Simulation-based Optimization of a Real-world Travelling Salesman Problem Using an Evolutionary Algorithm with a Repair Function

Anna Syberfeldt *anna.syberfeldt@his.se*
Engineering Science
University of Skövde
Skövde, SE-54148, Sweden

Joel Rogström *joel.rogstrom@his.se*
Engineering Science
University of Skövde
Skövde, SE-54148, Sweden

André Geertsen *andre.geertsen@his.se*
Engineering Science
University of Skövde
Skövde, SE-54148, Sweden

Abstract

This paper presents a real-world case study of optimizing waste collection in Sweden. The problem, involving approximately 17,000 garbage bins served by three bin lorries, is approached as a travelling salesman problem and solved using simulation-based optimization and an evolutionary algorithm. To improve the performance of the evolutionary algorithm, it is enhanced with a repair function that adjusts its genome values so that shorter routes are found more quickly. The algorithm is tested using two crossover operators, i.e., the order crossover and heuristic crossover, combined with different mutation rates. The results indicate that the order crossover is superior to the heuristics crossover, but that the driving force of the search process is the mutation operator combined with the repair function.

Keywords: Evolutionary Algorithm, Simulation-based Optimization, Travelling Salesman Problem, Waste Collection, Real-world Case Study.

1. INTRODUCTION

In this paper, we present a study of optimizing waste collection from households in Sweden. The study was undertaken in collaboration with the AÖS (www.aos.skovde.se) waste management organization. AÖS is responsible for collecting household waste in seven municipalities in West Sweden from a total of approximately 120,000 bins. Each bin is emptied once or several times during a 14-day period according to a predefined schedule. Currently, the bin lorry routes and their scheduling are specified manually by a transportation planner. Manually producing efficient routes and schedules is very difficult due to the vast number of bins combined with the many parameters to be considered. Parameters include driver working hours and breaks, truck capacity (in terms of both weight and volume), and time windows at offload facilities. Due to the complexity of producing efficient routes and schedules, AÖS has expressed a need to improve its current process by introducing an optimization tool, motivating this study.

We approach the waste collection problem as a travelling salesman problem, defined as finding the shortest possible route visiting each existing node exactly once and finally returning to the starting node [1]. Finding the shortest route between several nodes might seem simple, but is classified as NP-hard in its simplest form [2]. With an NP-hard problem, the time needed to solve

the problem grows exponentially with the problem size – in this case, the number of bins. Finding the optimal route is possible, but might take a very long time because all possible routes must be evaluated to find the best one. Even small travelling salesman problems involve a huge number of possible routes. For example, for a problem with only 15 bins to visit, there are 6,227,020,800 possible routes. One can easily understand that manually optimizing the waste collection problem is virtually impossible, at least within a reasonable timeframe.

Due to the complexity of routing problems, optimization techniques that are not guaranteed to find the optimal solution, but a sufficiently good one in a short time, are often used rather than exact methods [3]. Evolutionary algorithms are a class of such inexact optimization techniques usable to approach routing problems [4], and an evolutionary algorithm is used here. In the study, we enhance an ordinary evolutionary algorithm with a repair function that considerably improves the optimization performance. Unlike repair functions previously suggested in the literature, the suggested repair function does not aim to transform invalid into valid solutions but focuses solely on improving performance. To ensure valid solutions, a non-destructive crossover operator is used instead of a destructive one.

To evaluate the solutions generated by an evolutionary algorithm, a mathematical function representing the travelling salesman problem is usually used. In this study, we instead use a simulation that models the real-world waste collection scenario to evaluate solutions in order to ensure that the obtained results are valid in practice. We believe that that the combinatorial relationships, uncertainty factors, and nonlinearities present in the waste collection scenario – and in real-world scenarios in general – are far too complex to be effectively modelled analytically

In the next section, the concepts of evolutionary algorithms and optimization using simulations, so-called simulation-based optimization, are described in detail for the convenience of readers unfamiliar with them. Section 3 outlines the simulation model developed for the waste collection scenario considered here. Section 4 describes the evolutionary algorithm used for optimizations and its repair function. Section 5 presents the optimization experiments and Section 6 finally outlines the conclusions of the study and presents possibilities for future work.

2. THEORETICAL BACKGROUND

This section presents the fundamentals of evolutionary algorithms (Section 2.1) and simulation-based optimization (Section 2.2).

2.1. Evolutionary Algorithms

The basic idea of evolutionary optimization is to use computational models of evolutionary processes in designing and implementing problem solving applications [5]. Based on Darwin's "survival of the fittest" concept, candidate solutions to a problem are iteratively refined. Generally, an evolutionary algorithm consists of a genetic representation of solutions, a population-based solution approach, an iterative evolutionary process, and a guided random search. The genetic representation defines an individual solution as a set of genes.

In evolving a population of solutions, evolutionary algorithms apply biologically inspired operations for selection, crossover, and mutation. The operators are applied in a loop, an iteration of the loop being called a generation. The following pseudo code presents the basic steps of this evolutionary process:

```
Initialize population
Evaluate the fitness of solutions in the population
repeat
  Select solutions to reproduce
  Form a new generation of the population through crossover and mutation
  Evaluate the new solutions
until terminating condition
```

The solutions in the initial population are usually generated randomly. In each generation, a proportion of the solutions in the population is selected to breed offspring for the next generation of the population. Solutions are selected based on their fitness, representing a quantification of their optimality. Solutions with high fitness typically have a higher probability of being selected, but to prevent premature convergence, it is common that a small proportion of solutions with worse fitness is also selected. From the solutions selected, new solutions are created to form the next generation of the population. To create each new solution, two parent solutions are chosen and, through mating (called crossover), an offspring is produced (Figure 1).

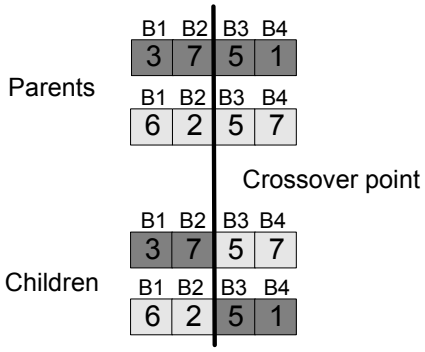

FIGURE 1: Example of a Crossover.

Occasionally, the new solution can undergo a small mutation in order to maintain high diversity in the population and avoid local minima. A mutation usually involves changing an arbitrary part of a solution with a certain probability, for example, slightly changing the size of the buffer in Figure 2.

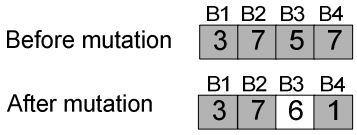

FIGURE 2: Example of a Mutation.

When the new population is formed, the average fitness will have generally increased. The process of evolving generations continues until a user-defined termination criterion has been fulfilled, for example, that the best solutions in the last *n* generations have not changed or that a certain time has passed.

2.2. Simulation-based Optimization

In simulation-based optimization, the evolutionary algorithm uses a simulation model to evaluate the fitness of solutions [6]. The simulation model is a representation of the real-world scenario that mimics reality as closely as possible, including stochastic events as well. Simulation-based optimization is an iterative process (Figure 3): the evolutionary algorithm generates a solution (in this case, a set of routes) and feeds it to the simulation model, which computes the fitness (in this case, the driving time). Based on the evaluation feedback obtained from the simulation model, the optimization algorithm generates a new set of parameter values and the generation–evaluation process continues until a user-defined stopping criterion is satisfied. Such a criterion may, for example, be that a certain amount of time has passed or that specific objective values have been achieved.

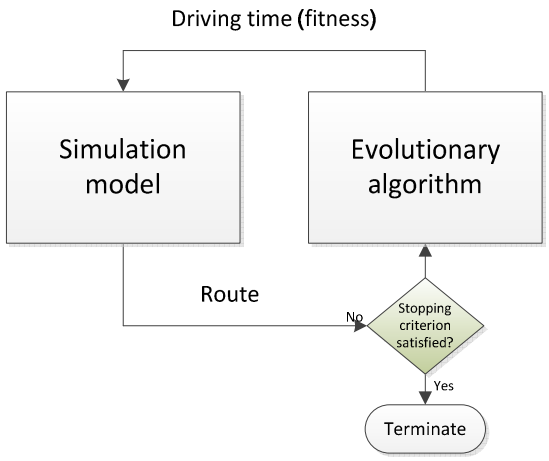

FIGURE 3: Simulation–optimization Process.

The next section describes the simulation model developed for the waste collection scenario considered here.

3. SIMULATION MODEL

To ensure that the optimized routes are valid in practice, a simulation is built that mimics the real-world scenario as closely as possible. The simulation model is based on data from the Swedish national road database, which holds information about all roads in Sweden. The information is quite detailed and covers not only the length and speed limit of each road but also, for example, traffic rules, road width, and whether the road is surfaced with asphalt or gravel. The database also contains geographical information that makes it possible to render the roads graphically. Although the Swedish national road database is huge, it is straightforward to use as it follows an open standard that defines how to read and extract data [7].

The next two subsections outline the basic structure of the simulation model (Section 3.1) and describe the graph used to navigate the roads in the model (Section 3.2).

3.1. Basic Structure

To represent the roads, the simulation model uses a road network. The road network comprises two main elements, nodes and links. Nodes are the logical endpoints that connect links to each other, while a link is a logical representation of a road. A link owns a set of link parts each of which describes the link at different moments in time. If changes are made to a road, the link is never changed; instead, the underlying link parts are replaced with newer parts that describe the link from that moment on. Each link part owns a time interval that describes when this link part is active.

To connect the nodes, links, and link parts, ports are used. A port is simply a connection between two elements, and every node owns a set of node ports while every link owns a set of link ports. A node connects to a link through a connection between one of its node ports and one of the link's link ports. A link's various link parts connect to one another using the link's link ports. Furthermore, every link port holds information about its relative distance from the link's starting point. Note that a node does not have to connect to the end points of a link but only to a link port. This concept is illustrated in Figure 4, in which three nodes each connect to the same link. The concept of a link comprising its link parts is illustrated in Figure 5, in which the greyed-out link part represents an older part of the road that has been removed. The link still represents the same road, but its underlying representation has been changed. The figure also illustrates how the link parts are connected through link ports.

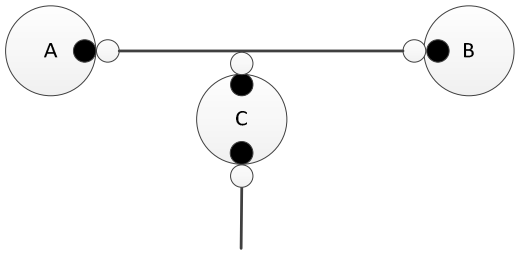

FIGURE 4: Three nodes connecting to the same link using the port system.

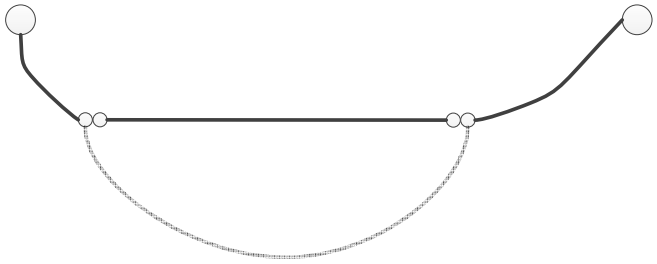

FIGURE 5: A link and its underlying link parts.

3.2. Navigation Graph

To navigate the road network, a navigation graph is used in the simulator. This navigation graph is a directed logical graph in which a set of nodes is connected to other nodes through edges. In implementation, an edge can only be connected to two nodes. The edge's beginning and ending nodes denote the direction of the edge. As each link is connected to a node through a port, it is straightforward to create edges between each port using the information contained in the link part. In other words, each link part is converted to an edge between two nodes in the navigation graph. This is illustrated in Figure 6, in which nodes A, B, and C are connected using edges instead of links and link parts. A directed graph is used because not all nodes are connected to each other in both directions. For instance, some roads allow traffic in only one direction. The direction of travel is also important as some bin lorries can only empty receptacles on the right-hand side of the road.

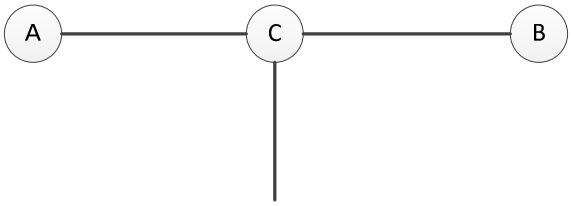

FIGURE 6: The same graph elements as in Figure 4, but in the navigation graph format.

An edge's identity is defined by its beginning and ending nodes, its underlying link, and the port ID used when connecting to its nodes. Using this approach makes it possible to separate two edges that connect the same nodes, as they are constructed from different links and ports. It also makes it possible to differentiate loops in different directions, as each loop will connect to different beginning and ending ports. Examples of both scenarios are illustrated in Figure 7, in which nodes A and B are connected through two different edges, and node A is connected to itself through a loop. As every node is connected to a link through a port, and each port stores its relative distance from the link's beginning and end, it is possible to calculate the length of each edge using the relative distance and the link's entire length.

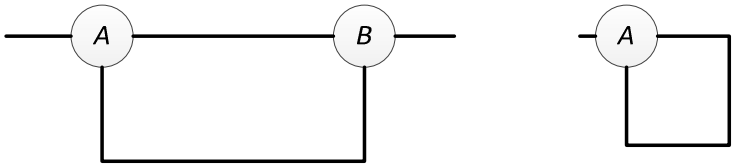

FIGURE 7: Examples of edge cases in the navigation graph.

The next section describes the evolutionary algorithm that uses the simulation model to evaluate generated routes.

4. EVOLUTIONARY ALGORITHM

This section describes the configuration of the evolutionary algorithm implemented to solve the optimization problem: first the genetic representation (Section 4.1), then the evolutionary operators (Sections 4.2–4.4), and finally the repair function (Section 4.5) is described.

4.1. Genome

To represent solutions, a combinatorial genome is used. Combinatorial genomes are commonly used when dealing with travelling salesman problems as they allow a simple, yet powerful, genetic representation of the problems [8]. In the implemented genome, each gene represents an edge to be visited by the bin lorry. All edges are visited in the order in which they appear in the genome, making the genome a direct representation of a route. This implies that all genes of the genome must be unique, and also that the crossover and mutation operators must never replace or remove any genes.

As the route has specific beginning and ending points, they must be represented in the genome and taken into account by the crossover and mutation operators. When building the genome, the first entry is therefore always the starting point and the last entry is always the ending point. To preserve these points, both the crossover and mutation operators operate on a subset of the genome, ignoring the first and last entries.

4.2. Selection Operator

As the selection operator, binary tournament selection [9] is chosen. In binary tournament selection, two lists are first created, each containing references to all individuals in the population. Both lists are then shuffled, in the present case using the Fisher–Yates shuffle method [10]. By comparing each element at the same index in both lists and choosing the best element, we select individuals for recombination. This selection operator is used because it is fairly simple, fast and easy to implement, and fair. The best solution in a population is guaranteed to be selected at least once and at most twice.

4.3. Crossover Operator

Two crossover operators are implemented and tested for the problem: the ordered crossover [11] and heuristic crossover [8]. Both operators are specifically designed for combinatorial problems and therefore only change the order of the genes, never removing values or introducing new ones into the genome. The ordered crossover is an older operator and is fairly simple. It is fast and efficient, but it operates in a black box manner and uses no problem-specific information. The heuristic crossover, in contrast, is more complex and uses problem-specific knowledge to guide the crossover towards better results.

The order crossover works as follows: Two crossover points are randomly chosen in the parents. The genes between the points in the first parent are copied into the second child, and the genes from the second parent are copied into the first child. The remaining genes not covered by the crossover are copied from the first parent into the first child in the order in which they appear. The same is done for the second parent and second child. An example of an order crossover is given in Figure 8.

Parent1	1	2	3	4	5	6
Parent2	4	6	5	2	3	1

Child1	1	6	5	2	3	4
Child2	6	2	3	4	5	1

FIGURE 8: Example of an order crossover.

The heuristic crossover (also known as the greedy crossover) uses a white box approach employing problem knowledge. The crossover includes the following steps: First, a random gene is chosen as the starting point for the crossover. The selected neighbouring genes in both parents are then identified, and thereafter any neighbours already present in the child are sorted out. If one or more neighbours remain, a heuristic is used to estimate the cost of travel between the selected gene and the remaining neighbours. The lowest-cost neighbour is added to the child's genome and is set as the next selected gene. If all neighbours of the selected gene exist in the child, one unvisited gene is randomly chosen as the next selected gene. The procedure is repeated until the genome has been filled and the child is complete. This complete process is repeated twice to produce two children.

4.4. Mutation Operator

As the mutation operator, the displacement operator [12] is used. This operator is well suited for combinatorial problems as it manipulates only the order of elements in the genome. The displacement operator works by randomly generating two offset points in the genome and then shifting all the elements between the two points in a randomly selected number of steps. The operator is demonstrated in Figure 9.

Displacement	1	2	3	4	5	6
	1	5	6	2	3	4

FIGURE 9: The displacement operator demonstrated.

4.5. Repair Function

A pattern emerged when implementing and testing the evolutionary algorithm. Initially, the routes resulting from the optimization often bypassed unvisited route stops on the way to other stops. This led to various weaving patterns in which a route would bypass an unvisited stop several times before it was finally visited (i.e., the bin was emptied). As it is not rational to drive past an unvisited stop, a constraint was considered specifying that the sum of bypassed unvisited stops must be zero. However, the act of detecting passed stops lends itself to fixing the problem, as we know that travelling between stops can entail bypassing at least one unvisited stop. Instead of adding a constraint, a solution was found in which the genome was reordered so that any unvisited stop bypassed when travelling between two stops would be placed between these stops. This approach was demonstrated to improve the performance of the optimization significantly.

The repair function is illustrated in Figure 10. In the example, the genome is in form [1, 5, 4, 6, 8], each number representing an edge in the graph. Asking the simulator for the shortest path between the first and second genes would return the following path: [1, 3, 4, 5]. By iterating through the path, we can find any unvisited stops bypassed. In the example given, the path bypasses edge number four. To fix this, we can rearrange the genome in such a way that we visit that edge before visiting edge number five. The genome after the fix would have the form [1, 4, 5, 6, 8] and the resulting route would be considerably shorter.

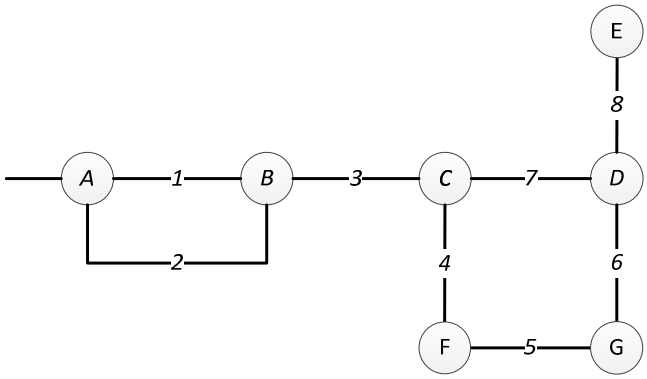

FIGURE 10: Graph illustrating the repair function.

It can be mentioned that repair functions have previously been described in several different studies in the literature (see for example [13-17]). The difference between the existing functions and the one suggested in this study is that the existing ones aim to transform invalid solutions into valid ones, while the suggested one aims to improve the general performance of the algorithm. The previously described repair functions are mainly used to compensate for a destructive crossover, while in this study a non-destructive crossover is used and the problem of invalid solutions is thereby eliminated.

The next section describes the results of optimizations using the evolutionary algorithm.

5. OPTIMIZATION

This section first describes the scope of the problem being optimized (Section 5.1), then presents the exact algorithm configuration (Section 5.2), and finally outlines the optimization results (Section 5.3).

5.1. Problem Scope

To test the optimization, one of the seven municipalities served by AÖS is selected. The selected municipality includes approximately 17,000 unique bins spread over an area of approximately 40 × 50 kilometres. For the reader to grasp the problem, the geographical locations of the bins are plotted on maps shown in Figures 11 and 12 (the latter zooms in on part of the former).

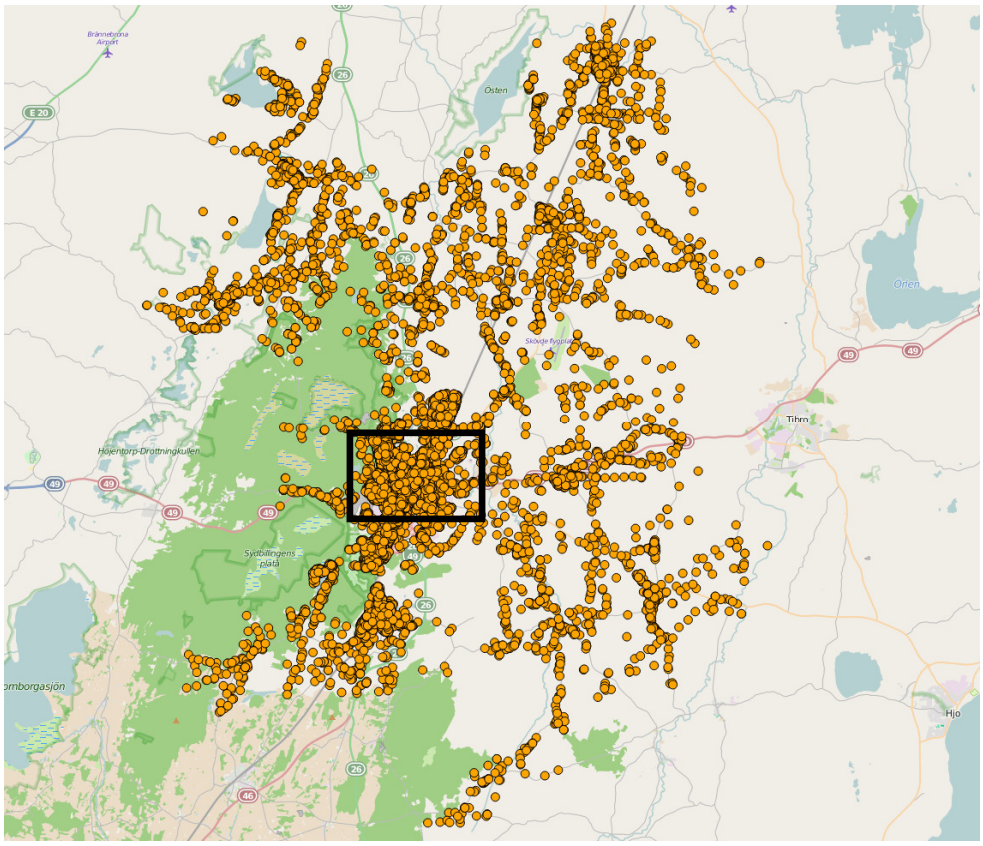

FIGURE 11: Garbage bins included in the optimization.

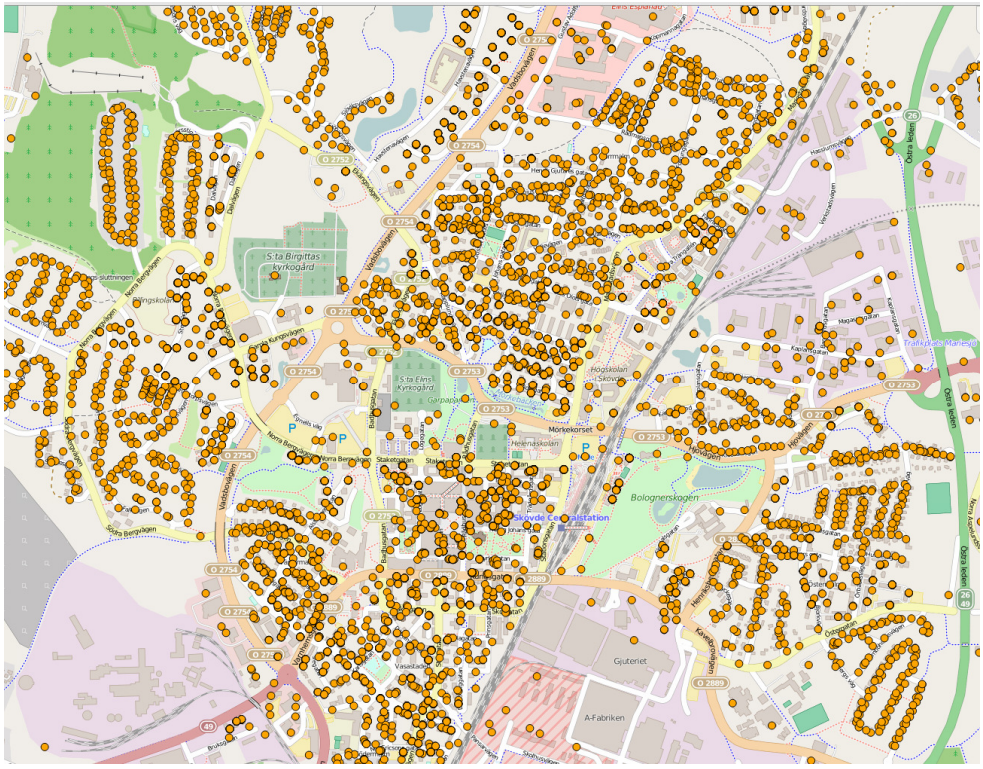

FIGURE 12: Zoomed-in view of the area corresponding to the black square in Figure 11.

Three trucks are available for emptying the selected bins, and each of them empties on average approximately 550 bins per day. This means that a unique route should be created for each truck and each workday during a 14-day period (excluding Saturdays and Sundays). If successful, the optimization might free one or several trucks for one or several days, which would save the waste management organization considerable money.

5.2. Algorithm Setup

The choice of parameters of an evolutionary algorithm has a distinct impact on the performance of the algorithm and thereby also on the final results [18, 19]. No single set of parameter values is universally optimal; rather, the parameters must be configured for each individual problem. In the study we therefore evaluate various settings of the crossover and mutation operators, as presented in the next subsection. In all presented experiments, the population size is set to 100 and the total number of iterations of the evolutionary algorithm is set to 10,000.

5.3. Results

The results of the experiments are presented in Table 1. Each combination of parameter values is tested ten times, the presented results representing an average of these ten runs. As can be seen in the table, the order crossover is superior to the heuristic crossover. It is also clear that the higher the mutation rate, the better the results. The reasons for and implications of these results are further discussed in Section 6. Due to lack of existing repair functions with equivalent focus, it has not been possible to perform benchmarking against any other solution previously presented in the literature. As mentioned in Chapter 4.5, existing repair functions have a completely different purpose than the suggested one and a performance comparison is therefore not meaningful.

Crossover	Mutation rate	Average minimum fitness	Average minimum iterations before local optimum is found
Order	1	2917.1	8920
Order	0.8	2938	8390
Order	0.9	2943.6	8460
Order	0.6	2958.1	9170
Order	0.7	2962.6	8490
Order	0.5	2977.8	9290
Order	0.4	3007.7	9380
Order	0.3	3034.4	8910
Order	0.2	3050.1	8930
Order	0.1	3160.8	9280
Heuristic	0.5	3163.2	6890
Heuristic	0.6	3167.5	3110
Heuristic	0.9	3168.6	5110
Heuristic	0.8	3182	5600
Heuristic	0.4	3194.5	4330
Heuristic	1	3211.5	5280
Heuristic	0.3	3218.7	1990
Heuristic	0.7	3223.1	6540
Heuristic	0.2	3230.6	2260
Heuristic	0.1	3249.6	3940
Heuristic	0	3260.4	610
Order	0	3885.9	100

TABLE 1: Results from experiments (ordered by average minimum fitness).

The routes resulting from the most successful optimization runs were presented to domain experts in the waste management organization, who carefully studied and evaluated them. The domain experts rated the improvement at about 25% when comparing the optimized routes with the routes created manually by the transportation planner, who can instead concentrate on other work tasks.

6. CONCLUSIONS AND FUTURE WORK

This study presented the optimization of waste collection, considering a problem including approximately 17,000 garbage bins. The problem was approached using an evolutionary algorithm and simulation-based optimization. To improve the performance of the evolutionary algorithm, it was enhanced with a repair function that adjusts genome values so that shorter routes are found more quickly. The algorithm was tested with two crossover operators, the order crossover and heuristic crossover, combined with different mutation rates.

The results of the optimization experiments clearly indicate that the order crossover produces much better results than does the heuristic crossover, at least as long as the mutation rate is above zero. This is likely because the heuristic crossover does not introduce into the genome changes as radical as does the order crossover. The order crossover, however, only works well when the repair function is used, as the operator introduces many changes into the genome. These changes might have negative effects, but these are immediately eliminated by the repair

function. The repair function is therefore of great importance and significantly influences the results. It can be noted that experiments with the repair function indicate that its importance increases with the size of the routes, i.e., the more points to be visited, the more value it produces.

Another interesting finding from the optimization experiments is that when the mutation rate is set to zero and the order crossover is used, the search stagnates after only 100 iterations. This indicates that the mutation rate has a much greater impact on the search than does the crossover, likely because the repair algorithm overwrites many of the changes made by the crossover. The repair algorithm tends to group together genes that are near each other, and as the mutation operator shifts groups of genes around in the genome, these genes can be assumed to work well together.

It is concluded from the study that the combination of mutation operator and repair function is the most important factor for rapid search progress and successful optimization results. In the future, it would be interesting to test additional crossover operators combined with the repair function to see whether the optimization results can be improved even further.

It would also be interesting to consider a dynamic rather than static approach to garbage collection. A problem is static if all transport requests are known when the routes are constructed [20], which was the case here. In a situation in which transport requests may be added while trucks are running the routes, meaning that routes have to be changed “on the fly”, the problem is instead dynamic. In reality, the problem is already dynamic as customers often overfill their bins and need immediate emptying outside the ordinary schedule. Dynamic problems are considerably more difficult to tackle as the data change constantly and the optimizations must be performed in real time, putting additional demands on the algorithm used.

7. REFERENCES

[1] R. Johnson and M. G. Pilcher. “The traveling salesman problem”, in Networks. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B Shmoys, eds. 18(3), 1988. 253–254.

[2] N. Jozefowiez, F. Semet, and E. G. Talbi. “Mulit-objective vehicle routing problems”, European Journal of Operational Research, vol. 189, pp. 293–309, 2008.

[3] M. Gendreau, J.-Y. Potvin, O. Bräysy, G. Hasle, and A. Lokketangen. “Metaheuristics for the vehicle routing problem and its extensions: a categorized bibliography”, in The Vehicle Routing Problem: Latest Advances.

[4] B. L. Golden, S. Raghavan, and E. A. Wasil, eds. New Challenges Operations Research/Computer Science Interfaces Series, vol. 43. New York: Springer, 2008, pp. 143–169.

[5] O. Bräysy, W. Dullaert, and M. Gendreau. “Evolutionary algorithms for the vehicle routing problem with time windows”, Journal of Heuristics, vol. 10(6), 587–611, 2004.

[6] K. Deb. Multi-Objective Optimization using Evolutionary Algorithms. Chichester, UK: John Wiley & Sons, 2008.

[7] J. April, M. Better, F. Glover, and J. Kelly. “New advances for marrying simulation and optimization”, in Proc. 2004 Winter Simulation Conference, 2004, pp. 80–86.

[8] Swedish Standards Institute. Geografisk information: Väg- och järnvägsnät – Applikationsschema [Geographical information: Road and railroad network – Application scheme], 2nd ed. Stockholm: Swedish Standards Institute, 2009.

[9] J. Grefenstette, R. Gopal, B. J. Rosmaita, and D. Van Gucht. “Genetic algorithms for the traveling salesman problem”, in Proc. 1st International Conference on Genetic Algorithms, 1985, pp. 160–168.

[10] D. E. Goldberg and K. Deb. “A comparative analysis of selection schemes used in genetic algorithms”, In Foundations of Genetic Algorithms. G. J. Rawlins, ed. San Mateo, CA: Morgan Kaufmann, 1991, pp. 69–93.

[11] D. Knuth. The Art of Computer Programming 2, 3rd ed. Boston, MA: Addison-Wesley, 1998.

[12] I. Oliver, D. Smith, and J. Holland. “A study of permutation crossover operators on the traveling salesman problem”, in Proc. Second International Conference on Genetic Algorithms and their Application, 1987, pp. 224–230.

[13] P. Larrañaga. “Genetic algorithms for the travelling salesman problem: a review of representations and operators”, Artificial Intelligence Review, vol. 13(2), 129–170, 1999.

[14] J.T. Richardson, M. Palmer, G.E. Liepins, and M. Hilliard. “Some Guidelines for Genetic Algorithms with Penalty Functions”, in Proc. Third International Conference on Genetic Algorithms, 1989, pp. 191-197.

[15] T. G. Bui and B. R. Moon, “A New Genetic Approach for the Traveling Salesman Problem,” in Proc. First IEEE Conference on Evolutionary Computation, 1994.

[16] T. Walters, “Repair and Brood Selection in the Traveling Salesman Problem”, in Proc. Fifth International Conference on Parallel Problem Solving from Nature—PPSN V. A.-E. Eiben, T. Back, M. Schoenauer, and H.-P. Schwefel, Ed. Springer, 1998.

[17] M.K. Kundu and N.R. Pal, “Self-crossover and its application to the traveling salesman problem”, in Proc. 12th international conference on Industrial and engineering applications of artificial intelligence and expert systems: multiple approaches to intelligent systems, 1999, pp. 326-332.

[18] S. Salcedo-Sanz, “A survey of repair methods used as constraint handling techniques in evolutionary algorithms”, Computer Science Review, vol 3(3), pp. 175-192, Aug 2009.

[19] M. Andersson, S. Bandaru, A. Ng, and A. Syberfeldt. “Parameter tuning of MOEAs using a nested optimization approach”, in Proc. 8th International Conference on Evolutionary Multi-criterion Optimization, 2015.

[20] S. Wessing, N. Beume, G. Rudolph, and B. Naujoks. “Parameter tuning boosts performance of variation operators in multiobjective optimization”, Parallel Problem Solving from Nature, PPSN XI, Lecture Notes in Computer Science, vol. 6238, 728–737, 2010.

[21] M. P. W. Savelsbergh and M. Sol. “The general pickup and delivery problem”, Transportation Science, vol. 29(1), 17–29, 1995.