithms for dynamic optimization problems*. Springer.

Chambers, L. D. (2000). *The practical handbook of genetic algorithms, applications, 2nd edition*. CRC Press.

Cuevas, E., Enriquez, L., Zaldivar, D., & Perez-Cisneros, M. (2018). A selection method for evolutionary algorithms based on the golden section. *Expert Systems With Applications*, *106*, 183–196.

Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. *Management science*, *6*(1), 80–91.

Eiben, A. E., & Schoenauer, M. (2002). Evolutionary computing. *Information Processing Letters*, *82*(1), 1–6.

Eiben, A. E., Smith, J. E. et al. (2003). *Introduction to evolutionary computing* (Vol. 53). Springer.

Escuin, D., Larrode, E., & Millan, C. (2016). A cooperative waiting strategy based on elliptical areas for the dynamic pickup and delivery problem with time windows. *Journal of Advanced Transportation*, *50*(8), 1577–1597.

Fleischmann, B., Gnutzmann, S., & Sandvoß, E. (2004). Dynamic vehicle routing based on online traffic information. *Transportation science*, *38*(4), 420–433.

Flood, M. M. (1956). The traveling-salesman problem. *Operations research*, *4*(1), 61–75.

Fogel, L. J. (1965). *Artificial intelligence through a simulation of evolution.*

Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex systems*, *3*(5), 493–530.

Holland, J. (1973). Genetic algorithms and the optimal allocation of trials. *SIAM journal on computing*, *2*(2), 88–105.

Holland, J. (1975). Adaptation in natural and artificial systems: An introductory analysis with application to biology. *Control and artificial intelligence*.

Huning, A. (1976). Evolutionsstrategie. optimierung technischer systeme nach prinzipien der biologischen evolution.

Koza, J. R., & Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection* (Vol. 1). MIT press.

Laplace, P. S. (1820). *Theorie analytique des probabilites*. Courcier.

Luke, S. (2015). Essentials of metaheuristicsl.

Michalewicz, Z. (2013). *Genetic algorithms+ data structures= evolution programs*. Springer Science & Business Media.

Mitrovic-Minic, S., Krishnamurti, R., & Laporte, G. (2004). Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, *38*(8), 669–685.

Montemanni, R., Gambardella, L. M., Rizzoli, A. E., & Donati, A. V. (2005). Ant colony system for a dynamic vehicle routing problem. *Journal of combinatorial optimization*, *10*(4), 327–343.

Necker, M. (1998). An introduction to evolutionary algorithms.

Novoa, C. M. (2005). *Static and dynamic approaches for solving the vehicle routing problem with* stochastic demands*. Lehigh University.

Pillac, V., Gendreau, M., Guéret, C., & Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research*, *225*(1), 1–11.

Psaraftis, H. N. (1988). Dynamic vehicle routing problems. *Vehicle routing: Methods and studies*, *16*, 223–248.

Reinartz, W., Wiegand, N., & Imschloss, M. (2019). The impact of digital transformation on the retailing value chain. *International Journal of Research in Marketing*, *36*(3), 350–366.

Schwefel, H. (1981). *Numerical optimization of computer models*. Wiley, Chichester.

Vonolfen, S., & Affenzeller, M. (2016). Distribution of waiting time for dynamic pickup and delivery problems. *Annals of Operations Research*, *236*(2), 359–382.

Wilson, N., & Colvin, N. (1977). Computer control of rochester dial-aride system.

Zhang, B.-T., & Kim, J.-J. (2000). Comparison of selection methods for evolutionary optimization. *Evolutionary Optimization*, *2*(1), 55–70.

Zhou, A., Kang, L., & Yan, Z. (2003). Solving dynamic tsp with evolutionary approach in real time. *2*, 951–957.

Appendix

A Derivation of probabilities on random string length

In this section, the probability distribution on both the starting point and length of a substring is derived, taking into account that these are mutually dependent. Here, both the length $1 \leq k < n$ and the starting point $1 \leq p \leq n - k + 1$ of the substring will be taken randomly with $k, p \in \mathbb{Z}$. Note that we choose a strict substring here, excluding $k = 0$ and $k = n$. First k is chosen randomly and then choose p given k. Given the string size of the solution, k and p are dependent on each other. This influences the probability distribution of substrings chosen, which we want to be uniformly distributed. We do this by assigning each k a probability of $\frac{2n-2k+2}{n^2+n-2}$, which is derived below.

In the following intuitive derivation, a discrete uniform probability density function is obtained for retrieving substrings randomly from a string. Given a string of size $n > 0$, we choose a substring of size $1 \leq k < n$ where all positions $1 \leq p \leq n - k + 1$ are taken into account as different substrings and each substring has an equal chance of being chosen. Here, $n, k, p \in \mathbb{Z}$ To derive the probabilities that that should be assigned to each k given n, Laplace's law of probability in Laplace (1820) is used, which states that 'the probability of an event is the ratio of the number of cases favorable to it, to the number of all cases possible when nothing leads us to expect that any one of these cases should occur more than any other, which renders them, for us, equally possible.'[1]

To apply Laplace's law, we first calculate the total possible number of substrings of a word. First recognize that there are n substrings of length 1 (i.e. k = 1). For k = 2 there are n-1 substrings, for k = 3 only n-2 et cetera, where k¡n. Thus, in total the number of possible substrings $s_n = n + n - 1 + n - 2 + ... + 3 + 2$. To calculate this sum, we derive:

$$\begin{aligned}
s_n &= s_{n-1} + n \\
&= s_{n-2} + n - 1 + n \\
&... \\
&= 2 + 3 + ... + n - 1 + n \\
&= \sum_{i=2}^{n} i \\
&= \sum_{i=1}^{n} i - 1 \\
&= \frac{n(n+1)}{2} - 1 \\
&= \frac{1}{2}n^2 + \frac{1}{2}n - 1
\end{aligned}$$

[1]The original French quote was written in Laplace (1820), Livre II, p.183: 'La probabilité d'un événement futur, tirée d'un événement observé, est le quotient de la division de la probabilité de l'événement composé de ces deux événements, et déterminée a priori, par la probabilité de l'évenement observé, déterminée pareillement a priori.'

Now it can be seen that the number of different substrings differs with the length of the substring. For k, there are $n-k+1$ different substrings possible. Thus, intuitively, a smaller value for k must be chosen more often, since more different starting positions p are still available than for larger substrings, yielding more variations of substrings.

Applying Laplace's law, the following probability equals $\mathbb{P}(k) = \frac{n-k+1}{\frac{1}{2}n^2+\frac{1}{2}n-1}$, which can be reduced to

$$\mathbb{P}(k) = \frac{2(n-k+1)}{n^2+n-2}.$$

B Results: line graphs of convergence

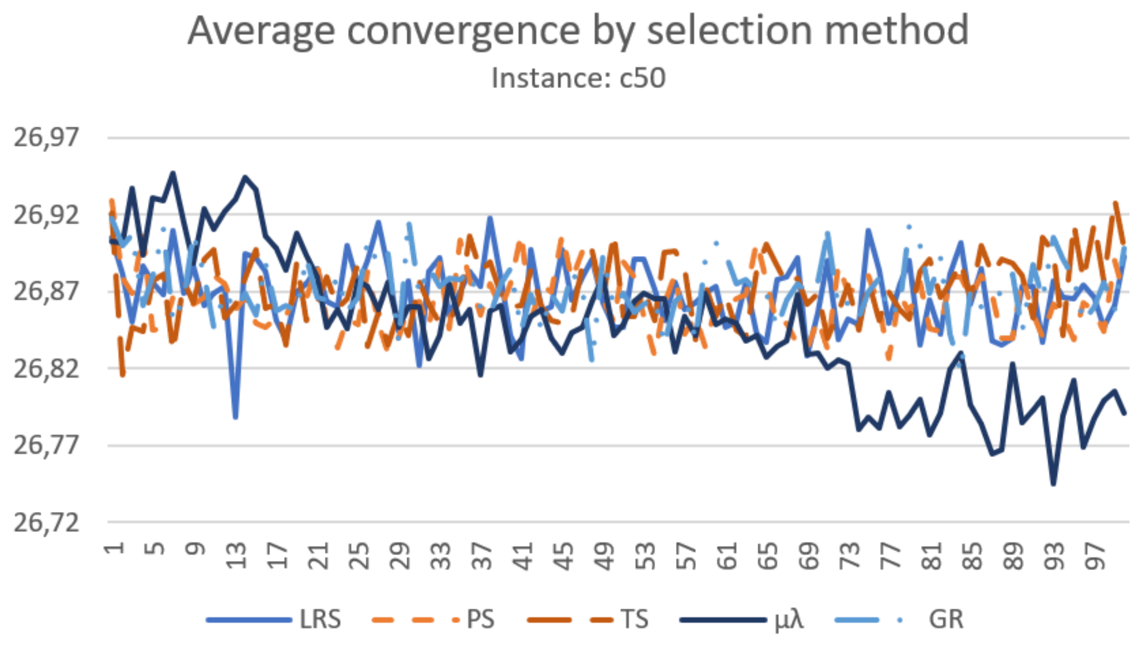


Figure 3: Average convergence by selection method - instance c50

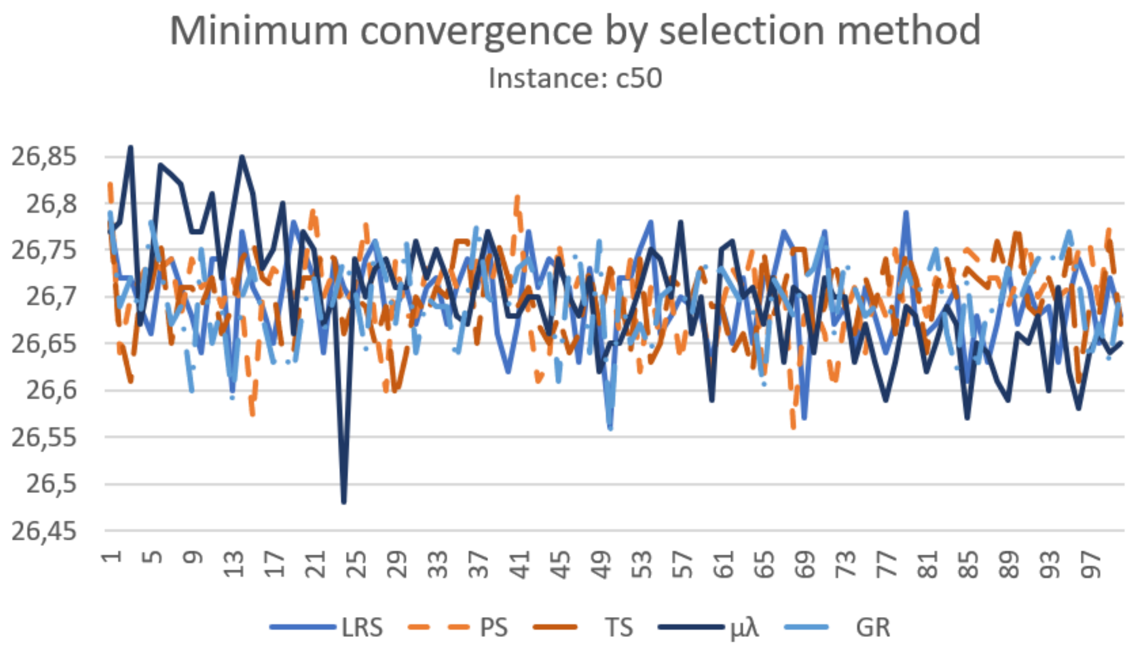


Figure 4: Minimum convergence by selection method - instance c50

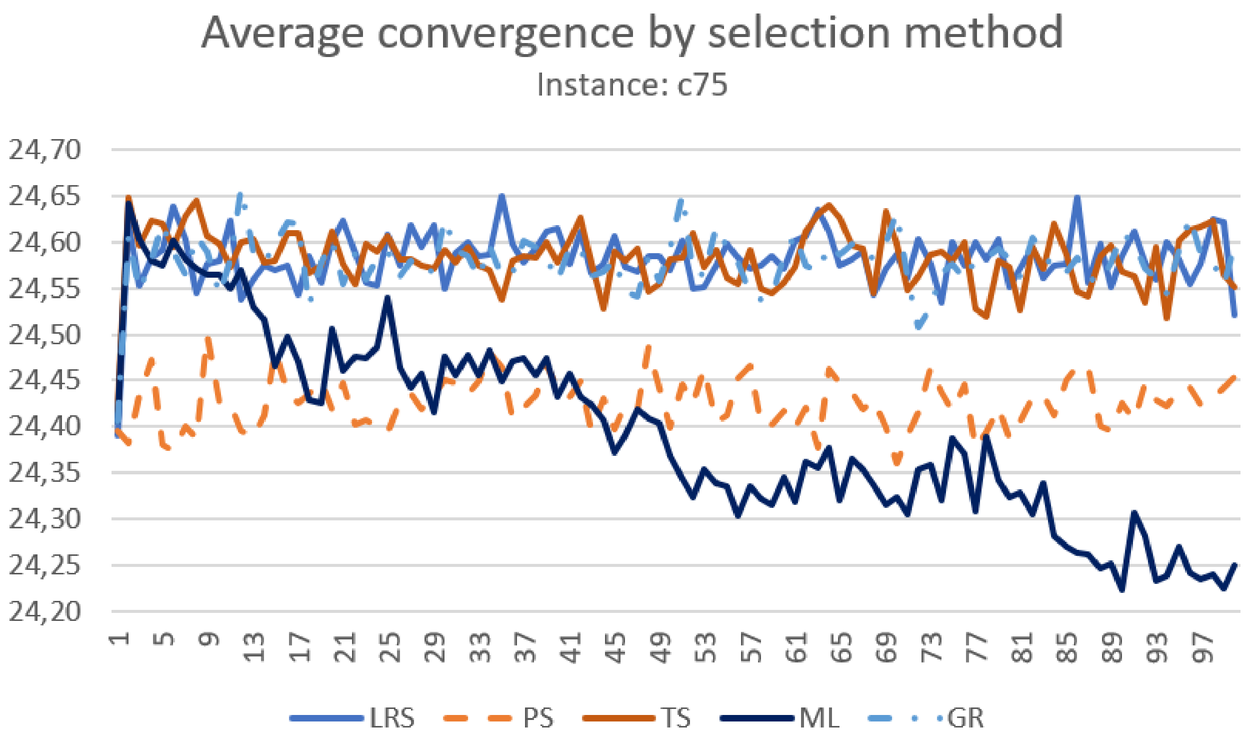


Figure 5: Average convergence by selection method - instance c75

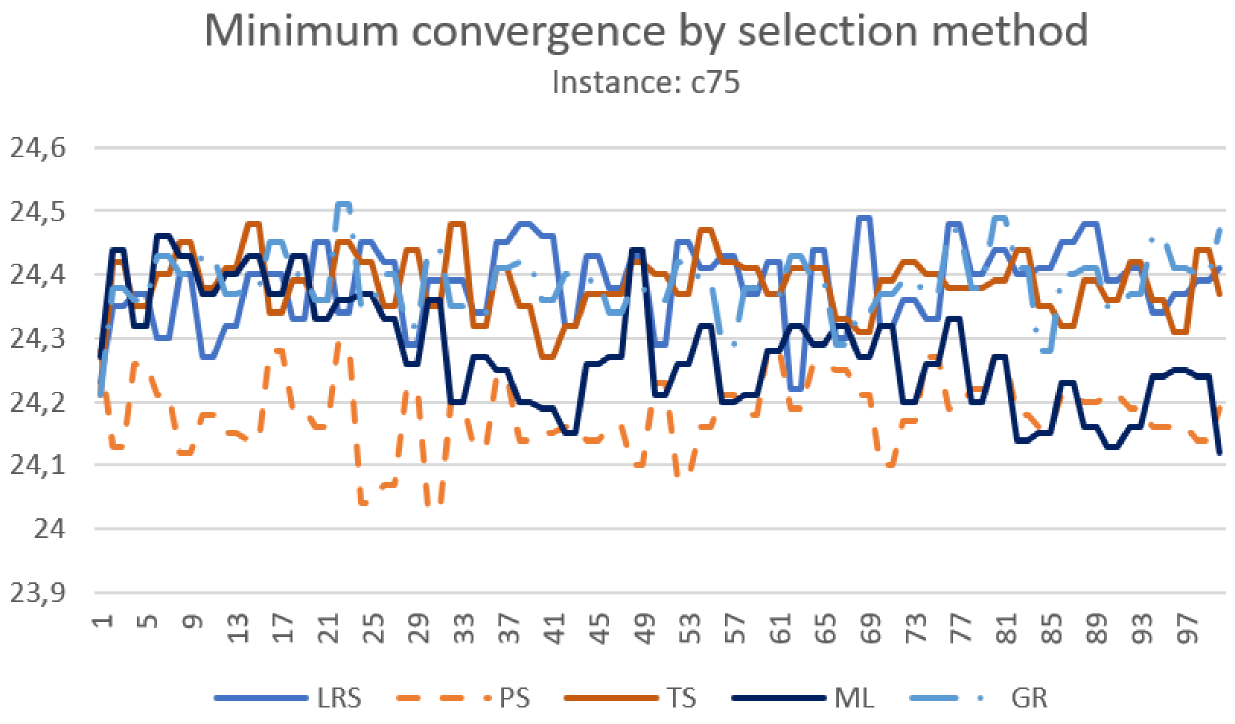


Figure 6: Minimum convergence by selection method - instance c75

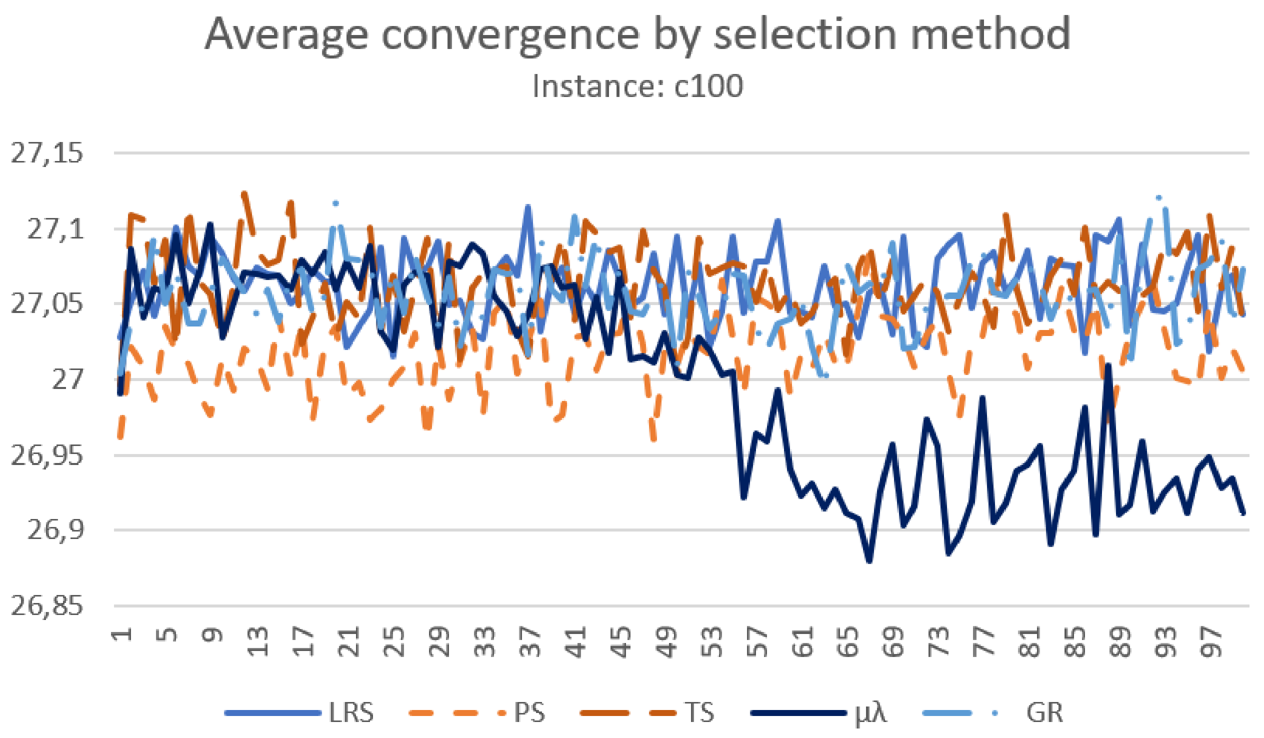


Figure 7: Average convergence by selection method - instance c100

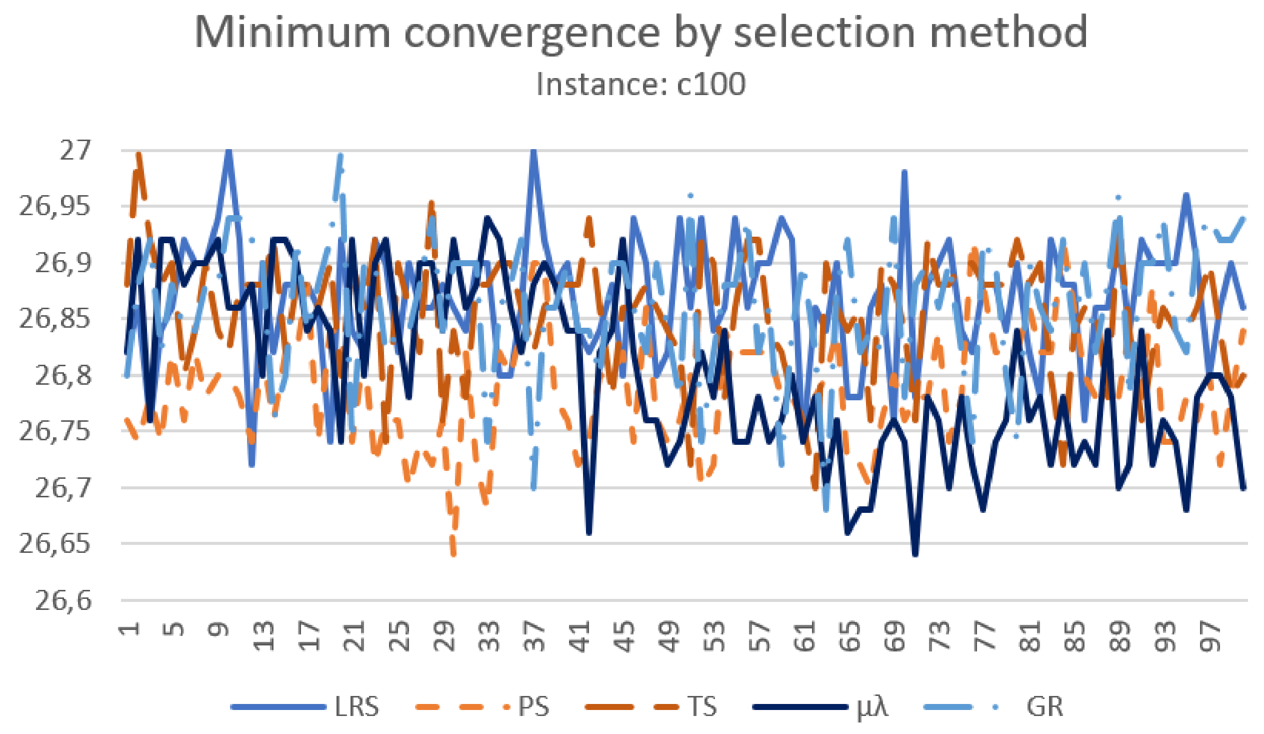


Figure 8: Minimum convergence by selection method - instance c100

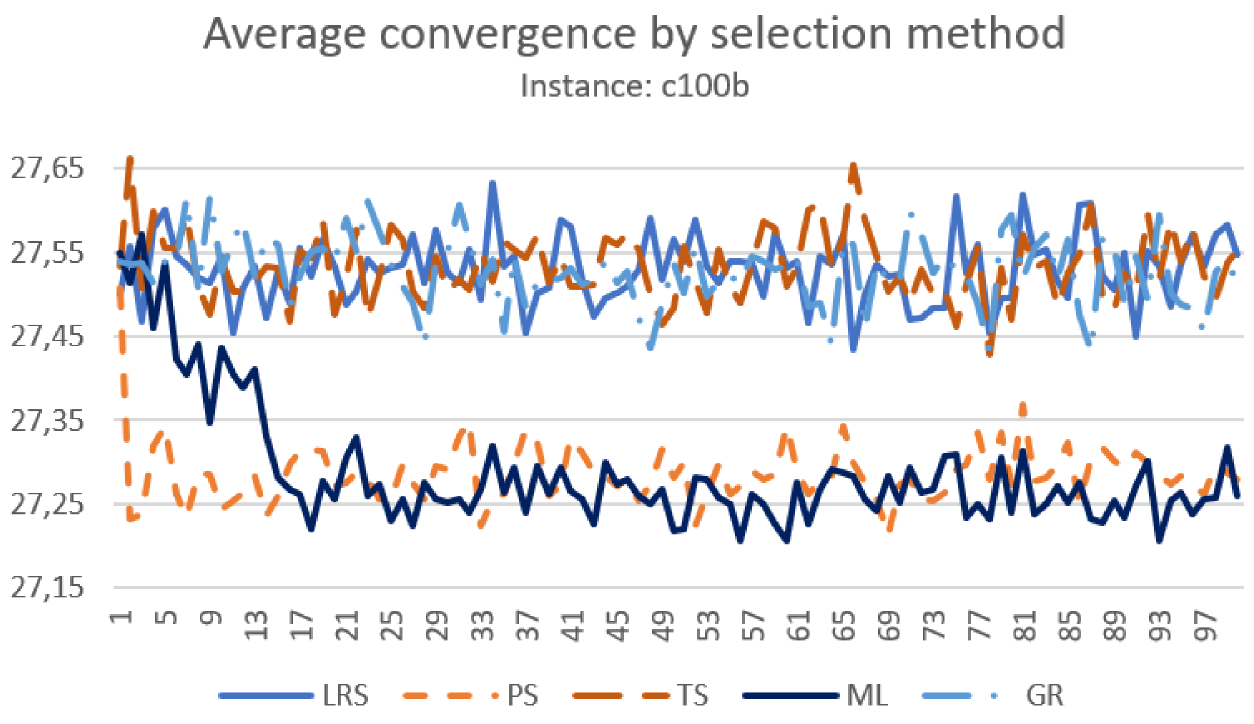


Figure 9: Average convergence by selection method - instance c100b

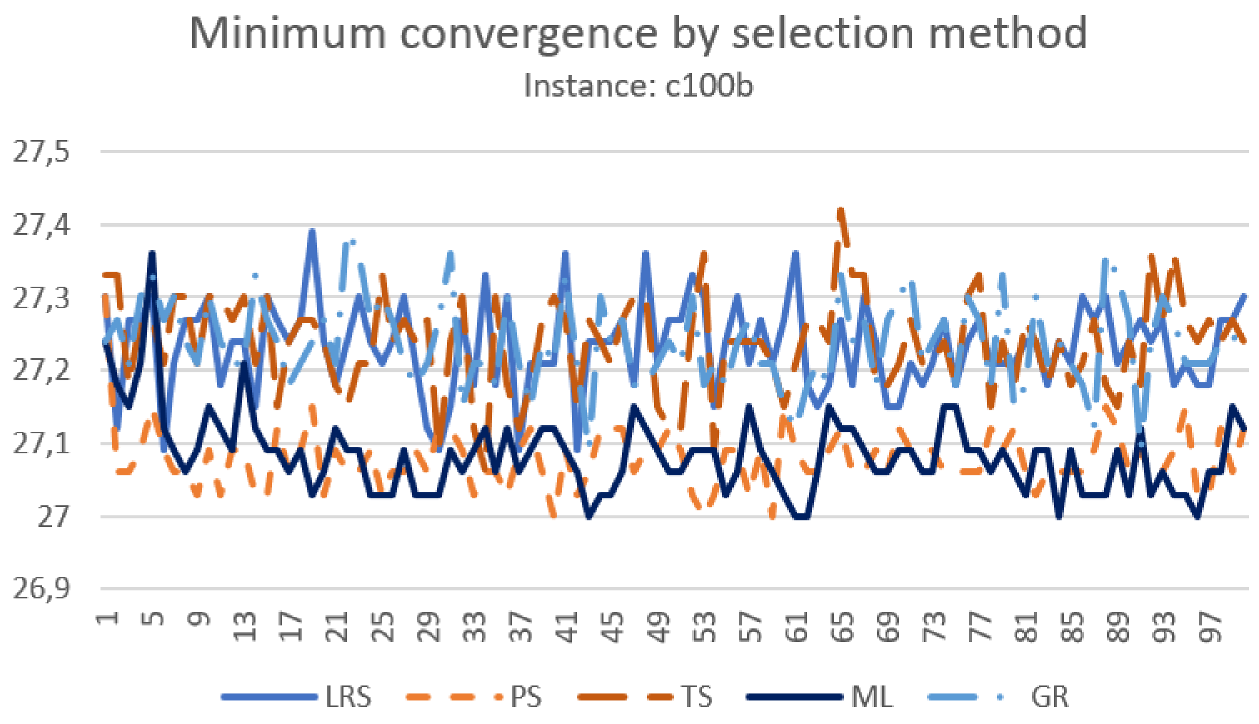


Figure 10: Minimum convergence by selection method - instance c100b

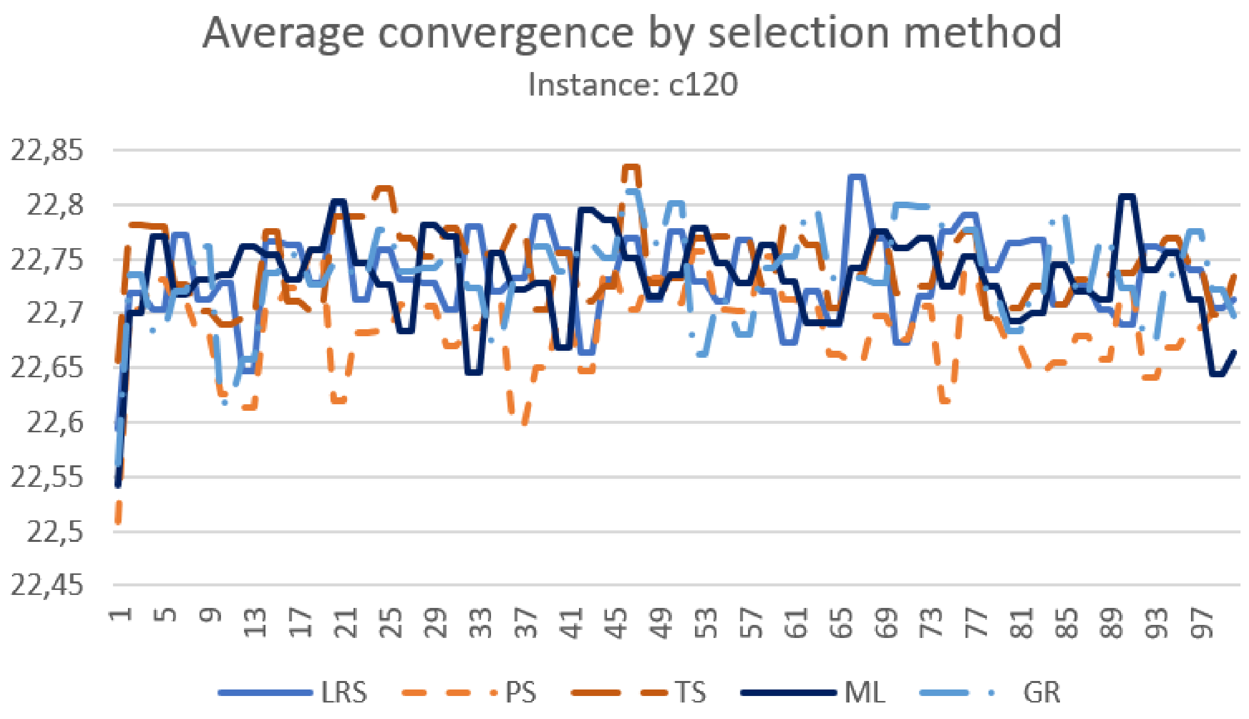


Figure 11: Average convergence by selection method - instance c120

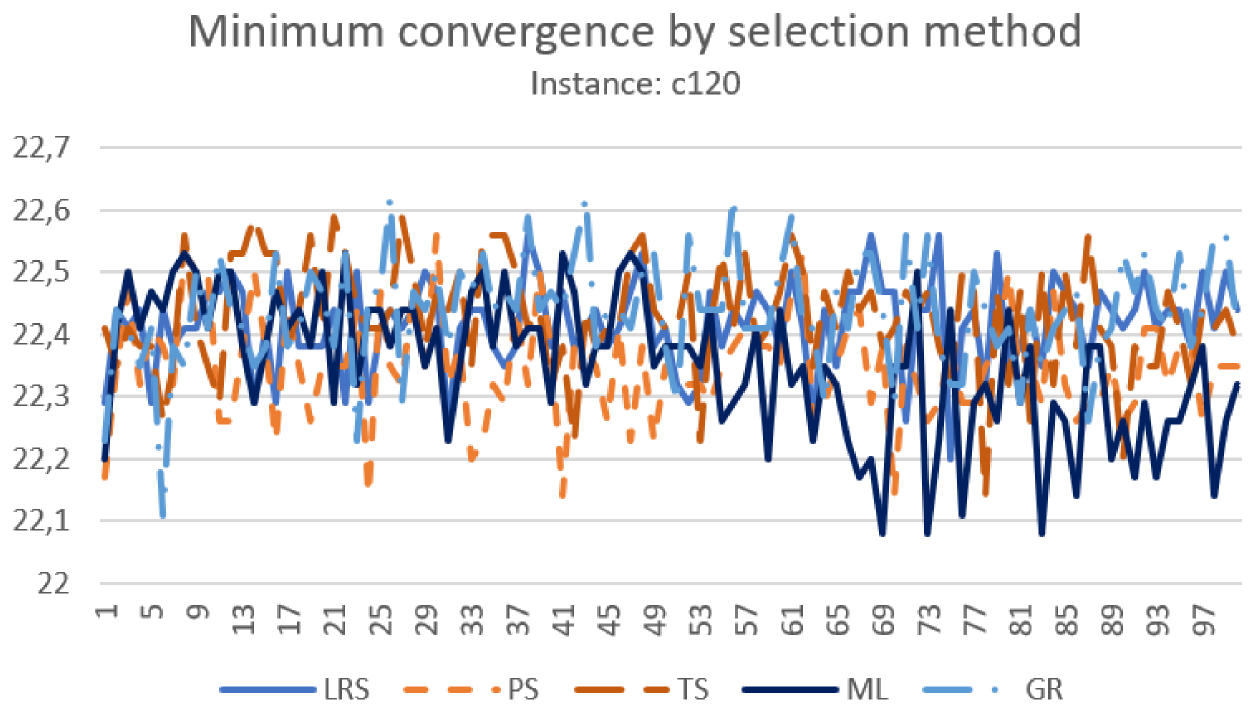


Figure 12: Minimum convergence by selection method - instance c120

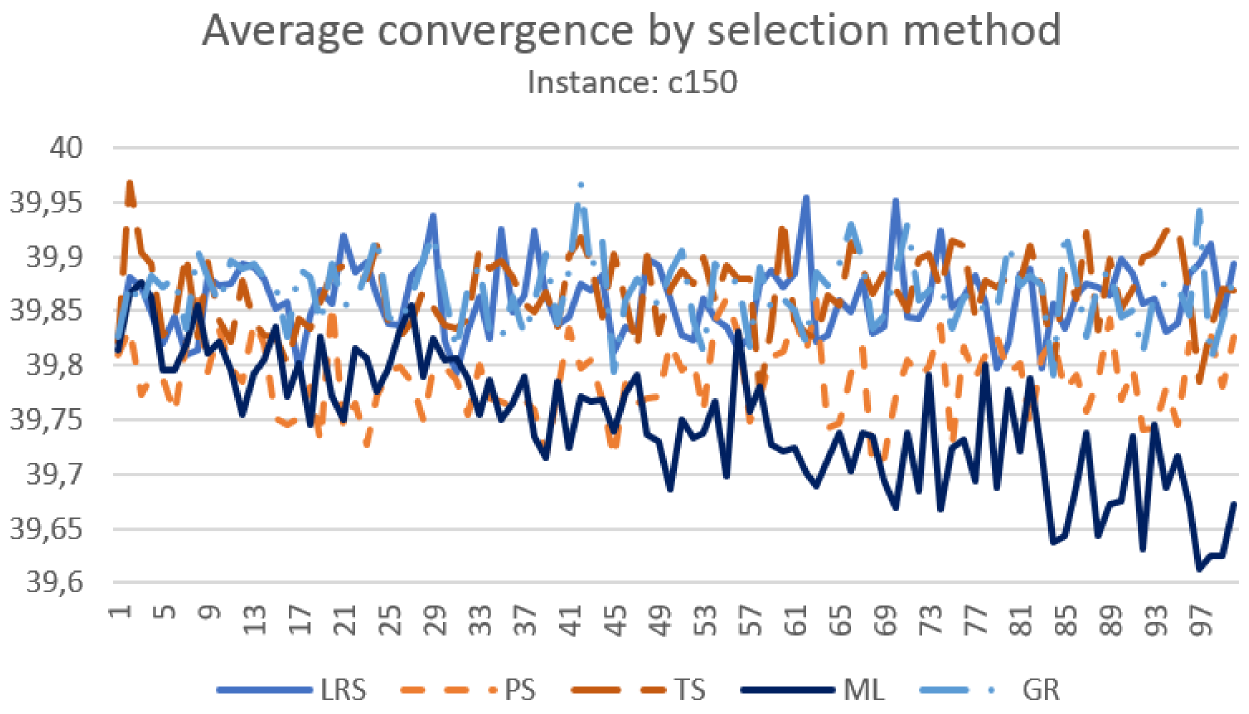


Figure 13: Average convergence by selection method - instance c150

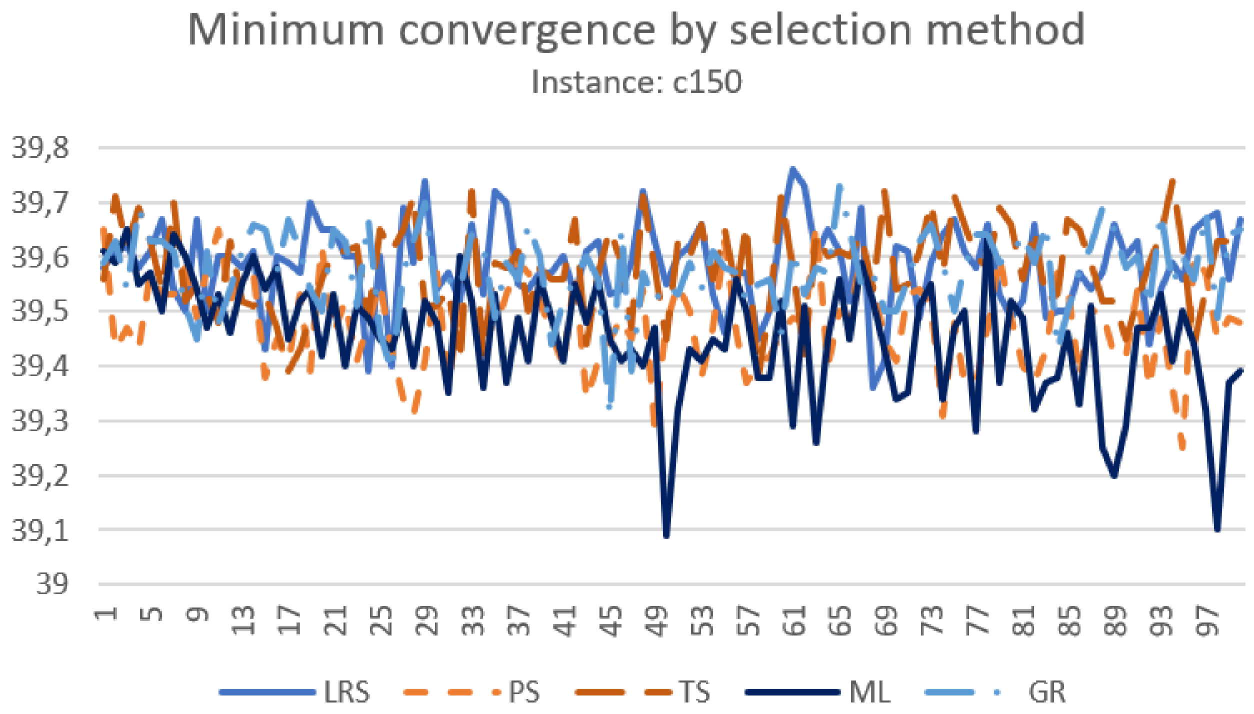


Figure 14: Minimum convergence by selection method - instance c150

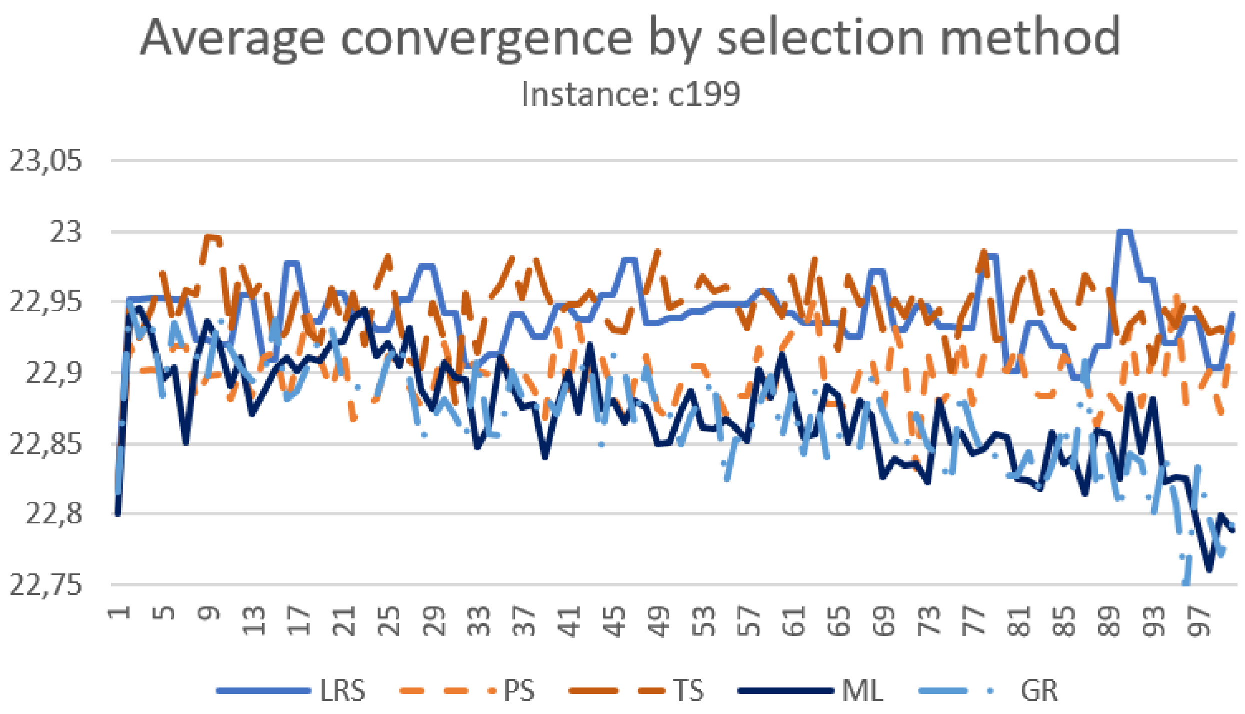


Figure 15: Average convergence by selection method - instance c199

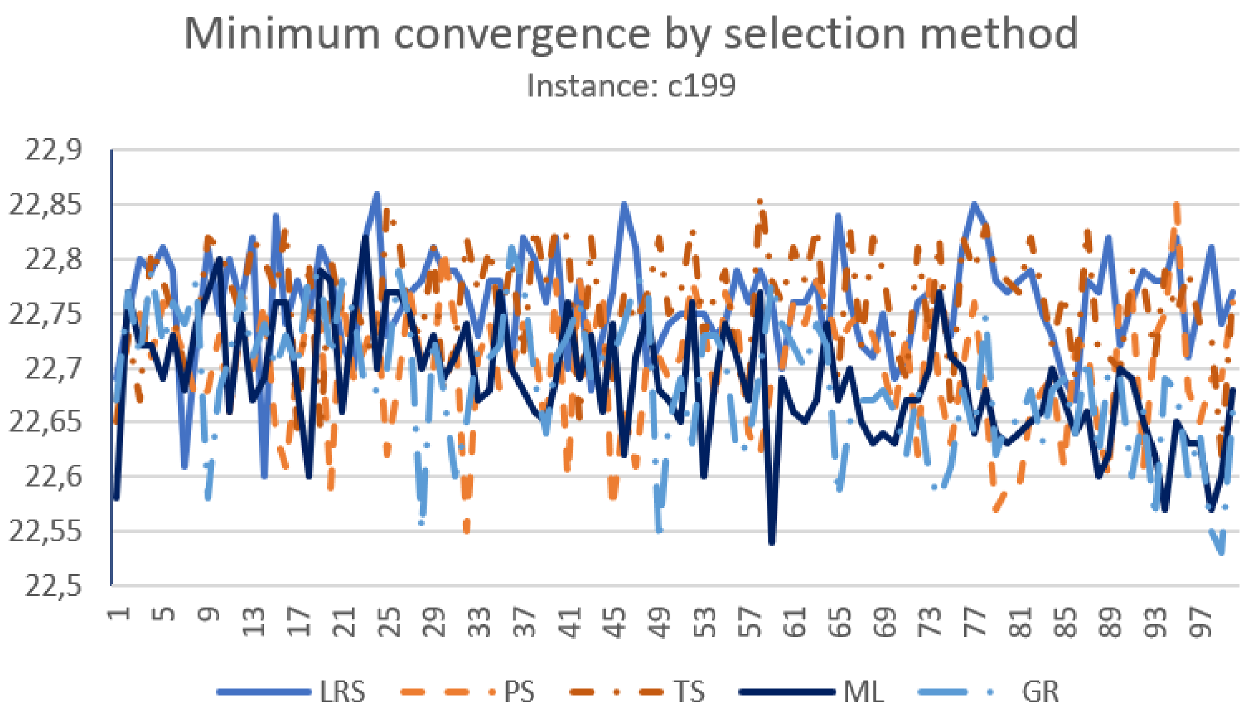


Figure 16: Minimum convergence by selection method - instance c199

C Description of coding

Main class In the Main class, a main method calls all methods that are necessary to run the program. First, the initialization methods of the abstract data structure and the reading of data are called. Then, all parameters are set. In the main method, one has to choose whether to run the simulation based on one of the heuristics or on the Evolutionary Algorithm. Also, a self-made test case is still visible but was not used in running the real cases.

The read method is first called and reads the data from a text file, which is slightly adapted to make reading it easier. The adaptions that were done are the following:

- any double spaces were replaced by single spaces;
- if there was a space in front of one of the first three lines, it was deleted;
- for all the routes, the first number was deleted if that number presented the number of customers in that route, which was the case in most files, but not in all;
- for all the routes, the last number was deleted if that number presented the solution, which was the case in some files, but definitely not in all;
- if present, the second set of routes was deleted since it was not used in our problem and has to do with capacitated problems.

In the reading method, the minimum and maximum x and y values of the customers were also obtained.

After some getter and setter methods, two distance methods are programmed which retrieve distances between two customers or between a set of coordinates and a customer. These distances are retrieved from a distance matrix that was set up in the method calcDistances, fed by the read x- and y-coordinates of the customers. A horizon is set to the length of the longest route in the setHorizon method. Also, a printer method was programmed, to be able to print results back into a comma-separated text file.

Lastly, a setWaitingTimes method was implemented which calls other methods in the Heuristics class to set waiting time proportions in a double array for each route. This results in a list of double arrays, with one double array for each route. For one of these heuristics, it is necessary to know which customer is located the farthest from the depot, whose index is obtained in the farthestIndex method and inserted as an argument for that particular heuristic.

Heuristics The Heuristics class is called from the main class and contains six methods presenting the six heuristics that are implemented in this paper. Also, two normalizing methods are added, such that it can be made sure that the waiting time fractions always add up to one, although this was already accounted for by the definitions of the heuristics themselves.

Route The Route class is a class that summarizes and keeps track of the distinct data for each route. For instance, the tour visits are kept track of and an array of accumulated distances is stored. Also, the slack time for each route is calculated here.

Customer The Customer class stores information on random initiated customers. As three randoms are given as arguments, a random occurrence time, a random x coordinate, and a random y coordinate are obtained and stored.

EvaluationSimulation The EvaluationSimulation class is meant for evaluation purposes, which checks for a certain number of random customers whether they can be serviced at their personal random occurrence time within the route and used waiting strategies. For this goal, one method accumulates the timestamps, adding the traveled distances and the waiting times (i.e. waiting fractions times the slack time) such that it is clear where all vehicles are at a certain point in time. The customerCanFitIn method checks for one customer if it fits by checking for all routes one by one, returning true as soon as it fits in one route within the given time. To check if it fits, it first is found out how far the truck driver has driven as the new customer request arrives and from which index we (almost) departed. We only need to decide where we will deviate from our predetermined route from that index because the customer did not show up earlier. If this index is equal to the last index in the route, it means that we are back at the depot and we can test if we can visit the new customer in the time that we are left with waiting. Otherwise, we first check how far we are away from the last customer that we visited, then we find out what the distance is that we are currently traveling. These two values are used as a fraction to find out at which exact x and y coordinates the driver is currently. To find out when it would be best to deviate and service the new customer, the best detour is first set to deviating immediately. Then, all possible detours by deviating from one of the upcoming customers are checked and the best detour is updated if it is shorter than the current one. The left distance to travel, including the best detour, is then calculated and finally, it is checked whether the left distance could be traveled from now on, within the time horizon. If so, the customer could be serviced by this particular vehicle. Otherwise, the other vehicles are checked in the same way and if none can service the customer, false is returned. For the Evolutionary Algorithm, this was done with 100 customers in each simulation run. For the final evaluation of both the Evolutionary Algorithm and the heuristics, it was checked for 1,000 customers whether they would fit in.

Evolutionary Algorithm In this class, first a population of starting solutions is obtained, either from the EA1 or EA2 method. For each generation, the improvement method is called, which first evaluates the previous, then selects the best parents for the new population. From this new population, parents for child creation are selected by one of the selection methods that all have their own method of creating a cumulative selection probability double arrays. The parents are then selected and crossover and mutation are performed so that a child is born. This is repeated to however many children we need. Then all children are evaluated in the EvaluationSimulation and these child solutions are ranked by the linearSortRanking method. Then, again a new population is being formed and a new generation can be obtained.