Mat-2.4108 Independent research projects in applied mathematics

The Aalto University School of Science and Technology

Systems Analysis Laboratory

Scheduling and routing geographically distributed tasks with a genetic algorithm

Anni Nieminen Tfy 51658U
April 12, 2010
Contents

1 Introduction .................................................. 3

2 Genetic algorithms .......................................... 5

3 Routing and scheduling problems ............................ 7

4 Approaches to solving routing and scheduling problems .......... 11
   4.1 Exact methods ............................................... 12
   4.2 Traditional heuristic methods and local search .......... 13
   4.3 Metaheuristics ............................................. 14

5 Genetic algorithms in solving routing and scheduling problems .......... 15
   5.1 Chromosome design ....................................... 16
   5.2 Genetic operators .......................................... 16
   5.3 Constraint handling ...................................... 18
   5.4 Initial population ........................................ 18
   5.5 Experiences from solving routing and scheduling problems with genetic algorithms ......................... 19

6 Experimental part ............................................. 20
   6.1 The test problem .......................................... 20
      6.1.1 Integer Linear Program formulation ................. 22
   6.2 The algorithm ............................................. 24
      6.2.1 GA Framework ....................................... 24
      6.2.2 Chromosome ......................................... 24
      6.2.3 Fitness function ..................................... 26
      6.2.4 Selection ............................................ 26
      6.2.5 Genetic operators .................................... 26
      6.2.6 Constraint handling .................................. 27
      6.2.7 Local improvement ................................... 27
6.2.8  Initial population ................................................. 28
6.2.9  Replacing the population by next generation ............... 28
6.2.10 Summary of the algorithm ....................................... 28
6.3  Test cases ............................................................... 29
6.4  Results of the experiment .......................................... 30

7  Conclusions .................................................................... 35
1 Introduction

Routing and scheduling problems arise in a multitude of real-life situations in areas of transportation, distribution, manufacturing, warehousing and service sector systems. Better and faster scheduling and routing processes may bring about considerable cost savings to the user in form of minimized fleet size and fuel consumption, improved service level and reduced planning time. The mathematical complexity of the problems falling into this category makes them interesting also from theoretical point of view, and this together with the magnitude of potential cost savings in real-life situations has provided the area for a continuing interest of researchers for at least the past three decades.

The routing and scheduling problems considered in this work are combinatorial, \(NP\)-hard, and often large in realistic cases, which has been the motivation for developing heuristic optimization methods that can find near-optimum solutions in reasonable computational time, instead of trying to solve the problems to optimum. Metaheuristics are general problem solving strategies that use some standard method to explore the search space, and often utilize some traditional heuristic or exact algorithm to improve the found solution candidates. In this work we study closer one type of these metaheuristic methods, namely genetic algorithms, in context of solving routing and scheduling problems.

The two probably most studied combinatorial optimization problems form the basis for routing and scheduling problems, namely the Travelling Salesman Problem (TSP) and Vehicle Routing Problem (VRP). In TSP, a list of cities and the travelling cost between each pair of cities are given. The task is to determine a tour that minimizes the total travelling cost so that the salesman returns to the first city of the tour and each city is visited exactly once. In the VRP, the aim is to find optimal set of tours departing from and returning to a depot for a fleet of vehicles with given capacity to serve a given set of customers, when the demand of each customer and the travelling cost between each pair of customers are given. These two models are of practical use as such, and also form a base for huge amount of variations that have been driven by both practical needs and theoretical interest and can be attained by, for example, defining different objective functions, loosening the constraints, or introducing additional ones. From the point of view of its economic consequence, one of the most important extensions of the VRP is the VRP with Time Windows (VRPTW). In VRPTW, for depot and each customer release and due times are defined, during which the node must be entered. Other common extensions include giving up the constraint that all customer nodes must be visited, assigning each node with a reward to be gained when visited or a penalty when failed to visit, or varying the number of salesmen or vehicles used, for example. Different objectives can also be defined: minimizing the total travel cost, travel time, fleet size or penalty, maximizing the number of customers visited or rewards collected, balancing the route lengths between different vehicles, etc.

In this work, 'routing and scheduling problems' refers to problems derived from TSP and VRP. Moreover, only node routing problems are considered, as opposed to arc routing problems that also have their importance in both theory and practice, being the
basic models for routing and scheduling whenever the capacity or demand of the arcs instead of the nodes is in the focus. This work concentrates on the static versions of the scheduling and routing problems, where all the orders and constraints during the planning horizon are known in advance. With today’s possibilities for real-time data transfer, in many important applications this is not the case, and predicting the demand and/or dynamically adjusting the schedule often form important and challenging part of the planning process. These questions, however, are out of the scope of this work.

Both TSP and VRP belong to the class of \(\mathcal{NP}\)-hard problems, so no polynomial time exact algorithms are known to exist that solve them to optimality. From this follows, that while exact methods usually provide for fastest and most accurate solutions when available, in case of large instances of \(\mathcal{NP}\)-hard problems other methods must usually be applied. As most important real-life problems tend to be large, much research has been carried out concentrating on heuristic optimization methods, i.e. procedures, that produce good solutions in reasonable amount of computational time but with no guarantee of the optimality of the solution. The development and success of the most recent exact algorithms and metaheuristic algorithms has been following the vast increase of computational capacity of modern computer systems.

Genetic algorithms are a general problem solving approach inspired by the evolution and natural selection. Some research on computer simulation of evolution processes was conducted as early as in the 1950s, but the idea was first introduced as an optimization method in 1975 by Holland in the famous work *Adaptation in Natural and Artificial Systems* [25]. The idea of a GA is to provide an encoding of solutions and to create a 'population' of thus encoded candidate solutions called 'chromosomes'. The population of chromosomes is evolved by reproducing 'offspring' from the 'parent' solutions and recombining the features of the previous generations to create better solutions in the next ones. The parent’s of the next generation are selected mimicking the survival of the fittest by applying specifically defined 'fitness function' to the individuals of the population. A randomization process called 'mutation' is incorporated to prevent premature convergence to poor local optima and to enable searching previously unknown areas of the solution space.

The test problem studied in this work’s empirical part was freely derived from the problems existing in literature and can be interpreted as a variation of the Vehicle Routing Problem (VRP). The problem concerns scheduling a given number of staff to complete a set of geographically distributed tasks with time window and skill level constraints. The planning is conducted for a given number of days ahead. The objective was to form a schedule so that as many tasks as possible are completed within their given time limits, working hour restrictions are not violated, skill level demands are met, and the total distance travelled by the workers minimized. A genetic algorithm to solve the problem was implemented in Java and tested on data modified from the well known Solomon benchmark data sets for the Vehicle Routing Problem (VRP) [43]. The test problem was on purpose defined with some more than minimum amount of complexity to allow experimenting on the flexibility of the GA implemented to solve it.
This report is organized as follows. In chapter 2 we introduce genetic algorithms as a general problem solving procedure. In chapter 3 we have a view at the problem class under consideration, and in the following chapter 4 at the methods applied to solve these problems. In the fifth chapter, a survey on genetic algorithms as applied to routing and scheduling problems is given. Chapter 6 contains the experimental part of this work: The formulation of the test problem of this work is first given. Then, the genetic algorithm used to solve the problem is introduced in detail, and last the arrangement and results of the computational experiments carried out to test the algorithm are explained. Analysis of the results is also given there. Finally, chapter 7 concludes this work with some discussion.

2 Genetic algorithms

Genetic algorithms (GAs) are a general problem solving approach inspired by the evolution and natural selection. Some research on computer simulation of evolution processes was conducted as early as in the 1950s, but first time the idea was made widely known as an optimization method in the early 1970s by Holland and his colleagues at the University of Michigan [25].

The idea of a GA is to provide an encoding of solutions, called chromosomes, and to create a population of thus encoded candidate solutions. The population of chromosomes is evolved by reproducing offspring from the parent solutions and hopefully recombining the good features of the previous generations to create better solutions in the next ones. To decide any solution’s goodness, a fitness function must be defined. Mutation is applied on the individual chromosomes with a small probability to prevent premature convergence to poor local optima and to enable searching previously unknown areas of the solution space. Two important things to note about GAs are that they are stochastic, which mean that different runs produce different solutions; and that they are approximate, so no guarantee is given of the found solutions optimality. [37]

The basic building blocks of a genetic algorithm are the following:

1. Solution encoding. To apply a genetic algorithm, a way to encode the solutions must be defined. The traditional approach introduced by Holland[] is to encode solutions as a bit string. The common way to encode permutation problems, scheduling and routing problems among the others, is to represent a solution as a permutation of integers.

2. Generation of initial population. To start the evolution process, the first generation of solutions must be founded some way. A plain randomization may be used, but often some more sophisticated method is needed to provide good enough initial guesses and to speed up the process. When the problem is highly constrained, pure randomization may function poorly.
3. Selection. To apply the basic idea of GAs, namely recombining the good qualities of the previous generation to produce the next generation, a way to select which individuals to use as parents is needed. The measure of a solution's goodness, called fitness, is associated with each solution. The fitness should present the quality of the solution as accurately as possible and still be computable in a reasonable time. The selection should prefer the solutions with good fitness, but not too strongly because enough heterogeneity in the population is needed to avoid premature convergence. A standard way to implement selection is the so-called \textit{k-way tournament selection}, where \( k \) individuals are first randomly picked from the population, and then the fittest of these according to the fitness function is selected.

4. Crossover. Crossover is the process in which the qualities of the parenting solutions are recombined to generate offspring. The conventional idea is to use two parents, but other techniques may be applied. One-point crossover is the simplest crossover with two parents of bit string encoded solutions: a single cut point of the bit strings is selected, and the data on the other side of this cut point is swapped between the parents to produce the offspring chromosome.

5. Mutation. Mutation is used to bring randomization to the search process. Without mutation, the algorithm wouldn't be able to enter new regions of the search space. The classical mutation operator on bit string encoded chromosomes negates each gene on some small probability \( p \).

6. Replacing the population by next generation. The previous population is replaced either entirely or partly by a new generation of solutions generated using steps 3, 4, and 5. Different schemes have been applied in the literature: For example, the entire population can be replaced at a time; or some percent of the fittest individuals may be transferred to the next generation as such; or the replacement may be allowed to happen gradually by only replacing a few individuals at a time and so letting the generations get mixed.

7. Ending terms. Appropriate ending terms must be defined for the algorithm. Algorithm can end, for example, when no progress has taken place in a given number of iterations, when a predefined solution quality is achieved, or when the iteration has been running through a given time or generation limit.

Genetic algorithms were initially introduced as a general, multi-purpose optimization procedure. The general GA framework can be applied to any problem once the genetic encoding of the solutions and the fitness function for the specific problem are defined. However, to implement an efficient and robust GA, a design based on problem specific information and much fine-tuning are often required. Today, the most promising GAs to solve combinatorial optimization problems are usually hybridized with one or more local search methods and/or exact methods. All that creates better optimization methods in terms of efficiency and solution quality, but it also brings the idea further away
from the original simplicity of the GA framework. As the application areas grow more complex and diversified and the problem sizes increase, the importance of problem specific information and engineering also grows, creating need for more specialized solving procedures.

GAs are usually not competitive with traditional optimization methods where those are available. On the other hand, as long as the genetic encoding and fitness function can be defined, GA can be applied to the problem even if the problem domain is very complex or from other reasons poorly understood, if there exist many discontinuities or local extrema, or if the solution space of a \( \mathcal{NP} \)-hard problem is very large. GAs are approximate solution procedures and are therefore mostly applied to problems where a reasonably good or near-optimal solution suffices. This is often true with large combinatorial problems where showing that a given solution is optimal may be a \( \mathcal{NP} \)-hard task in itself. One of the known drawbacks of genetic algorithms is that they usually contain several parameters that have to be adjusted for the algorithm to be robust and to give a good performance. The same configuration of parameters may give excellent performance with some instances of the same problem while giving premature convergence in others. [2]

3 Routing and scheduling problems

Routing and scheduling problems are amongst the most important combinatorial optimization problems because of their vast amount of practical applications in systems where goods, people or messages need to be routed from one place to another, and because of the significant economical impact that optimizing of these systems may have. Another reason for the popularity of these problems with researchers is the challenge arising from their mathematical complexity. In this section some of the most common models used to handle these problems are introduced. All the models considered here can be interpreted as an extension or a special case of either of the two true classics of combinatorial optimization problems, namely the Travelling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP).

The problems described here are all defined in a connected digraph \( G = (V, E) \) consisting of a set \( V \) of \( n + 1 \) nodes and a set \( E \) of arcs. Node 0 represents the depot and nodes \( i, i \in V \setminus \{0\} \) are the customers. Each arc \( (i, j), i, j \in \{V\} \) is associated with travel cost \( c_{ij} \) and travel time \( t_{ij} \) including the service time of node \( i \).

**Travelling salesman problem (TSP).** TSP is probably the most well-known and widely studied of the combinatorial optimization problems. As the basic TSP is both very simple to formulate and hard to solve, it has long been a popular test problem for new algorithms in combinatorial optimization. In TSP, the \( n + 1 \) nodes represent cities that the salesman has to visit, and \( c_{ij} = t_{ij} \) for all \( i, j \in \{V\} \). The salesperson has to visit each city exactly once and return to the first city so that the total traveling time
is minimized, i.e. the objective is to find the shortest Hamiltonian cycle in the given graph $G$.

TSP is $\mathcal{NP}$-hard, i.e. no polynomial-time solving procedure is known to exist. Several important combinatorial optimization problems can be formulated as a variant of TSP by introducing e.g. time dependencies, time or tour length constraints, or allowing multiple tours. TSP has proved indisputably useful in practical problem solving, as many problems arising on the fields of scheduling and routing can be modelled as TSP or some of its variants or generalizations. Examples where TSP is used in modelling include machine sequencing and scheduling, cellular manufacturing scheduling, frequency assignment in communication networks, routing in telecommunication networks, data analysis and clustering, and artificial intelligence. [41]

**Vehicle routing problem (VRP).** Vehicle routing problem is the basic model for problems arising in many distribution systems. In VRP a fleet of identical vehicles, each with capacity $Q$, leaves from the depot, serves the customers with known demands and service times, and then returns to depot. The objective is to find optimal set of routes for the vehicles so that each customer is served exactly once, all demands are met, and the total demand of the customers assigned to any route does not exceed capacity $Q$. Split services are not allowed, i.e. each customer is served by exactly one vehicle. This basic version of VRP is sometimes called Capacitated VRP (CVRP). By setting the number of vehicles to one, the service times to zero, and the vehicle capacity to $\infty$ the problem reduces to TSP. The first time VRP was introduced as a mathematically formulated optimization problem by Dantzig and Ramser in 194x handling a real-world problem of delivering fuel to gas stations.

The most common ways more complex models are derived from these basic TSP/VRP setups are

1. Extending the number of tours from one to several (multiple traveling salesmen problems)
2. Extending the number of depots from one to several (multi-depot VRPs)
3. Introducing additional constraints on the nodes (time windows, skill levels, etc.)
4. Introducing additional capacity constraints (limited time for the tours)
5. Changing the objective function.

Here, a few of these models are introduced together with examples of their real-life applications.

Constraints of VRP and TSP type problems can be divided to intra-route and inter-route constraints, of which intra-route constraints concern one tour only, as opposed to inter-route constraints, in which multiple tours are considered simultaneously [21].
Constraints are usually defined with help of so-called resource variables. Examples of intra-route constraints include maximum time of each route, maximum capacity of each vehicle, and time window constraints. Inter-route constraints are often much more difficult to handle and also less research is focused on them, although they are present in many important applications. For example, the depot’s capacity constraints in logistics applications could be modeled as inter-route constraints, as well as constraints concerning the equal distribution of work load.

In the literature, vehicle routing problems are often optimized on a hierarchical objective with minimizing the number of tours as the primary objective and minimizing the total travel cost of the tours as the secondary objective. In many real-life cases other aspects than direct tour costs need to be taken into account. The travel cost itself can be defined in several ways, for example in terms of financial cost of travelling or the time spent on travelling. Real-life demands considering for example tour fairness (don’t allow too big variation between shortest and longest tour) or customer satisfaction (minimize customer wait time) are often most conveniently modelled as multi-objective problems rather than additional constraints. Sometimes constraints, like time windows constraints in VRPTW, are most conveniently handled by replacing them by objective function(s). [27], [28]

In many real-world situations, stochastic or time variant properties must be included into the model. One example of this sort of problems is VRP with online demands. However, these questions are out of the scope of this work.

**Multiple travelling salesman problem (MTSP).** In the multiple travelling salesman problem, the objective and constraints are the same as in TSP, but instead of one tour, \( M \) tours are looked for so that the total cost of all tours is minimized.

**Profitable tour problem (PTP).** In profitable tour problem, each node is associated with a prize, and not all nodes have to be visited. The aim is to find a tour that minimizes travel costs while maximizing prizes collected from the visited nodes.

**Orienteering problem (OP).** In orienteering problem, each node is associated with a prize, and not all nodes have to be visited. The objective is to find a tour that maximizes the prizes collected from the visited nodes so that the total cost of travelling doesn’t exceed a preset limit \( c_{\text{max}} \). The orienteering problem is usually stated so that the start and end points are separate and an optimal path between them is searched for instead of a cycle starting from and ending to a single depot. However, this difference is insignificant, for the problem can always be transformed to the single depot form by adding a dummy arc between start and end nodes. OP is sometimes also referred to as the maximum collection problem (MCP) or the selective TSP.
Prize collecting TSP (PCTSP). In the prize collecting TSP, each node is associated with a prize, and not all nodes have to be visited. The problem is to find a cycle so that the total prize collected from the visited nodes is at least $p_{min}$ and the total travel cost is minimized.

Generalized travelling salesman problem (GTSP). In the problem called generalized travelling salesman problem, the nodes are partitioned into groups, and the salesman has to visit at least one node from each group.

Minimum latency problem (MLP). Minimum latency problem is also known as the traveling repairman problem. In this problem, a tour must be found through all the nodes so that the average latency of the nodes is minimized. Given a tour, the latency of a node is the total time travelled before the node is reached. Examples of applications for MLP are disk-head scheduling and searching information in a network [9].

Vehicle routing problem with time windows (VRPTW). The vehicle routing problem with time windows extends the basic VRP by associating each customer $i$ with a time interval $[a_i, b_i]$, within which the service must be started. VRPTW is one of the most important variants of VRPTW with its wide applicability in real-world problems.

Vehicle routing problem with backhauls (VRPB). In the vehicle routing problem with backhauls, the customers are divided into linehaul customers, who require a given quantity to be delivered, and backhaul customers, where a given quantity must be picked up. This induces a precedence constraint for the customer nodes: if a route serves both linehaul and backhaul customers, all the linehaul customers must be visited before visiting any backhaul customers.

Vehicle routing problem with pickup and delivery (VRPPD). In the vehicle routing problem with pickup and delivery, the customer nodes are associated with two quantities: one to be delivered to the node and one to be picked up. For a customer node $i$, $O_i$ designate the origin node of the delivered demand, and $D_i$ is the destination of the pickup demand. The optimal tours must be found so that the load of the vehicles never exceeds the given capacity and is never negative, and all the customer demands, both deliveries and pickups, are satisfied. When the VRPPD is used to model pickup and delivery of persons instead of goods, it's usually called dial-a-ride problem.
4 Approaches to solving routing and scheduling problems

As TSP and VRP, together with many of their extensions, are both NP-hard and important in practical applications, extensive amount of solving procedures have been proposed in the literature. The decades of work on TSP has produced vast amount of research that has also been useful in developing algorithms for other scheduling and routing problems. The size of the problems that can be solved to optimality even with the best current exact algorithms on current computers, is often limited in comparison to problem sizes arising in real-world applications, which has spurred the development of approximative algorithms.

TSP instances with up to several thousands of nodes can be solved optimally with exact branch-and-cut methods [4], [24], while solving instances of more complex routing and scheduling problems with more than about fifty nodes to optimality still remains a challenge [21]. Cordeau et al. [19] state that no exact algorithm is capable of solving constantly basic VRP instances of more than 50 customers. Alvarenga et al. [2] summarize that many problems, as well as many of the classical Solomon's test instances with 50 or 100 customers, still remain unsolved. With the exception of TSP, most scheduling and routing problems with a real-life size can't be solved with exact algorithms effectively. According to [47], the largest instances of capacitated VRP that can be consistently solved by the most effective exact algorithms contain a few tenths of customers, whereas larger instances may be solved to optimality only in particular cases. According to [30], the largest VRP instances for which feasible solutions have been reported, include around 1200 customer nodes; real-life instances may have tens of thousands of nodes.

The performance of an algorithm with an optimization task often strongly depends on the utilization of problem-specific knowledge in design of the algorithm [23]. There are many ways to weight algorithms against each other; the point of view always depends on the application area. The quality of the solution in terms of objective function value, the speed of the algorithm, the simplicity of implementation, flexibility and robustness are listed as the most central criteria of a good algorithm in e.g. [13]. In many practical applications, the flexibility and robustness of an algorithm are of high importance, as the real-life data is often messy and additional constraints may be present [13]. Brasy et al. [13] also note that the quality of a heuristic algorithm is most commonly evaluated by experimental methods, thus requiring large set of test cases. The fastest is not always best, as well as it is not always necessary to invest much computing time to get a slightly better objective value than what could be obtained in a lot shorter time. So, a simple-to-implement general search algorithm may be fully sufficient to some purposes. However, as planning and scheduling problems often show high complexity, the performance of such general approaches is usually rather moderate. Many metaheuristic approaches were initially introduced as general problem solving strategies, but the algorithms showing best performance practically always incorporate at least some information specific to the problem domain [13].
Various surveys on solving vehicle routing and scheduling problems are available. For example, Laporte & al. (2000) [33] give an survey on both classical and modern heuristics to solve capacitated vehicle routing problem; Brávly and Gendreau give a comprehensive survey on classical heuristic methods in [13] and metaheuristic methods in [14] applied to VRP with time windows; Kallehauge (2008) [29] gives a thorough review on most recent achievements on exact algorithms for the vehicle routing problem with time windows.

The main approaches to solving these problems are only lightly touched here to add some depth to the more thorough discussion on genetic algorithm approaches in the next chapter. Following the common convention used by e.g. [48], the solving methods are divided here into exact methods; traditional heuristic and local search based methods; and metaheuristics.

### 4.1 Exact methods

With the exception of the basic TSP, where problem instances of thousands of nodes can be solved routinely with the best exact algorithms, more complex routing and scheduling problems can be extremely hard to solve to optimality even in small instances [?]. The exact algorithms for other types of routing and scheduling problems inherit a lot from the huge amount of work done on the TSP. Here only some of the main characteristics of the proposed exact algorithms are introduced; formulations and any details are omitted in the scope of this work. Note that most of the discussion here concerns the basic capacitated VRP or VRP with time windows; the same ideas, however, emerge in the algorithms designed for other types of routing and scheduling problems.

Laporte and Nobert (1987) [32] classified exact VRP algorithms as follows:

(i) direct tree search methods based on different relaxations (branch-and-bound),
(ii) dynamic programming, and
(iii) integer linear programming methods

Detailed descriptions of different VRP formulations and exact methods based on them can be found e.g. in and [48].

The early branch-and-bound algorithms of category (i) are based on basic combinatorial relaxations, like the assignment problem, the degree-constrained shortest spanning tree relaxation, and the state space relaxation, and were the most successful exact algorithms for TSP and TSP-derived problems until the late 1980’s [48].

In practice dynamic programming, i.e. breaking the problem into smaller subproblems and solving them recursively, can only be applied to routing and scheduling problems in small problem instances. Applied together with a proper relaxation of the problem, however, it can prove useful in solving lower bounds that can be used in branch-and-bound algorithm. [48]
More sophisticated bounds based on the integer linear program formulations of the problem have been proposed more recently. Branch-and-cut, that combines branch-and-bound with cutting plane methods, and branch-and-price that does the same with column generation approaches, form the state-of-the-art for exact algorithms for VRP type problems as well as many other problems that can be formulated as integer linear programs. Review on these approaches can be found in [29]. An advanced branch-and-cut-and-price algorithm has solved to optimality thus far the largest TSP instances with tens of thousands of nodes; the results were reported in [3]. Laporte (2007) [35] notes that to date the integer linear programming based branch-and-cut algorithms are the only workable exact methods for the capacitated VRP, and even their success in solving problem instances of realistic size is limited.

4.2 Traditional heuristic methods and local search

The approximate solving procedures for routing and scheduling problems developed mainly between 1960 and 1990 are sometimes called traditional or classical heuristics. These methods normally perform quite limited search in the solution space, and produce quite good solutions in modest execution time. Many of them are easy to implement and also easy to modify to take into account different constraints arising in real-life applications. These qualities account for their still being common in commercial applications even though more modern heuristics unanimously beat them in solution quality and sometimes also in execution time. [34]

Traditional heuristics can be categorized into three groups: constructive heuristics, that gradually build a feasible solution while keeping account for the solution cost; two-phase heuristics that cluster the nodes into groups and construct the routes inside each group separately either in cluster-first, route-second manner or vice versa; and improvement heuristics, that aim to improve any feasible solution by a sequence of arc or node exchange moves. Improvement heuristics are also known as local search. [13]

Even though the importance of classical heuristics as the primary solving procedure has been diminishing, they still play an important part in more complex modern meta-heuristic search procedures. Route construction or two-phase heuristics are often used to provide for initial guesses for the metaheuristic, and some sort of local search is used as a quick improvement method once the more general search has entered an interesting part of the solution space in all successful metaheuristics for complex routing and scheduling problems.[14]

All local search methods are based on the use of elementary moves that transform a given solution into a different neighbor solution. The set of all solutions that can be reached from the current solution using the set of moves are called the neighborhood of the current solution. Feasible solutions that don’t have any improving solutions in their neighborhood, i.e. solutions with a better objective function value, are called local optima. A local search algorithm looks for at least one improving solution in the neighborhood, moves to that solution making it the new current solution, and proceeds
repeating this until a local optimum is found or a predefined stop condition is met. The efficiency of an local search method depends greatly on the definition of the neighborhood and the search algorithm used to explore it. Funke (2005) states that the close relationship between the neighborhood and the search algorithm is exploited in many of the most successful VRP algorithms, but still relatively little attention is given to this area [21].

According to [21], edge-exchange neighborhoods are the most commonly used choice in vehicle routing and scheduling problems; well-known Or-Opt, k-Opt and Lin-Kernighan neighborhoods belong to this group. A proper design of the neighborhood and an efficient local search method can decrease the running time of the optimization method substantially and enable handling bigger problem instances [21]. According to Funke (2005), local search and local search-based metaheuristics are currently the only available methods for obtaining good solutions to large vehicle routing and scheduling problems [21].

4.3 Metaheuristics

The metaheuristics, sometimes referred to as modern heuristics, have been under extensive study since the 1990s for solving routing and scheduling problems and other hard combinatorial optimization problems. Metaheuristics are general problem solving procedures that explore the solution space according to some specific strategy, using diversification scheme to enter new search areas, and intensification schemes to exploit the information already accumulated during the search to avoid wasting time in regions that either are already explored or don’t provide good solutions [10]. Metaheuristic search methods mainly utilizes the principles of local search, population search, and learning mechanisms [35]. Metaheuristics often embed one or more of the traditional route construction heuristics, and all the succesful ones on routing and scheduling problems utilize some form of local search [14]. Metaheuristics can also be combined with exact algorithms with the aim at improving either solution quality or the computation time; a survey on these topics can be found in [40].

The most well known metaheuristics based on local search are simulated annealing (SA) and tabu search (TS), that both move at each step from the current solution to another promising solution in their neighborhood and use a specific scheme to avoid getting stuck into bad local optima. Genetic algorithms are a good example of population based metaheuristics, where the idea is to maintain a pool of good solutions and use it by to produce better solution candidates at each iteration; these are discussed in more detail in the next chapter. Ant systems and neural networks are examples of learning systems used in combinatorial optimization. An overview on metaheuristics in combinatorial optimization can be found in [10]; [22] gives an survey on applying these methods on capacitated vehicle routing problem, and [14] for the vehicle routing problem with time windows.

Metaheuristic methods often produce solutions of much better quality than what can
be obtained by classical heuristics or pure local search algorithms. On the other hand, designing, implementing, and calibrating them is often complex, and they usually require much more CPU time and memory to execute than classical heuristic and local search algorithms. [14]

According to Cordeau et al. (2002) [19], tabu search clearly stood out as the best meta-heuristic for the VRP, when accuracy and speed of the algorithm as well as simplicity and flexibility of the implementation are taken into account. However, soon afterwards it was reported that the results obtained with the fewer attempts with genetic algorithms to VRP with time windows are highly promising [23] and the most recent work shows that hybridized GAs are becoming more and more competitive as can be concluded from the short survey on work done with GAs given in the next chapter.

5 Genetic algorithms in solving routing and scheduling problems

Initially, GAs were proposed for solving unconstrained optimization problems that can be represented as bit strings. Routing and scheduling problems, on the other hand, are often highly constrained. Besides that, they are permutation problems by nature, which has two important consequences: it makes bit string representation of chromosomes impractical; and it causes the traditional crossover and mutation operators to produce infeasible solutions with high probability. Thus, specialized operators as well as chromosome structures have been designed for this type of problems. In this chapter genetic algorithms are viewed in the context of these special demands, and a brief survey to the work where GA approach is used to solve routing and scheduling problems is given.

Bräysy (2001) [12] stated that for VRP with time windows, no pure genetic algorithm is competitive with best published results obtained by other heuristic methods. However, some of the best VRPTW algorithms to date are hybridizations of GA with different construction heuristics, local search and other metaheuristics [14].

GA’s were first introduced as general problem solving framework that can be applied to any problem once the genetic encoding and fitness function are be defined. In practice, however, when working with complex problems like VRP, at least some degree of problem specific customization is needed to make GA efficient enough. The best GA’s introduced in literature for solving routing and scheduling problems, all utilize one or more local search methods to increase the solution quality and speed up the optimization. Hybridizations with other optimization methods like branch and cut algorithms or other metaheuristics are also often seen.
5.1 Chromosome design

When applying GAs to routing and scheduling problems, the solutions are usually encoded as chromosomes using permutation of integers. Different set of integer indices is often used as route delimiters. A good chromosome structure is essential for an efficient GA. Good chromosome design should reduce or eliminate redundant chromosomes in the population. By redundancy, we mean that the same solution can be represented in more than one ways and thus appear multiple times in the population. Multiple representations of the same solution increase the search space and consequently slow down the search. A good chromosome structure reduces or eliminates redundancy, accurately represents a solution to the problem, allows the GA operators to work effectively, and enables easy evaluation of the fitness. [15]

5.2 Genetic operators

As routing and scheduling problems are permutation problems by nature, the traditional genetic operators usually produce infeasible offspring or otherwise fail to capture the desired features of the solutions. The design of operators also depends on the design of the chromosome representation. The basic crossover and mutation operators for permutation chromosomes are introduced here. All of them were first proposed for TSP and, but are applicable for other problems as well although the ideas of preserving certain properties of the parent solutions will suffer if the operators are applied as-is into chromosomes with multiple tours or other special characteristics not present in basic TSP. Larrañaga and al. (1999) [36] give a thorough review on GA operators for solving travelling salesman problem. The same ideas are widely applied with other routing and scheduling problems as well.

In partially-mapped crossover (PMX), two locations in the chromosome are selected, and the portion between these locations in one parent is mapped onto a portion between the same locations on the second parent, and the remaining information is exchanged. The idea of the operator is to pass the ordering and value information from the parents to the offspring. For example:

Parent 1: (1 2|3 4 5|6)
Parent 2: (2 4|6 5 1|3)

Offspring 1: (x x|3 4 5|x)
Offspring 2: (x x|6 5 1|x)

Using mapping (3 ↔ 6, 4 ↔ 5, 5 ↔ 1) derived from the mapping section, the offspring becomes

Offspring 1: (2 1|3 4 5|6)
Offspring 2: (4 2|6 5 1|3)
In cycle crossover (CX), the idea is to preserve the locations of the genes in the parent chromosomes in the offspring as well as possible. This is done so that each city and its position comes from one parent. In the example below, we start building offspring 1 by selecting the gene at first location in parent 1 and insert that into first location in the offspring. Then we see that value 2 occupies the same location in the second parent; we find the same value in parent 1 and copy it together with its location to the offspring. The same location in parent 2 is occupied by value 4, which gets copied next. On its location in parent 2 we find value 5, and so we copy value 5 from parent 1 to the offspring. Now we find that value 1 that occupies the corresponding location in parent 2 is already used, so we have found a cycle. The rest of the genes along with their locations are thus copied from parent 2 to the offspring. Applying the same method again but starting form parent 2, we get the other offspring chromosome.

Parent 1: (1 2 3 4 5 6)
Parent 2: (2 4 6 5 1 3)

Offspring 1: (1 2 6 4 5 3)
Offspring 2: (2 4 6 5 1 3)

The idea in order crossover (OX1) is to preserve the relative ordering of the genes, not their locations. Two locations in the parent chromosomes are selected, and the portion between them is transferred to the offspring from one parent to the corresponding location. Then, the rest of the genes are copied to the offspring, starting from the second cut point, in the order in which they appear in the second parent, omitting the genes that already exist in the offspring:

Parent 1: (1 2|3 4 5|6)
Parent 2: (2 4|6 5 1|3)

Offspring 1: (x x|3 4 5|x)
Offspring 2: (x x|6 5 1|x)

Offspring 1: (6 1|3 4 5|2)
Offspring 2: (3 4|6 5 1|2)

The simplest mutation operators for permutation chromosomes are exchange mutation, which swaps the locations of two randomly selected genes, and insertion mutation, that removes one randomly selected gene and inserts it into new randomly selected location. Displacement mutation selects a subtour randomly, removes it from the original place and inserts into a new random location. In the traditional GA approach, mutation is used to escape local optima and enable entering new regions of search space, but with complex scheduling and routing problems, local search methods are sometimes used as mutation operators and may even be the main driving mechanism of the search.

[14]
5.3 Constraint handling

Sometimes it is possible to design the chromosome structure and genetic operators so that infeasible solutions never occur, but this is rarely the case when dealing with complex, highly constrained real-life problems where finding any feasible solution can be an exhaustive task. Coello Coello (2001) [17] gives a survey to various constraint handling methods proposed in literature, including penalty function methods, repair algorithms, and co-evolution and other methods exploiting the separation of objectives and constraints.

The simplest way to handle constraints in a GA is to embody them in the fitness function as penalty terms related to the constraint violations. This approach has the drawback of producing additional parameters that have to be experimentally evaluated [17]. The assumption that infeasible solution candidates are always worse than feasible ones may also seriously limit the exploration of highly constrained solution space thus making the algorithm perform poorly [17]. Chu and Beasley (1998) [16] point that the use of penalty functions was the main reason why some early attempts to apply GAs for highly constraint problems had difficulties in producing feasible solutions.

Coello Coello (2001) [17] suggests the use of repair methods with combinatorial optimization problems. In repair methods, unfeasible solutions are allowed but repaired to feasible solutions before the fitness is evaluated. Then, the original infeasible solution is replaced by the repaired version with some given probability. This approach is reported to be a good choice, when an infeasible solution can be easily transformed to a feasible one. This is not always the case, and also the repair operators may introduce a strong bias in the search harming the evolutionary process itself.

For highly constrained problems Coello Coello (2005) [17] suggests the use of methods that handle objective function and constraints separately. This can be implemented in many various ways. One method is to define fitness function so that it is different for feasible and infeasible solutions, fitness of any feasible solution always dominates fitness of any infeasible solution, and the fitness of an infeasible solution reflects the amount of constraint violations in a proper way. Another approach is defining the problem as an multi-objective optimization probelem where some of the constraints are introduced as objectives instead, and using for example Pareto ranking or other multi-objective optimization methods when the fitness of the solutions is compared. Other methods as co-evolutionary and behavioral memory approach are described in [17].

5.4 Initial population

The initial population may have a considerable effect on the performance of a genetic algorithm. Especially in case of highly constrained problems, the quality of the initial population may become crucial to the success of the algorithm. Traditional constructive heuristics are fast, popular, and often readily available solution to this. [14]
5.5 Experiences from solving routing and scheduling problems with genetic algorithms

In their article published in 1995, Rochat and Taillard [42] presented encouraging results for VRP and VRPTW with hybridized algorithm combining tabu search and GA and improved by a post-optimization technique based on solving the set partitioning problem. Later, Alvarenga et al. [2] developed their idea further and presented a hybrid algorithm that employs both genetic and set partitioning approaches in solving the vehicle routing problem with time windows, outperforming all previously published heuristic methods in terms of the minimal travel distance. In their approach, GA was used to generate diversity of local optimal solutions containing high quality tours. After running the GA, set partitioning algorithm was employed to construct final solution from the generated tours.

In the work conducted by Prins (2004) [39] an effective evolutionary algorithm for VRP was presented. The paper presented the first hybrid GA for the VRP that is able to compete with TS algorithms in terms of average solution cost. The algorithm was able to find new best-known solutions to several benchmark instances. It was reported, however, that the algorithm was considerably slower than several TS based algorithms. One of the new improving features of Prins’ GA was the use of a local search procedure as the mutation operator instead of the traditional move or swap operators.

Alba and Dorronsoro (2006) [1] reported a good performance of a cellular genetic algorithm in solving the capacitated vehicle routing problem, being able to improve best-known solutions in several benchmark problem instances with very reasonable computing effort. In their algorithm, the population was organized as a 2D toroidal grid and crossover was only allowed between neighbors.

Berger and Berkouwer (2003) [8] developed a genetic algorithm hybridized with local search algorithms to solve the capacitated VRP. Their algorithm evolved two populations simultaneously with migration of best solutions in each generation. In the computational tests, their approach proved to be cost-effective and very competitive in comparison with the best VRP metaheuristics available.

Baker and Ayechew (2003) [5] studied the capacitated VRP and applied a pure genetic algorithm and a hybridized algorithm with embedded local search. They obtained tolerable results also with the pure GA, and solutions very close to the best known solutions with the hybridized algorithm.

Ombuki et al. (2006) [38] studied VRPTW as an multi-objective optimization problem to avoid bias to either number of vehicles or travel distance, and developed a genetic algorithm to solve it. Their results were reported to be comparable with the best published results, and also introduced new best solutions with respect to Pareto optimality between the two objectives.

A GA with new chromosome structure and hybridization with local search to solve VRPTW developed by Tan et al. (2001) [45] produced competitive results with respect
to both best and average results obtained.

Carter and Ragsdale (2005) [15] developed a genetic algorithm to solve the multiple TSP, where instead of one tour a given number \( m > 1 \) tours are searched for. E.g. production line scheduling and vehicle scheduling are often modeled as MTSP. The proposed a GA had a new chromosome structure, and the computational tests showed performance superior to chromosome structure and related operators traditionally used to represent TSP related problems. However, no comparison with other metaheuristic or exact algorithms was presented.

Tasgetiren (2001) [46] applied genetic algorithm to orienteering problem. Comparison with the best results obtained by problem specific heuristics and an artificial neural network optimizer showed the GA to be competitive with them.

## 6 Experimental part

The second part of this work was to define a test problem that falls into the category of the routing and scheduling problems studied in this work, to implement a genetic algorithm for solving it, and to test the algorithm and report the results. The idea was not to develop a competitive algorithm, but to give the writer some hands-on experience of handling this type of problems with a genetic algorithm in practice. The outcome of the experiment is described in this chapter.

### 6.1 The test problem

The test problem is to schedule \( N \) technicians to complete \( M \) geographically distributed tasks. 'Vehicles' are called 'technicians' here just to emphasize that this is not a VRP but could model a situation where e.g. maintenance technicians need to be scheduled to travel between distributed work sites. The planning horizon is \( H \) days. A completion time \( t_i \) is defined for each task \( i \in \{1, ..., M\} \), as well as a time window \( [t_{i,h}^{\min}, t_{i,h}^{\max}] \), where \( t_{i,h}^{\min} \) indicates the earliest and \( t_{i,h}^{\max} \) the latest possible time to start handling task \( i \) on day \( h \), \( h \in \{1, ..., H\} \). If a technician arrives to task \( i \) on day \( h \) earlier than \( t_{i,h}^{\min} \), he has to wait. If technician arrives to task \( i \) later than \( t_{i,h}^{\max} \) on day \( h \), the solution is infeasible. Additionally, each technician \( c \in \{1, ..., N\} \) has an individual skill profile which has to match the skill level demand defined for each task. Each technician's each day’s tour begins and ends at the depot; depot’s time window \( [t_{0,h}^{\min}, t_{0,h}^{\max}] \) defines the earliest possible start time and latest possible end time for each tour each day. It is assumed that each task is atomic and exact completion times can be defined. The time needed for travelling and the travel distance between any two tasks or a task and the depot is given as the Euclidean distance of the task locations. It is not required to complete all tasks, and we do not assume anything about whether all the tasks can be completed or not by the given set of technicians within the given planning horizon. The
primary objective is to complete as many tasks as possible, and the secondary objective is to minimize the total travel distance.

The problem formulation is summarized as follows:

- The number of technicians is $N$.
- The number of tasks is $M$.
- Planning horizon is $H$.
- Each task $i \in \{1, \ldots, M\}$ has location $(x_i, y_i)$. Node 0 is called depot and its location is given by $x_0, y_0$.
- Distance and travel time between any two nodes $i$ and $j$, $i, j \in \{1, \ldots, M\} \cup 0$ is given as their Euclidean distance.
- Completion time for task $i \in \{1, \ldots, M\}$ is $t_i$ and $t_0 = 0$.
- For each task $i \in \{1, \ldots, M\}$, time window $[t_{i,h}^{\min}, t_{i,h}^{\max}]$ is defined for each day $h \in 1, \ldots, H$ within which the task can be started.
- Depot’s time windows $[t_{0,h}^{\min}, t_{0,h}^{\max}]$, $h \in 1, \ldots, H$ define the earliest possible start time and the latest possible end time for each tour.
- There are $S$ skills defined in the system, and each technician possesses some subset of these skills. Each task demands some subset of these skills, and the task can be allocated only to a technician that has all the demanded skills.
- The problem is to find $MN$ tours that start and end in the depot so that the number of completed tasks is maximized, the total travel time minimized, and the time window, day length and skill level constraints are not violated.

The test problem can be reduced to a VRP with time windows by defining for each technician equal skill profile and each task equal skill demand. In the corresponding VRP, the capacity of the vehicles is $\infty$ and the demand of the nodes is insignificant. As a generalization of VRP, the test problem is also $\mathcal{NP}$-hard, which makes the choice of trying to solve it with a GA reasonable.

By allowing part of the tasks to remain unallocated we avoid directly addressing the difficult question whether there exists a feasible solution where all the tasks can be completed or not. Indeed, finding a feasible solution to VRP with time windows with a fixed fleet size is a $\mathcal{NP}$-hard problem itself [18], so the same would be true here if completing all the tasks was required here. Now, in our case, any infeasible solution can easily be repaired to a feasible one by simply removing from the tour all the tasks that violate the time window constraints, and leaving them unallocated.
6.1.1 Integer Linear Program formulation

To give more formal definition of the test problem, integer linear program (ILP) formulation is given in this section. Works of Kallehauge (2008) [29], Dohn et al. (2009) [20], and Toth & Vigo [48] were used as reference in formulating the ILP. The ILP formulation is not directly needed in the implementation of the GA, but it is useful for understanding the problem and could be used if an exact algorithm was developed or an ILP solver applied to the problem.

**Definition 1.** A time constrained digraph $D = (V, A, t, E, F, R, c)$ is defined by the set of nodes $V$, set of arcs $A$, arc durations $t$, node release times $E$, NODE due times $F$, rewards from the nodes $R$, and arc costs $c$ as follows: $V = V^* \cup \{0, n + 1\}$ is a set of nodes where $V^* = \{1, \ldots, n\}$ is the set of $n$ customer nodes and 0 and $n + 1$, respectively, the start and destination depot nodes. Arc set $A = A^* \cup \{\delta^+(0)\} \cup \{\delta^-(n + 1)\}$, where $A^* = A(V^*)$ is the set of arcs spanned by the customer nodes and $\delta^+(0) = \{(0, i) | i \in V^*\}$ is the set of arcs leaving the start depot node and $\delta^-(n + 1) = \{(i, n + 1) | i \in V^*\}$ is the set of arcs entering the destination depot node. Service times on nodes are $p \in \mathbb{Z}_+^n$ where $p_0 = p_{n+1} = 0$ and $p_i > 0$ for $i \in V^*$, and travel times on arcs are $t' \in \mathbb{Z}_+^n$ where $t'_{ij} \geq 0$. Arc durations are $t \in \mathbb{Z}_+^n$, where $t_{ij} = p_i + t'_{ij}$. Release and due times on nodes are $E, F \in \{\mathbb{Z}_+ \cup \{\infty\}\}^{V H}$ where $c_d^d = c_{n+1}^d = L$ for $d \in H$ and $f_0^d = f_{n+1}^d = M$ for $d \in H$ where $H = \{1, \ldots, h\}$ is the set of days for which the release and due dates are defined, and $L$ and $M$ are the minimum starting time and maximum ending time of each day’s tours, respectively. Rewards from serving node $i$ on day $d$, $d \in H$, are $R \in \mathbb{Z}_+^{V H}$, where $r_0^d = r_{n+1}^d = 0$ and $r_i^d > 0$ for $i \in V^*, d \in H$. Costs from travelling arcs are $c \in \mathbb{Z}_+^n$ where $c_{ij} \leq c_{ik} + c_{kj}$ for $i, j, k \in V$.

**Definition 2.** The arrival times of nodes $V(P_d)$ on path $P_d = (v_1^d, v_2^d, \ldots, v_k^d)$, $d \in H$, is vector $s \in \mathbb{Z}_+^{V(P)}$ where $s_{v_1}^d = e_{v_1}^d$ and $s_{v_i}^d = \max\{s_{v_{i-1}}^d + t_{v_{i-1} v_i}, e_{v_i}^d\}$ for $i = 2, \ldots, k$. The reward of the path on is $r(A(P_d))$ and the cost of the path is $c(A(P_d))$ where $A(P_d)$ is the set of the arcs of the path.

**Definition 3.** Path $P = (v_1, v_2, \ldots, v_k)$ is feasible on day $d \in H$ if

\[ s_{v_i}^d \leq f_{v_i}^d \quad \text{for all } i \in V(P) \]  

and

\[ s_{v_k}^d \leq L . \]  

**Definition 4.** Path $P = (v_1, v_2, \ldots, v_k)$ is infeasible on day $d \in H$ if

\[ s_{v_i}^d > f_{v_i}^d \quad \text{for any } i \in V(P) \]  

or

\[ s_{v_k}^d > L . \]  

**Definition 5.** Route $R_d$ in $D$, $d \in H$, is an feasible path from start depot to end depot, i.e. $R_d = (0, v_2^d, \ldots, v_{k-1}^d, n + 1)$. $\mathcal{R}$ is the set of all routes in $D$.
With decision variables

\[ x_{ijkd} = \begin{cases} 1 & \text{if technician } k \text{ goes from task } i \text{ immediately to task } j \text{ on day } d \\ 0 & \text{otherwise} \end{cases} \]

and

\[ s_i = \text{arrival time at task } i; \ i \in \mathbb{Z}^+ \cup \{0\} \]

and using the definitions given above the optimization problem can be formulated as

\[
\begin{align*}
\text{max} & \quad \sum_{d \in D} \sum_{k \in K} \sum_{j \in J} \sum_{i \in V} (x_{ijkd})r_i - \sum_{d \in D} \sum_{k \in K} \sum_{j \in J} \sum_{i \in V} (x_{ijkd})c_i \\
\text{s.t. } & \quad \sum_{d \in D} \sum_{k \in K} \sum_{j \in J} j \in Vx_{ijkd} \leq 1 \quad \forall i \in V^*, \\
& \quad \sum_{j \in J} x_{0jkd} = 1 \quad \forall k \in K \forall d \in H, \\
& \quad \sum_{j \in J} x_{j,n+1,kd} = 1 \quad \forall k \in K \forall d \in H, \\
& \quad \sum_{i \in V} x_{imkd} - \sum_{i \in V} x_{mjd} = 0 \quad \forall m \in V \forall k \in K \forall d \in H, \\
& \quad T_{min} + t_0 - M(1 - x_{0jkd}) \leq s_j \quad \forall j \in V^*, \forall k \in K \forall d \in H, \\
& \quad s_i + t_0 - M(1 - x_{0kd}) \leq T_{max} \quad \forall i \in V^*, \forall k \in K \forall d \in H, \\
& \quad s_i + t_{ij} - M(1 - x_{ijkd}) \leq s_j \quad \forall i \in V^*, \forall j \in V^*, \forall k \in K \forall d \in H, \\
& \quad \sum_{j \in J} x_{ijkd} a_i^d \leq s_i \quad \forall i \in V^*, \forall k \in K \forall d \in H, \\
& \quad \sum_{j \in J} x_{ijkd} s_i \leq b_i^d \quad \forall i \in V^*, \forall k \in K \forall d \in H, \\
\end{align*}
\]

Object functions (5) maximize the total profit from all the tours and minimizes the total cost of travelling. The constraints (7) guarantee that each task is assigned at most once during the planning horizon. Constraints (8) and (9) force each tour to start at the start depot and end at the destination depot, respectively. Constraint (10) ensure that each entered node is also left by the same technician during the same day. Each tour must start and end inside the working hours, which is enforced by constraints (11) and (12). A task can only be started within its time windows. This is enforced by (14) and (15), where \( M \) is a large positive scalar. Constraints (16) and (17) are the integrality
constraints of the decision variables. Routing problem formulations often include sub-
tour elimination constraints; here, however, they are unnecessary due to the definition
of arrival times $s_i$ and the fact that $t_{ij} > 0 \forall i, j \in V$, and at least one of $i, j \in V^*$. In
this formulation, the day length on each day of the planning horizon was fixed to the
same for simplicity.

6.2 The algorithm

In this section, the essential components of the GA design and implementation are
described. The algorithm was implemented in Java and tested in UNIX environment.

6.2.1 GA Framework

The programming task started with designing and implementing a general GA fram-
ework that could be used to solve the test problem. The idea was to follow object-oriented
software design principles and implement a reusable framework that could be extended
to solve any type of problems. This was achieved by hiding all problem-specific informa-
tion behind the interface of the fitness function. The class diagram of the implemented
Java application is shown in picture 6.1. The implementation allows free creation of
different types of genes, chromosomes, and fitness functions that can be used in the gen-
eral evolution framework. The implementation now only includes the featured needed
to solve the test problem.

There are several open or free source Java packages available in the internet that could
have been used instead of creating the algorithm framework from scratch, see e.g. JGAP
(Java Genetic Algorithms Package) [26]. However, the writer wanted to gain experience
on designing the system from start and so ready-made solutions were not used.

6.2.2 Chromosome

The chromosome used in the test algorithm is formed by first listing the tasks that are
not allocated to any of the technicians, i.e. those tasks that will not be completed within
the given planning period. Then, each technician’s each day is indicated by a negative
integer from $-1$ (first day of technician 1) to $-M \cdot H$ ($H$ th day of technician $M$),
followed by a list of tasks allocated to the indicated technician to be completed in the
given order on the indicated day. An example of a chromosome with two technicians,
planning horizon of two days and ten tasks is illustrated in 6.2.

Alternative chromosome structures were also considered. There are several codings
proposed for multiple traveling salesperson problems in the literature, see [15], that
could also be applicable here. Carter [15] proposes a two-part chromosome for MTSP
and shows its advantages over the traditional one chromosome and two chromosome
approaches. However, the constraints emerging due to the test problem’s skill matching
Figure 6.1: Class diagram of the Java application implemented to solve the test problem.
and time windows constraints make the chosen simple coding efficient enough in respect to redundancy in the general case: the tours are not interchangeable as long as the technicians and time windows for different days are not identical.

6.2.3 Fitness function

A hierarchical two-level fitness function was used for the test problem. The first objective was to allocate as many tasks as possible to the tours. The second objective was to minimize the total travel cost of the tours. Given two chromosomes \( c_1 \) and \( c_2 \) that belong to the group of all possible chromosomes \( C \) and the fitness function \( F : C \to \mathbb{N} \times \mathbb{R} \), \( F_1(c) : C \to \mathbb{N} \) is the number of completed tasks in solution \( c \), \( F_2(c) : C \to \mathbb{R} \) is the total travel distance in solution \( c \), the comparison between the fitness of the two chromosomes is defined as

\[
F(c_1) > F(c_2) \iff F_1(c_1) > F_1(c_2) \lor (F_1(c_1) = F_1(c_2) \land F_2(c_1) > F_2(c_2))
\]

\[
F(c_1) = F(c_2) \iff F_1(c_1) = F_1(c_2) \land F_2(c_1) = F_2(c_2)
\]

\[
F(c_1) < F(c_2) \iff F_1(c_1) < F_1(c_2) \lor (F_1(c_1) = F_1(c_2) \land F_2(c_1) < F_2(c_2))
\]

6.2.4 Selection

\( k \)-way tournament selection was employed to select the parents of each generation. This means, that to select a parent chromosome, first \( k \) random individuals were picked from the population, and then the parent would be the one with the highest fitness among these \( k \) individuals. In the chosen implementation, every pair of parents would produce one offspring, and then new parents would be selected the same way.

6.2.5 Genetic operators

Three standard crossover strategies for permutation chromosomes were implemented: partially mapped crossover (PMX), cycle crossover (CX), and ordered crossover (OX1). The proportions in which these methods are applied is parametrized, and each time
crossover is performed, the method to use is drawn from these options randomly according to the given probabilities.

After crossover, mutation was applied to the offspring chromosome with given probability $p$. The mutation applied here simply selects two random genes from the chromosome and swaps their locations.

It should be noted, that the crossover and mutation operators don’t make any difference between tasks and route delimiters in the chromosome.

### 6.2.6 Constraint handling

Because it is highly possible that applying crossover and mutation operators produces infeasible solutions with respect to the time window and skill level constraints, a repair procedure was implemented: whenever a solution violates the constraints, the violating tasks are removed and unallocated until the solution is feasible. The violating tasks are looked for in random order to avoid any biases in the fixing procedure. The constraints were checked every time fitness was evaluated, and after the evaluation the infeasible solutions were repaired and fitness of the feasible solution calculated.

### 6.2.7 Local improvement

Six local improvement methods were implemented:

- Take a random unallocated task and insert it into first found feasible location in a tour.
- Take a random allocated task and try to find a better location within the same tour.
- Take a random allocated task and try to find a better location in any tour.
- Take a random allocated task and replace it with a random unallocated task if this improves the solution.
- Take a two random tasks in the same locations in two tours and swap them if this improves the solution.
- Take a random allocated task and swap it with another task so that the improvement of the solution is maximized.

Two neighborhood structures are involved in these methods: The first three methods are based on relocation of a task, and the last three methods on the swap of two tasks. Only the last method does exhaustive search of the neighborhood while the others stop with first improving move that is found when searching the neighborhood in random order.
It is obvious that these heuristics are not very efficient due to the limitations of the neighborhoods. The methods are not able to efficiently relocate complete segments of more than one tasks, for example, so the GA crossover operators alone are left responsible of this fundamental task.

6.2.8 Initial population

As the highly constrained solution space makes most of the permutations of the tasks and tour delimiters infeasible, the initial population was generated so that it consists of feasible solutions only. Here, the initial solutions were generated by a greedy heuristic so that first a technician and a work day were selected randomly, and then randomly selected tasks were added to the schedule as long as this could be done without breaking the day length or skill level limitations. With this simple heuristic the average quality the solutions in the initial population still remains rather weak, but a little testing showed that it is still remarkably better than pure randomization with no guarantee of feasibility.

6.2.9 Replacing the population by next generation

A given percent of the fittest individuals from the last generation is directly transferred to the next generation. The rest of the new individuals are produced by the means of selection, crossover and mutation explained above.

6.2.10 Summary of the algorithm

The algorithm consists of the following steps:

1. Initialize the algorithm by giving values to the following parameters:
   - Size of the population \( n \)
   - Stop condition for the algorithm: algorithm is terminated either after a fixed number \( s_{\text{fixed}} \) of steps is reached, or when there’s been no progress in the last \( s_{\text{noProgress}} \) generations
   - Probabilities for using different crossover methods
   - Mutation probability \( p_{\text{mutation}} \)
   - Tournament selection parameter \( k \)
   - Proportion \( p_{\text{survival}} \) of fittest individuals to transfer directly to the next generation

2. Generate initial population by a greedy heuristic.
3. Find $p_{\text{survival}} \ast n$ fittest individuals from the population and transfer them directly to the next generation.

4. Select two parents by $k$-way tournament selection to produce two offspring individuals. The crossover method for each offspring chromosome is randomly selected from the options according to the given probabilities. Apply local improvement operator and random mutation operator on probability $p$ on each offspring and add the offspring to the new generation. Repeat this until the new population has $n$ individuals.

5. Repeat steps 3 and 4 until stop condition is met.

6.3 Test cases

The test data for the algorithm was derived from the well-known Solomon benchmark problems for the capacitated vehicle routing problem with time windows [44]. The original data consists of six sets of problems, but in the limited scope of this work, only Solomon problem instance C101 with 25 and 50 customer nodes was chosen for testing. In the problem instance C101, the geographical data is clustered and randomly distributed within a $[0,100]^2$ square. Scheduling horizon is short allowing only a few customers per route, approximately from 5 to 10. The 25 first customer nodes in the 25 and 50 node sets are same. The travel distances and times are assumed equal and are defined as the Euclidean distances between the nodes; in addition each customer node has a service time. Identical vehicles with a given capacity for each problem are assumed. It is assumed that the objective is to minimize the number of the vehicles and the total travel distance.

The modifications to the original data needed to form a test problem for this study listed below:

- Omit all vehicle capacities and customer demands.
- Define a fixed number of vehicles. A 'vehicle' is called 'technician' in the case of this work's test problem.
- Define a skill profile for each technician and skill requirements for each task.
- Define a planning horizon.
- Define time windows for the tasks for each day in the planning horizon. Here this was done by simply copying the time windows given in the original problem to each day of the planning horizon.

Eight test cases were defined with different planning horizons and technician set-ups. The definition of each test case is given in table 6.1. The skill demand for the customer
\begin{table*}  
\centering  
\begin{tabular}{|c|c|c|c|}
\hline
Test case & Solomon problem & Planning horizon & Technicians \\
\hline
Test 1 & C101, 25 nodes & 1 & 1, all skills \\
\hline
Test 2 & C101, 25 nodes & 1 & 6, all skills \\
\hline
Test 3 & C101, 25 nodes & 3 & 3, mixed skills \\
\hline
Test 4 & C101, 50 nodes & 1 & 3, all skills \\
\hline
Test 5 & C101, 50 nodes & 1 & 3, mixed skills \\
\hline
Test 6 & C101, 50 nodes & 3 & 6, all skills \\
\hline
Test 7 & C101, 50 nodes & 5 & 3, mixed skills \\
\hline
Test 8 & C101, 50 nodes & 1 & 6, all skills \\
\hline
\end{tabular}  
\caption{Test cases used in this study.}  
\end{table*}

nodes was generated randomly so that different skills were indexed from 1 to 3. In ’technicians’ column, the number refers to the number of technicians in the test case set-up. ’All skills’ means that each every technician possesses all the possible skills that can be required, making the skill matching irrelevant in the test case. ’Mixed skills’ means that not all technicians possess all the skills.

Eleven different parameter settings were used in the test runs and are given in table 6.2. Each test case was run using each parameter set 15 times to enable making some conclusions about the algorithm’s average behaviour, giving a total of $8 \times 11 \times 15 = 1320$ test runs.

6.4 Results of the experiment

The results of the test runs with respect to different parameter configurations are summed up in figure 6.3. Figure shows for each test case which set of parameters gave the best and the worst performance in respect of solution quality. Parameter set(s) that produced the best solutions of all 15 test runs is listed for each test case, as well as the parameter set(s) that gave best performance when measured by the average of the best fitness over test runs. Also, the parameter set producing the ’fittest’ population, i.e. population with best average fitness in the end, is listed. However, best average fitness in the end doesn’t automatically mean best performance, for it can also mean that the population tends to converge towards one good solution quickly without enough diversity left to produce even better solutions.

From figure 6.3 we see that to solving the easiest test case the selection of parameter set had little effect, but with the other test cases the parameters had clear influence. The following cautious conclusions are drawn based on these results: partially matched crossover is not suitable for the test case studied here, at least not as the only crossover method applied; increasing the size of the population doesn’t automatically mean improved solutions; that with population size of 100 tournament selection parameter should be more than 2; applying different crossover methods improves the overall performance of the GA; and that mutation rate around 0.05 is a good choice.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Population size</th>
<th>Mutation probability $p$</th>
<th>Tournament selection parameter $k$</th>
<th>Percent of individuals to copy directly to the next generation</th>
<th>Probabilities for using different crossover methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>0.01</td>
<td>2</td>
<td>0.01</td>
<td>PMX: 1/3, CX: 1/3, OX1: 1/3</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>0.05</td>
<td>2</td>
<td>0.01</td>
<td>PMX: 1/3, CX: 1/3, OX1: 1/3</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>0.1</td>
<td>2</td>
<td>0.01</td>
<td>PMX: 1/3, CX: 1/3, OX1: 1/3</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>0.05</td>
<td>3</td>
<td>0.01</td>
<td>PMX: 1/3, CX: 1/3, OX1: 1/3</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>0.05</td>
<td>5</td>
<td>0.01</td>
<td>PMX: 1/3, CX: 1/3, OX1: 1/3</td>
</tr>
<tr>
<td>6</td>
<td>200</td>
<td>0.01</td>
<td>2</td>
<td>0.01</td>
<td>PMX: 1/3, CX: 1/3, OX1: 1/3</td>
</tr>
<tr>
<td>7</td>
<td>200</td>
<td>0.05</td>
<td>2</td>
<td>0.01</td>
<td>PMX: 1/3, CX: 1/3, OX1: 1/3</td>
</tr>
<tr>
<td>8</td>
<td>200</td>
<td>0.1</td>
<td>2</td>
<td>0.01</td>
<td>PMX: 1/3, CX: 1/3, OX1: 1/3</td>
</tr>
<tr>
<td>9</td>
<td>100</td>
<td>0.05</td>
<td>2</td>
<td>0.01</td>
<td>PMX: 1, CX: 0, OX1: 0</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>0.05</td>
<td>2</td>
<td>0.01</td>
<td>PMX: 0, CX: 1, OX1: 0</td>
</tr>
<tr>
<td>11</td>
<td>100</td>
<td>0.05</td>
<td>2</td>
<td>0.01</td>
<td>PMX: 0, CX: 0, OX1: 1</td>
</tr>
</tbody>
</table>

Table 6.2: Parameter sets used in test runs. With each parameter set, the test run is terminated when there’s been no progress in the last 50 generations.
Figure 6.3: Parameter sets giving best and worst performance for each test case.
In table 6.3 the results with the best parameter set for each test case are given. In the optimal solutions for the original Solomon test problem C101 with 25 customers the number of vehicles is 3 and the total travel distance 191.3. For the same problem with 50 customers the number of vehicles is 5 and the total travel distance is 362.4. In the original problem set-up, all the customer nodes must be visited, capacity constraints as well as time window constraints satisfied, and number of vehicles is minimized first and travel distance second. The optimal solutions were first reported by [31] and were obtained with a branch-and-bound algorithm. The results are not directly comparable because of the differences of the problem definition but give at least some baseline for a comparison. It was also be noted that the execution times with the GA are relatively high and especially compared to the couple of seconds in which e.g. the branch-and-bound algorithm introduced in [31] was able to solve C101 instance with 50 customers. The execution time could be much reduced by better design of data structures and recycling of intermediate calculation results, which questions were mostly omitted here.

Comparison shows, that for the 25 node cases the GA implemented here was probably able to produce close-optimal solutions, but didn’t perform that well with bigger problem instances. Also the variance of the results is relatively high in all but the most easiest test cases, which can be seen by comparing the average of the best solution over all 15 test runs to the single best solution found for the same instance. This behaviour was common and shows that the algorithm is not very robust but quite prone to random variation. Figure 6.5 illustrates this by showing the evolution process in the best and worst test runs for test case 4 with parameter set 10. In spite of the similarity of the average quality of the initial population, the fist run is able to rather quickly to find reasonably good solutions while the other one develops slowly and then converges to a bad local optimum. There can be several reasons for this. One possibility is, that there is not always enough diversity in the initial population, but no data was stored from the test runs concerning this so this possibility cannot be analyzed further. In any case, it seems that the algorithms capability to escape mediocre local optima is not very strong. This might be improved by better local search methods and also slowing down the convergence of the algorithm when the diversity of the population starts to drop too much.

The best solution’s evolution in the best test run for test case 4 and parameter set 10 is illustrated in figure 6.5. This figure shows the gradual development of the solution from the randomly generated initial guess towards the optimum. It is known from the C101 instance that the time window constraints allow one technician to visit all customers in one cluster in one tour, from which we know the final solution shown here is not optimal. We can see that only one task should be relocated to obtain this. It is known that a pure GA often fails in fine-tuning of the solution near optimum and therefore needs support of e.g. a local search procedure. Here, this is demonstrated as the GA was not able to relocate the one ‘misplaced’ task in 50 iterations, which indicates a clear need for better method than pure randomization to decide which tasks the local improvement should concentrate on. There should be means to recognize ‘good’ and ‘bad’ partial solutions and to use this knowledge, but such a feature was not implemented in the scope of this
Figure 6.4: Evolution in the best and the worst test run for test case 4 and parameter set 10.
<table>
<thead>
<tr>
<th>Test case</th>
<th>Param set</th>
<th>Number of tasks</th>
<th>Total route length</th>
<th>Number of tasks</th>
<th>Total route length</th>
<th>Number of tasks</th>
<th>Total route length</th>
<th>Average number of iterations</th>
<th>Av. exec. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>10.90</td>
<td>59.78</td>
<td>11</td>
<td>59.49</td>
<td>11.00</td>
<td>59.49</td>
<td>77.27</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>23.40</td>
<td>239.82</td>
<td>25</td>
<td>191.81</td>
<td>25.00</td>
<td>207.11</td>
<td>231.40</td>
<td>6.9</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>24.90</td>
<td>274.40</td>
<td>25</td>
<td>260.99</td>
<td>25.00</td>
<td>273.54</td>
<td>147.73</td>
<td>4.4</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>32.18</td>
<td>260.18</td>
<td>33</td>
<td>194.15</td>
<td>32.33</td>
<td>259.55</td>
<td>150.60</td>
<td>8.5</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>29.47</td>
<td>327.39</td>
<td>31</td>
<td>368.67</td>
<td>29.60</td>
<td>327.64</td>
<td>121.53</td>
<td>5.6</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>49.82</td>
<td>566.02</td>
<td>50</td>
<td>437.40</td>
<td>50.00</td>
<td>563.76</td>
<td>272.67</td>
<td>29.5</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>49.92</td>
<td>674.15</td>
<td>50</td>
<td>558.39</td>
<td>50.00</td>
<td>672.67</td>
<td>214.87</td>
<td>20.5</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>49.83</td>
<td>522.63</td>
<td>50</td>
<td>440.41</td>
<td>50.00</td>
<td>521.73</td>
<td>145.73</td>
<td>12.7</td>
</tr>
</tbody>
</table>

Table 6.3: Results with the best parameter set for each test case. Averages are calculated over the 15 test runs.

It was also noticed, that even the rather harsh method of eliminating infeasible solutions from the initial population provided much better results than an initial population with plain randomized permutations of the tasks and tour limiters. On the other hand, the effect of this method on the diversity of the population was not considered further. It was only assumed that the randomized selection of the tasks to be added to the tours would suffice to prevent any bias to any specific region of the search space.

7 Conclusions

This work considered applying genetic algorithms on routing and scheduling problems that can be either reduced to or derived from the well-known Travelling Salesman Problem or the Vehicle Routing Problem. This type of problems are used to model problems arising in e.g. logistics and transportation industries. First, literature study was given, where different problems and solving methods applied on them were introduced, and the role of genetic algorithms in solving them was discussed. Then, the experimental part followed. The definition of the test problem was given, GA application developed with Java to solve it was introduced, and the test data and results of the test runs were given and analyzed.

Based on the literature study it can be stated that genetic algorithms are worth consideration when choosing a method for solving hard routing and scheduling problems. Exact methods are not an option for solving routing and scheduling problems of realistic size with much more complexity than the basic TSP, so in practice other methods must be employed when optimizing e.g. complex transportation systems. It seems
Figure 6.5: Evolution of the best solution in the best test run for test case 4 and parameter set 10.
that hybridization of metaheuristic search strategies with other methods, either exact or heuristic, is the key to success. Genetic algorithms have been playing a part in this development, but it is impossible to state any one best algorithm or combination of search methods as the success or failure of an approach can vary greatly depending on the problem definition, implementation details, and parameter configuration.

During the experimental part, it became very clear that designing and configuring a good genetic algorithm for a complex routing and scheduling problem is not a simple task. Exhaustive amount of experimenting is required to get the GA parameters optimized; in a more advanced approach this would probably be automatized, perhaps with the use of another GA. As GAs are stochastic procedures, it is also obvious that a lot of testing is needed to establish an algorithm as a proved, robust solving method. Leaving out more sophisticated local search algorithms in the GA implementation of the experimental part probably doomed it to modest success. The next interesting task in developing the algorithm further would be to implement a better local search procedure to improve the solution quality. It was also clearly demonstrated in both the literature study and the experimental part of this work, that to successfully apply a GA - or any metaheuristic approach - to a highly complex problem space, to understand the mechanisms of the metaheuristic alone is not enough but a strong understanding of the underlying problem is essential to make it possible to embed the needed problem-specific improvement methods to the general problem solving framework.

References


