Baltic J. Modern Computing, Vol. 1 (2013), No. 3-4, 161-185

Genetic algorithms and VRP: the behaviour of a crossover operator

Gintaras VAIRA and Olga KURASOVA

Vilnius University, Institute of Mathematics and Informatics,
Akademijos St. 4, LT 08663, Vilnius, Lithuania

Gintaras@vaira.net, Olga.Kurasova@mii.vu.lt

Abstract: In the paper, we investigate the crossover operators for a vehicle routing problem where only feasible solutions are taken into account. New crossover operators are proposed that are based on the common sequence in the parent solutions. Random insertion heuristic is used as a reconstruction method in a crossover operator to preserve stochastic characteristics of the genetic algorithm. The genetic algorithm together with the new crossover operators can be applied to different VRP problems or other problems that can be expressed as a graph and depend on a sequence of elements. The proposed crossover operators are compared with other crossovers that deal with feasible solutions and insertion heuristics.

Keywords: Vehicle routing problem, constraints, genetic algorithms, crossovers, insertion heuristic.

1. Introduction

In recent years, a vehicle routing problem (VRP) attracts much attention due to the increased interest in various geographical solutions and technologies as well as their usage in logistics and transportation. More and more logistic companies try to organize deliveries of goods better by enabling various today's technologies proposed. It can be various logistic systems coupled with widely used positioning systems, etc. One of the most important parts of a cost is a better organization of routes by solving a vehicle routing problem. For example, a better organization of fleet routes in various distribution areas – delivery of post, supply delivery to markets, fuel delivery to gasoline stations, etc. – can save fuel, money and/or time that can be used for new customers. The variations of VRP exist depending on the additional parameters and constraints; for example, a vehicle routing problem with time windows (VRPTW), with pick-up and deliveries (VRPPD), with multiple depots (MDVRP), etc. Typical VRP defines constraints that should be satisfied in solution: to ensure that all customers are definitely visited, to ensure that a customer is visited only once per vehicle route, to prevent overload of vehicles. The defined constraints should not be violated in a final solution.

One of the heuristic approaches dealing with VRP is a genetic algorithm (GA). Genetic algorithms are stochastic methods based on natural selection and genetics. They work with individuals, sometimes also called chromosomes, each representing a possible solution to a given problem. GA typically works with the initial population of solutions; with each new generation GA creates a new potential offspring, based on the selected

individuals from the previous generation using a set of stochastic transition operators. The iterative process of generations and evaluation of individuals continues until a sufficient stop criterion is met.

The success of the genetic algorithm depends on a couple of factors: selective pressure and diversity maintenance. These factors play an important role in the genetic algorithm, where the selective pressure describes intensification of search for a solution in each next generation by choosing better individuals for reproduction, and the diversity maintenance is responsible for having a non-homogeneous population. The diversity maintenance depends on the overall genetic algorithm: on the selection operator, on the crossover and mutation operators, and also on other approaches for population management. The designed genetic operators can either increase or reduce diversity in population, so the success of the genetic algorithm depends on the selected operators. Although all the operators in genetic algorithms are important and influence the efficiency of the algorithm as well as the behaviour of any other operator, in this research, we focus on genetic crossover operators used for solving VRP. Genetic operators can be specially designed for a specific problem and can produce an inadequate result when they are applied to different problems. Although, problem-specific genetic algorithms produce a more accurate result, the aim of this research is to design genetic operators that could be applied to a larger group of vehicle routing problems.

The rest part of the paper is organized as follows. Section 2 describes a VRP problem and constraints. In Section 3, the main principles of genetic algorithms and genetic algorithm approaches to VRP are reviewed. Crossover operators that deal with feasible solutions are reviewed in Section 4, where new crossover operators are proposed as well. Section 5 presents the experimental results. The last section concludes the paper.

2. Vehicle routing problem

VRP is a general name given for a class of problems, in which a set of vehicles services a set of customers. This statement was first defined by Dantzig and Ramser (1959). VRP is a generalization of a travelling salesman problem (TSP), where only one traveller is taken into account. The TSP problem is defined as a set of cities, where a single traveller needs to visit all of them and return to the starting city. The objective of the TSP problem is to find the shortest route.

The vehicle routing problem typically is described as a graph $G = (N, E)$ and a set of homogeneous vehicles $V = \{v_1, \ldots, v_t\}$, where t is the number of vehicles. The graph G consists of the nodes $N = \{n_0, n_1, ..., n_k\}$, where n_0 is a depot and $N\backslash\{n_0\}$ are k customers that need to be serviced, and edges $E = \{e_{ij}\}$, where $i \neq j$, $0<=i<=k$, $0<=j<=k$, $e_{ij}=(n_i, n_j)$. Each vehicle that services customers starts the travel from depot and finishes it in the depot as well. The objective of the typical VRP problem is to find the solution, at first, minimizing the total vehicle number required, and secondly, minimizing the length of the total travelled path (Dantzig and Ramser, 1959; Jih et al., 1996; Potvin and Bengio, 1996; Tan et al., 2001; Jung and Moon, 2002; Ombuki et al., 2002; Jih and Hsu, 2004; Alvarenga et al., 2005; Ombuki et al., 2006; Yeun et al., 2008). For the set E, the cost matrix D is defined, where d_{ij} is the cost of the edge $e_{ij}=(n_i, n_j)$, and $d_{ii} = 0$. Usually the VRP problem is treated as symmetric, where $d_{ij} = d_{ji}$. In the real world problem, the cost matrix is asymmetric and needs to be calculated from geographic data by using the shortest path algorithms. Moreover, if a vehicle set is not homogeneous, some roads can

be forbidden for certain vehicles and allowed for others. The different shortest path can exist for a different vehicle type, so a different matrix needs to be calculated for all the different vehicle types. A review of various speed-up techniques for the shortest path problem can be found in Vaira and Kurasova (2010, 2011).

2.1. VRP and constraints

The typical VRP problem can be extended by adding additional constraints and other parameters to the problem. The MDVRP problem includes additional depot nodes and CVRP includes load capacity limitation for a vehicle. VRP with time windows (VRPTW) is an extension, where time window constraints are added. The time window constraint defines a time frame in which a customer can be serviced, i.e. loading or unloading of a vehicle. A vehicle may arrive earlier, but it must wait until the start of the service is possible. The VRP problem can be extended with some additional constraints, like driver working hours, time, required for a driver to take a rest, etc. Similarly, depending on additional parameters, other variants of VRP problems are defined. In Yeun et al. (2008), particular mathematical formulations can be found for each VRP, VRPTW, VRPPD, CVRP problem, where each formulation is based on a customer set, represented as nodes in a graph. Jih and Hsu (2004) proposed the problem definition, based on transportation requests as tasks to be completed.

To generalize a VRP problem, it can be divided into the following components:

- *data*, used in the problem;
- *tasks*, defined to be accomplished;
- *constraints* that should be satisfied*;*
- *objective* of the problem;

Data definition includes the graph $G = (N, E)$, which consists of the nodes N and edges E. The data definition also includes a set of vehicles $V = \{v_1, \ldots, v_t\}$. The set of nodes can be divided into subsets of a) N_d – depots, b) N_c – customers, c) N_o – other nodes that can be divided into rest areas, gasoline stations, etc. For data definition, a start position is assigned to each vehicle v_i, where the initial node can be marked as n_i^{init}. Additional data, like drivers and their properties or types of the goods, can be defined within the problem.

Tasks, similarly as in Jih and Hsu (2004), define a set of targets to be achieved. Let us define a set $R = \{r_1, \ldots, r_q\}$ as a set of q requests and $T = \{t_1, \ldots, t_p\}$ as a set of p tasks to complete requests. Each request r_i can be expressed via a set of tasks $r_i = \{ t_{i1}, t_{i2},...\}$, where $\forall t_{ij} \in T$, $|r_i| > 0$, $\forall r_i \in R$, $r_1 \cap \ldots \cap r_q = \varnothing$ and $r_1 \cup \ldots \cup r_q = T$. The main difference between the request and task is that the task can be processed one at a time by a single vehicle and the requests may be processed in parallel. The task can have other smaller subtasks, in such a way granularity increases, however, for VRP problems, the task does not require to be split to smaller tasks, if it means "to be processed one at a time by the vehicle". In the VRP problem, each task t_i is defined as $t_i = (n_i^{start}, n_i^{end})$, where the node $n_i^{start} \in N$ is a start node of the task t_i and $n_i^{end} \in N$ is the end node. To complete the task t_i, at first, a vehicle needs to arrive to the node n_i^{start} to start service, and then to complete service at the node $n_i^{end.}$.

For VRP problems that deal with a delivery of cargo, the request can be defined as $r_i = \{t_i^+, t_i^-\}$, where r_i is a request to deliver cargo from one place to another and to complete it, tasks t_i^+ (to load cargo at a specific place) and t_i^- (to unload cargo at a specific place) have to be performed. The properties of cargo are defined for each

request. Let us define a function $w(r_i)$ that evaluates the cargo capacity value $w_i = w(r_i)$. Tasks in the delivery problem can be defined as $t_i = (n_i^{start}, n_i^{end}, w_i)$ for loading/unloading of w_i at the node $n_i^{start} = n_i^{end}$. Usually VRP defines the return to the depot tasks $T^{end} = \{t_1^{end}, \ldots, t_t^{end}\} \subseteq R$, $\forall t_{vi} \in T$. An example of the task that starts and ends at different places could be found in the taxi problem, where a service of each customer starts at pickup place and ends at the destination place.

The VRP target is to complete tasks by using vehicles. Let us define a single solution of the VRP problem as $x = \{s_1, \ldots, s_t\}$, where $\forall s_j = (t_{j1}, t_{j2}, \ldots)$, $\forall t_{ji} \in T$, $s_1 \cap \ldots \cap s_t = \varnothing$ and $s_1 \cup \ldots \cup s_t = T$. $\forall s_j$ defines a sequence of tasks assigned to the vehicle $v_j \in V$ and $|x| \leq |V|$. Let us define a function $F_r(x)$ that evaluates the solution x for incompleteness of requests and $f_r(x)$ that evaluates a single request for task incompleteness.

$$F_r(x) = \sum_{r \in R} f_r(r, x);$$

$$f_r(r, x) = \sum_{t \in r} f_t(x, t);$$

$$f_t(x, t) = \begin{cases} 0 & \text{if the task } t \text{ is completed in the solution } x; \\ 1 & \text{otherwise.} \end{cases}$$

All the requests are completed, if $F_r(x) = 0$.

Constraints define restrictions to the problem that usually reflect real life situations. Let us define a set of constraints C, where $c \in C$ defines a single constraint. The constraints can be defined for a task (i.e. time window), for a vehicle (i.e. capacity), for a cargo, etc. One of the constraints of the delivery problem (VRPPD) is that t_i^+ needs to be completed before t_i^- or return to the depot task should be completed after all the other tasks. So, the constraint can define the order of tasks. Let us define a function $F_c(x)$ that evaluates violation of constraints in the solution x, and $f_c(x)$ that evaluates violation of the single constraint $c \in C$:

$$F_c(x) = \sum_{c \in C} f_c(x)$$

$$f_c(x) = \begin{cases} 0 & \text{if the constraint } c \text{ is satisfied} \\ z, \text{ where } z \in \mathbb{R}, z > 0 & \text{otherwise} \end{cases}$$

The solution x does not violate any constraint, if $F_c(x) = 0$.

Objective. The objective of typical VRP is to minimize the number of the used vehicles and then to minimize the length of the total travel path. So, the objective is, at first, to minimize the function $f_v(x)$, then $f_d(x)$, in addition, the equalities $F_c(x) = 0$ and $F_t(x) = 0$ need to be satisfied:

$$f_v(x) = \sum_{j=1}^{|x|} f_a(s_j), \forall s_j \in x$$

$$f_d(x) = \sum_{j=1}^{|x|} d_l(s_j), \forall s_j \in x$$

$$f_a(s_j) = \begin{cases} 1 & \text{if } d_l(s_j) > 0 \\ 0 & \text{otherwise.} \end{cases}$$

$$d_l(s_j) = \sum_{i=1}^{|s_j|} l\left(n_{j_i}^0, t_{j_i}\right), \text{where } \forall t_{j_i} \in s_j, n_{j_i}^0 = n_{j_{i-1}}^{end}, n_{j_0}^{end} = n_j^{init}$$

$$l(n_i, t_j) = l\left(n_i, n_j^{start}\right) + l\left(n_j^{start}, n_j^{end}\right)$$

$$l(n_i, n_j) - \text{distance between the nodes } n_i \text{ and } n_j$$

Different objectives could include different minimization/maximization functions, i.e. a real life problem could be defined, where the feasible solution that completes all the tasks is not possible. The objective of such a problem could be to find a solution, where the maximum number of requests is completed.

3. Genetic algorithm approaches for VRP

3.1. Main principles of the genetic algorithm

Genetic algorithms are based on ideas of evolution theory (Holland, 1975). The main principle here is that only the fittest entities survive (Reid, 2000; Jung and Moon, 2002; Lukasiewycz et al., 2008). A genetic algorithm can be divided into several sub-parts that are used in this algorithm: representation, fitness function evaluation, initialization, selection, recombination (crossover and mutation), termination. The whole genetic algorithm process is described in Fig. 1.

1. The initial population is created, where each individual is expressed via defined representation;
2. The fitness function is evaluated for the initial population;
3. The subset of the population (so-called parents) is selected that will be used in recombination operators to generate offspring;
4. The crossover operator is applied to parents to create new offspring;
5. The mutation operator is applied with a certain probability;
6. The fitness function is evaluated and the individuals with the worst fitness value are removed;
7. If the stop criterion is not met, go to Step 3.

Fig. 1. Steps of a genetic algorithm

Representation. The classical genetic algorithm paradigm deals with the literal string encoded form of solutions, called chromosomes. A chromosome is the representation of a single solution of the problem and requires additional encoding/decoding steps to be defined in the algorithm. Genetic algorithm approaches can be divided into two sets: algorithms that are applied to the VRP problem represented as a chromosome, and algorithms that skip the encoding/decoding step. In genetic algorithms, where encoding/decoding is bypassed, a single solution is usually called individual. The TSP problem has a single constraint – all cities should be visited. The solutions of the TSP problem are vectors of the nodes:

$$x_{tsp} = (n_{i_1}, n_{i_2}, \ldots, n_{i_k})$$

A single solution within a problem can be defined as $x_{tsp} \in S_{tsp}$, where S_{tsp} is the whole search space of the TSP problem and $|S| = k!$. Each TSP solution can be easily encoded as a queue of indexes (chromosome):

$$i_1, i_2, \ldots, i_k$$

Such encoding can be useful for any problem that can be expressed as TSP, for example in computer wiring, scheduling of jobs on a single machine, etc. (Deep and Adane, 2011). However, it is worth mentioning that such a representation does not hold any additional information.

Population and initialization. In initialization, the initial set of chromosomes, also called as initial population, is created. The size of initial population is important for the overall genetic algorithm. A small size of the initial population can lead to only a local optimum result, while a larger initial population gives a higher probability that the global optimum will be found, however, computation time increases (Reid, 2000). While the TSP problem is defined as a complete graph, usually the initialization is done by randomly selecting a node and assigning it to the route.

Evaluation and selection for reproduction. The selection operator is used to identify chromosomes which will be used in reproduction and will survive in the next generation. Different techniques can be used in selection operators, however, usually a natural selection process is simulated, where the "strongest" individuals are used in reproduction. One of the methods for selection is called Roulette Wheel. The name explains the method: a wheel is divided into parts according to the fitness of the individuals in population, where better individuals get a larger part of the wheel and the worst individuals get a small part of the wheel. So, the probability to be selected is directly proportional to the fitness value. When the wheel is spinning, a pin on the wheel will most probably point to a better individual. The individuals with a higher fitness value have a higher probability to be selected for reproduction, and vice versa.

The second method for selection is called Ranking. In the Ranking method, all individuals in population are sorted according to the fitness value $f(x)$ to assign ranks, where the individual with a better fitness value gets a higher rank. In TSP, usually $f(x)$ defines the length of the total path travelled, however, this function can include additional characteristics and measurements in order to keep individuals in the population. If in the Roulette Wheel method the fitness value is used when assigning a probability to be selected, in the Ranking method individuals are selected proportionally to the rank.

Another method for selection is called Tournament Selection. This method uses characteristics from the Ranking method, but, in contrast to it, Tournament Selection ranks only a subgroup of individuals. At first, two subgroups from a population are selected. Each subgroup must contain at least two individuals. The individuals are ranked within a group like in the Ranking selection operator. The best individual from each group is selected for reproduction, and the worst individuals are chosen to leave the population. To generate l new offspring in each iteration, assuming that two new offspring will be generated from two selected parents, l subgroups have to be selected from the population (Alvarenga et al., 2005).

Recombination. An important part of the genetic algorithm is recombination operators. The crossover operator simulates the reproduction between two individuals, where the created offspring inherit some characteristics from parent individuals. Many crossover and mutation operators exist that operate with a chromosome encoded as a literal line of symbols or numbers. The list of common crossovers used for solving TSP as well as for solving the VRP problem is as follows (Jih et al., 1996; Ombuki et al., 2002; Tan et al., 2006; Kumar et al., 2012):

- Partially matched crossover (PMX),
- Cycle crossover (CX),
- Ordered Crossover (OX),

- Uniform Crossover (UX),
- Uniform Order Crossover (UOX),
- Edge Assembly Crossover (EAX),
- Merge crossovers (MX1, MX2), etc.

Crossovers listed above produce an encoded chromosome or chromosomes as a result, that need to be decoded for evaluation. In Jih et al. (1996), we can find a review, where the Uniform Order Crossover is mentioned as a good approach for solving VRP. It is an analogue of the Uniform Crossover translated into an order-based form:

1. a binary string of the same length as parent chromosomes is generated;
2. the first intermediate offspring preserves nodes from the second parent, where the generated string contains "1";
3. permute nodes from the second parent where a binary string contains "0" in the same order as they appear in the first parent;
4. fill these permuted elements in the gaps of the first intermediate offspring;
5. switch the parents and perform the steps 2 – 4 to create the second offspring (Jih et al., 1996).

Figure 2 provides an example of a uniform order crossover.

Binary string:	0 1 1 0 1 1 0 0
1st parent:	1 2 3 4 5 6 7 8
2nd parent:	3 5 1 8 4 7 2 6
Intermediate offspring	
1st offspring:	– 5 1 – 4 7 – –
2nd offspring:	1 – – 4 – – 7 8
Generated offspring:	
1st offspring	2 5 1 3 4 7 6 8
2nd offspring	1 3 5 4 2 6 7 8

Fig. 2. Uniform order crossover (UOX)

A mutation operator is used with intention to prevent getting stuck in the local optimum and increase a probability to find the global optimum (Hong et al., 2002). In the mutation operator, a new offspring is created from the selected single solution by changing some characteristics within it. In the genetic algorithm, crossover and mutation operators are applied by a predefined probability. We can find the values for these probabilities proposed in Srinivas and Patnaik (1994), Hong et al. (2002). In the adaptive probability approach, the probabilities are adjusted during computation depending on the current population characteristics, flow of the computation and other parameters. The simplest mutation operator extracts a single gene (an element of the chromosome) and places it back to the chromosome by randomly choosing a new location (Potvin and Bengio, 1996).

Diversity maintenance and selective pressure. Two important factors of the genetic algorithm are population diversity and selective pressure. These two factors are related: if the selective pressure is increasing, the population diversity decreases and vice versa. The selective pressure is a task of the selection operator. Too-weak selective pressure

can lead to ineffective search. The selection operator as well as other operators influences overall diversity of population. A good performance is achieved, while maintaining the diversity of population as long as possible. The mutation is important in the variation of individuals, when the population becomes homogeneous (Srinivas and Patnaik, 1994). The population diversity can also be maintained by increasing the size of the population or by having greater mutation rates, however, the performance factor should be taken into account. Other techniques are also used. A common approach is to avoid duplicates in the population. It means that the generated offspring is not allowed in the population, if it is the clone of the existing individual.

Termination. The genetic algorithms are stochastic methods that could run forever, if a termination criterion is not applied. A simple stop criterion could be the maximum computation time or the maximum iteration number. In Berger and Barkaoui (2004), iterations are counted from the last successful improvement of the best individual and the process is stopped, when the maximum limit is reached. The probability to improve the best individual decreases proportionally to the computation time. So, the number of iterations without improvement is directly proportional to the probability of improvement. A large value would increase the computation time and possibly a better solution will be found, while a low value will involve an early stop with a poor solution found.

Many derivative GA approaches can be found in the literature, some of which include multiple populations, dynamically chosen genetic operators or any hybrids with other known heuristic approaches (Yeun et al., 2008). However, the main principles of the genetic algorithm remain the same. In the rest of the paper, we will investigate a genetic algorithm implementation for VRP. The main research presented here is based on crossover operators used for solving VRP, and their influence on the whole genetic algorithm.

3.2. Genetic algorithm and VRP

As already mentioned, VRP is a generalization of the TSP problem. VRP includes additional components, i.e. fleet of vehicle, and additional constraints. An additional component of the problem can affect computation and even require to design the problem specific genetic operators. Genetic algorithm approaches to solve the VRP problem can be categorized according to the following features:

- *Representation*. Solution in GA can be encoded as a chromosome (expressed as a literal string), or unencoded, where encoding of the solution within chromosome is not addressed.
- *Feasibility handling*. Genetic algorithm operators can be designed to preserve the feasibility of individuals within a population or allow the generation of infeasible individuals.

An example of VRP solution, where 3 routes are used to service customers expressed as a chromosome is as follows:

$$| n_{e1}\, n_{e2} \ldots | n_{f1}\, n_{f2} \ldots | n_{g1}\, n_{g2} \ldots |$$

The standard genetic operators can be applied to such a chromosome, however, such a representation does not hold any problem specific information and, depending on the encoding approach, the selected genetic algorithm can be ineffective. Different approaches for encoding the VRP solution can be found in the literature, i.e. in Thangiah

et al. (1991), a chromosome representation based on the angles of vectors starting from a depot node is proposed, where the VRP problem is treated as a planar graph problem (Thangiah et al., 1991; Jung and Moon, 2002). Researches can be found that compare crossover operators designed to work with the chromosome representation (Jih et al., 1996; Misevičius and Kilda, 2005; Kumar et al., 2012).

For a constrained problem, there exist feasible and infeasible search spaces F and U. Let us define the whole search space S, then $F \subseteq S$, $U \subseteq S$, $U \cup F = S$, $U \cap F = \varnothing$. Approaches, where a solution is represented as a chromosome or where solutions are allowed to be generated in the infeasible search space U, require additional approaches for constraint handling. The following approaches are used to deal with the infeasibility in genetic algorithms:

- *Penalty*. A penalty method is widely used in genetic algorithms for constrained problems. The main target is to add a significant value to the fitness value for the generated offspring that violate constraints. Two different functions can be applied to a solution depending on the search space to which the solution belongs: $f_f(x)$, where $x \in F$, and $f_u(x)$, where $x \in U$. The relation between these two functions can be defined via an additional function, called a penalty function, $p(x)$: $f_u(x) = f_f(x) + p(x)$. The penalty method is directly applied to the fitness value, where the highest benefit of the penalty function is to adjust the ranking mechanism in the population and increase the selective pressure on the feasible individuals
- *Repair*. The second approach for feasibility handling is a repair method. The repair method defines the transition function $y = R(x)$, where y is the repaired version of x, such that $y \in F$, and $x \in U$. The repair of individuals can be used for evaluation only or to replace the previous (infeasible) individual.
- *Multi-objective*. In a multi-objective approach, the constrained problem is transformed in to a multi-objective problem. In Berger and Barkaoui (2004), Ombuki et al. (2006), Tan et al. (2006), the Pareto ranking method is used to solve the VRPTW problem expressed as multi-objective, where Pareto ranking, similarly to the penalty approach, is used to adjust the ranking mechanism of the genetic algorithm.

Local route improvement algorithms are considered for a chromosome improvement as an additional step of the genetic algorithm. Multiple improvement algorithms are also considered in computation to better exploit their characteristics. The local route improvement is used to add additional intensification to the genetic algorithm with a view to increase the convergence speed (Potvin and Bengio, 1996; Jung and Moon, 2002; Berger and Barkaoui, 2004; Nagata and Bräysy, 2009). In Jung and Moon (2002), usage of Or-opt, crossover and relocation methods together are investigated for the improvement of routes. Another known improvement algorithm, commonly used in VRP implementations is 2-opt, and also its generalization 3-opt and k-opt.

In this research, we deal with the operators that preserve the feasibility of individuals. Encoding of a solution as a chromosome is not considered in this research. In the following section, we will mainly concentrate on crossover operators of the genetic algorithm that preserve feasibility. New crossover operators will be proposed and compared to other genetic algorithm approaches for the VRP problem. No additional repair method is considered to keep the solution in a feasible search space in the proposed crossover operators.

4. Crossovers for VRP with constraints

Genetic algorithm approaches that deal with infeasible individuals require additional approaches to intensify a search to a feasible search space. Usually these approaches require a specific improvement or repair methods to avoid situations, where repair of a single constraint can have a negative impact on other constraints. Another approaches are to avoid infeasibility in the created solutions.

Depending on the defined data and constraints, a feasible solution that finishes all the tasks could not be possible to be created. Solutions are possible, where either some of the constraints are not satisfied or not all the tasks are completed. In this paper, solutions are treated to be feasible, if all the constraints are satisfied. However, the solution can be incomplete – not all the tasks are completed in the constructed solution. The objective requires to be extended to handle this approach: to find the solution that accomplishes a higher number of tasks.

We have defined the problem solutions as a set of sequences constructed from the tasks. Such a definition differs from a widely used definition for VRP, where the solution is defined as a sequence of nodes visited by separate vehicles, where each sequence is called a route. Tasks are accomplished "travelling by the vehicle in the graph". The sequence of nodes can be expressed via a sequence of tasks, where the solution does not have duplicated tasks. The sequence of tasks can be expressed via the sequence of nodes, however, the same node can be visited a couple of times per solution, if a couple of different tasks includes it. In this paper, the methods and algorithms, proposed by other authors, will be expressed by a sequence of tasks instead of the original expression – route of nodes. For a better explanation, sequence of tasks will be called a route of tasks.

In Reid (2000), probability functions are defined to find a feasibility-preserving two-point crossover for a linear constraint problem. However, for a highly constrained problem, where a feasible search space is reasonably small compared to an infeasible search space, only a half-feasible crossover with a single boundary point is discussed.

Usually, in order to create feasible solutions, various approaches of construction heuristics are taken into consideration. Construction heuristics can include the minimization function and work as the stand-alone algorithms. Insertion heuristics are one group of construction heuristics, where the routes are constructed by inserting all the tasks one by one into the routes. In Alvarenga et al. (2005), Ombuki et al. (2006), a push-forward insertion heuristic (PFIH**)** is used to create an initial solution and also as part of the crossover operator. PFIH originally was defined for the VRPTW problem by Solomon (1987). PFIH starts by selecting the first task and forming the initial route from a depot. The algorithm inserts all the other tasks into the constructed route by minimizing the insertion cost function for each task. The concept "push-forward" originally means checking pushed-forward values of all the subsequent tasks in the route (Tan et al., 2001). In PFIH, the first task of the new route is identified deterministically, where the task to be inserted is the one that is distant from the depot, not too far from the last inserted task in the previous route, and that has an early time window. Other tasks are inserted by minimising the insertion cost by evaluating insertion of all the free tasks in all the existing insertion positions in the route. An important characteristic of PFIH is that insertion of the task is possible, only if no constraint is violated.

When the insertion heuristic is used as a part of the crossover operator, it plays an important role in the general genetic algorithm approach: a random insertion can

increase the diversity of the population, whereas usage of the minimization function in the insertion can give better results initially, but reduce the diversity. For further crossover operators, we define the following questions:

- what information is taken from parents to create partial (or full) offspring?
- which insertion approach is used to insert unassigned tasks back?

Best Cost Route Crossover (BCRC), proposed in Ombuki et al. (2006), creates two offspring from two parents. For a better explanation, let us denote the parent solutions as x_{p1} and x_{p2}, denote the offspring solutions as x_{o1} and x_{o2} and intermediate offspring solutions as x'_{o1} and x'_{o2}. The defined crossover creates an offspring solution in the following steps:

1. T_{temp} = select a random route $s_r \in x_{p2}$,
2. create a partial solution $x'_{o1} = x_{p1} \backslash \{t \in T_{temp}\}$,
3. create x_{o1} by inserting the task $t \in T_{temp}$ into x'_{o1}, by randomly selecting a task from T_{temp} and inserting a task with the minimal insertion cost. Tasks are inserted into the existing routes. If it is not possible to insert due to constraint violation, a new route is created,
4. create x_{o2} by swapping the parent solution and repeating steps 1-3.

The defined crossover operator takes a single parent, forms an offspring from it, partly destroys it and reconstructs it back (*reconstruction* is not the same as *repair*, where *repair* is used to create a feasible version of an infeasible solution). For the stage of destruction, Ombuki et al. (2006) proposed to use the second parent as a reference, where a single randomly chosen route provides information which tasks should be removed from the offspring solution. A couple of cases can be noticed in such an approach: a) if a solution has a lot of small routes, a single route could include a small set of tasks, where removal of a small number of tasks from the solution could not give the expected intensification result; b) if the problem is defined only for a single vehicle (i.e. TSP), the resulting solution will have only one route and, in the destruction stage, the whole route will be destroyed. The first case can be solved by increasing the number of routes selected as references. However, the question, what useful information is shared between the parents and why this approach is better than random task remove, is not explained in Ombuki et al. (2006). The design of BCRC leads to a minimization of routes, because the nodes to be removed can form the route in the second parent solution, and there exist a probability that the whole route will be removed in the offspring solution.

*SBX***.** In Potvin and Bengio (1996), two crossover operators are proposed that repair the generated offspring by removing correlating tasks from it and reinserts them by minimizing the additional detour. The first crossover, called a Sequence-Based Crossover (SBX), selects two routes from the parent solutions and merges them by selecting a split place (break-point) in each route:

1. $x'_{o1} = x_{p1}$;
2. select a random route s_{r1} from x_{p1} and a random route s_{r2} from x_{p2};
3. create a new route s_{new} by adding tasks from s_{r1} starting from the beginning till a randomly selected place;
4. append tasks to s_{new} from s_{r2} starting from a randomly selected place till the end;
5. remove duplicates from s_{new} if such exist;
6. $x'_{o1} \backslash \{t \in s_{new}\}$ – remove the tasks from x'_{o1} that belong to the new route s_{new};

7. remove s_{r1} from x'_{o1}, add $\forall t \in s_{r1}$ to T_{temp};
8. add s_{new} to x'_{o1};
9. create x_{o1} by inserting the tasks $t \in T_{temp}$ to x'_{o1} by evaluating the insertion cost function;
10. create x_{o2} by swapping the parent solution and repeating steps 1-9.

RBX. The second crossover proposed in Potvin and Bengio (1996) is called a route-based crossover (RBX). In this crossover, a route from one parent replaces one route from the second parent:

1. $x'_{o1} = x_{p1}$;
2. select a random route s_{r2} from x_{p2};
3. $x'_{o1} \setminus \{t \in s_{r2}\}$ – remove the tasks from x'_{o1} that belong to the route s_{r2};
4. remove a random route s_{r1} from x'_{o1}, add $\forall t \in s_r$ to T_{temp};
5. add the route s_{r2} to x'_{o1};
6. create x_{o1} by inserting the task $t \in T_{temp}$ to x'_{o1} by evaluating the insertion cost function;
7. create x_{o2} by swapping the parent solution and repeating steps 1-6.

Both crossovers, SBX and RBX, add some parts from both parents to the final solution. The first crossover merges two routes from the opposite parents, so it can be applied in the cases where parent solutions have only one route. If solutions have more than one route, the probability to select parent routes for a crossover, such that the created offspring were competitive in the population, decreases, when the number of routes increases. The operation of removing duplicates in the route might be insufficient. A merge of two routes at random positions can involve a violation of constraints in the offspring solution. For example, let us have a VRPTW problem, where time window constraints are defined for all tasks. Let us have a break-point selected in the first route s_{r1} after the task $t_{r1,i}$, and a break-point selected in the second route s_{r2} before the task $t_{r2,j}$. The new route constructed will connect two routes to the following route ($t_{r1,1}$, …, $t_{r1,i}$, $t_{r2,j}$, …). The task $t_{r2,j}$ could have an early time window, then, since it was at the beginning of the route s_{r2}, the time window constraint will probably be violated when the task $t_{r2,j}$ is added to the "late" position in the new route. An additional constraint check should be applied to avoid a constraint violation in the offspring.

The crossover RBX preserves a feasibility in the offspring. If the parent routes are feasible, then the routes in the offspring remain feasible. Randomly selected routes in both individuals may have no common tasks, so the removal of duplicate tasks and removal of a randomly selected route can reduce the number of routes in the intermediate solution. So, this crossover has a possibility to minimize the number of routes. However, after the reconstruction the number of routes can still be increased. If solution has a larger number of routes, then the approach can be adjusted to take a larger number of routes from the second parent. However, there exists a limitation if there is only one route in the parent solution.

LRX. The crossover used in Alvarenga et al. (2005) is similar to that of RBX described above, because it combines the routes from the parent individual by evaluating the number of tasks in the routes (let us call it Largest Route Crossover (LRX)). Originally, the genetic algorithm approach was defined to handle infeasibility as well. In the original crossover, infeasible routes are skipped in the offspring and added to the list of unassigned tasks. This crossover can also be applied to feasible solutions:

1. $L_r = \varnothing$ is a list of routes;
2. add $\forall\ r \in x_{p1}$ to L_r;
3. add $\forall\ r \in x_{p2}$ to L_r;
4. $x_{o1} = \varnothing$ is the initial empty solution;
5. r_s = select a route from L_r with the largest number of tasks;
6. for $\forall\ r \in L_r$, $r = r \setminus \{n \in r_s\}$ - remove the tasks belonging to r_s from all the other routes;
7. add r_s to x_{o1};
8. repeat the steps 5-7 while L_r has routes with at least one task.

The LRX crossover produces only one offspring. Stochastic PFIH was used as a reconstruction method to insert unassigned tasks in the LRX crossover. If in the deterministic PFIH, the first task (initial route) is found deterministically, in the stochastic PFIH, each new route is started by choosing an unassigned task randomly. By inserting routes with a larger number of tasks, the described crossover intensifies the first objective of the VRPTW problem. So, this crossover is designed for a special problem (or a special objective) and is not effective in the cases, where parent solutions have only one route.

Reconstruction. All the crossovers described use an insertion heuristic for reconstruction of solutions. However, insertion approaches slightly differ in each crossover. In all of them, at first, tasks are inserted into the existing routes, if the constraints are not violated, and a new route created, otherwise. Such a method intensifies the route minimization objective of the VRPTW problem. Usually, in GA, intensification is a task of the selection operator and depends on a selective pressure. The usage of intensification in the crossover operator needs to be adequate to the intensification in the selection operator, otherwise, the crossover will, most probably, generate an offspring that will not survive in the population.

Let us express the route of tasks as a graph $G_r^T = (T_r, A_r)$, where the set $T_r = \{t_r^{init}, t_{r1}, \ldots, t_r^{end}\}$ defines the set of tasks assigned to the route, and t_r^{init} represents the start of the route at the node n_r^{init}. Expression of route via tasks eliminates the possibility of duplicate entries, where the expression of a route via nodes could give the same node, visited a couple of times, if the same node is part of a couple of different tasks. The arcs from the set A_r connect the tasks: $\forall a_{ri} \in A_r$, $a_{ri} = (t_{ri}, t_{r(i+1)})$, where t_{ri} is the start of the arc and $t_{r(i+1)}$ is the end. Insertion of the new task t_m into the arc a_{ri}, means to:

1. remove the arc a_{ri} from the set A_r;
2. add the task t_m to T_r;
3. add two new arcs (t_{ri}, t_m) and $(t_m, t_{r(i+1)})$ to the set A_r.

An empty route (vehicle without tasks assigned) is still a graph with $T_r = \{t_r^{init}, t_r^{end}\}$ and $A_r = \{(t_r^{init}, t_r^{end})\}$ (Figure 3). The insertion process can be split into the following parts and intensification can be applied in both of them:

- select a task for insertion;
- select a place for insertion.

The following approaches can be used for implementing the insertion heuristic:

- Choose randomly a task and then search for the best arc to insert in. A random task selection is a stochastic approach that can be used to choose a task for insertion. This method does not affect overall intensification and corresponds to the general idea of the genetic algorithm being a stochastic

approach. Usage of the minimization function then can be applied to arc selection.

- Choose a random arc and then search for the task. It is the opposite approach to the previous one.
- Search for a task and arc at the same time by evaluating the minimization function. The minimization function could also include evaluation of the vehicle and additional information that could help to identify the best task and the best place for the next insertion.

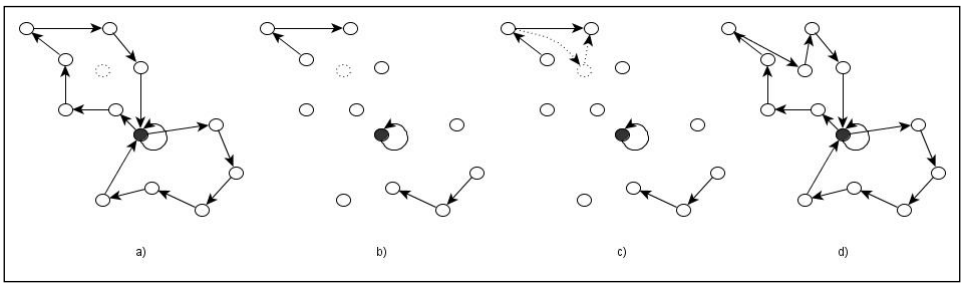

Fig. 3. Insertion with the constraint check: a) partial solution and the task (dotted) to be inserted, b) identified arcs where a feasible insertion is possible, c) found arcs with a minimal insertion, d) newly constructed solution

To select a place for the task insertion means to select an arc from the set of A_{all} (all the arcs in all the routes in the partial solution x'). For a feasible insertion, we have to evaluate the violation of all the constraints. Let us have a function $f_c(a_i, t_m)$ that checks the violation of all constraints, while inserting the task t_m into the arc $a_i \in x'$:

$$F_c(a_i, t_m) = \Delta F_c^{i,m}(x') = F_c\left(x'^{i,m}\right) - F_c(x')$$

$x'^{i,m}$ is the solution x' with the task t_m inserted into the arc a_i

The insertion of a task does not increase the constraint violation, if $F_c(a_i, t_m) = 0$. To follow the objective of the VRP problem, the arc needs to be selected by minimizing the functions $f_v(a_i, t_m)$ and $f_d(a_i, t_m)$ that evaluate the difference of objective functions, while inserting the task t_m into the arc a_i. At first, the difference of the route number is evaluated and, in the second function, the difference of the route length is evaluated:

$$f_v(a_i, t_m) = \Delta f_v^{i,m}(x') = f_v\left(x'^{i,m}\right) - f_v(x')$$
$$f_d(a_i, t_m) = \Delta f_d^{i,m}(x') = f_d\left(x'^{i,m}\right) - f_d(x')$$

In Figure 3, the overall insertion process with the feasibility check is presented, where filled circle represents a depot node, arrows represent arcs in the partial routes, a dotted circle represents the task t_m selected for insertion, dotted arrows represent possible insertion arcs.

In the following subsection, new crossover operators are proposed that are also based on insertion heuristics. However, apart from the insertion heuristics, the proposed crossover operators handle most of the negative aspects of the reviewed crossover operators and include another intensification approach.

4.1. The proposed crossover operators

The crossover operators, proposed in this paper, are based on the idea of a large neighbourhood search heuristic. The large neighbourhood search (LNS) heuristic belongs to the class of heuristics known as a very large scale neighbourhood search (VLSN) (Pisinger and Ropke, 2009). The main principle of the large neighbourhood search is as follows. For solution $x \in F$, a neighbourhood in the feasible search space can be defined as $Nh(x) \subseteq F$, where $Nh(x)$ is a function that maps the solution x to a set of solutions. Neighbourhood algorithms are an iterative process that takes the initial solution x and, in each iteration, searches for the cheapest solution x' in the neighbourhood of x: $x' = \text{argmin}_{x'' \in Nh(x)} f(x'')$. The solution x' replaces x by evaluating the acceptance function. An example of neighbourhood $Nh(x)$ can be one of the local improvement approaches, i.e. 2-opt, where the 2-opt neighbourhood is a set of solutions that can be obtained by removing two edges in solution x and adding new ones to reconnect the route. In the large neighbourhood search, the neighbourhood is defined as $Nh(x) = r(d(x))$, where neighbourhood solutions can be found by applying, at first, the destroy function $d(.)$ and then the reconstruction function $r(.)$. Other than the genetic algorithm approaches, the large neighbourhood search maintains only two solutions: the best solution found x^b and the current solution x is used that takes part in the exploration of the neighbourhood. If x is found such that $f_a(x) < f_a(x^b)$, where $f_a(x)$ is the acceptance criteria function, x^b is replaced with a new solution: $x^b = x$. The effectiveness of LNS depends on the degree of destruction, where, if only a small part is destroyed, LNS can have trouble in exploring the search space, or can be involved in the repeated re-optimization, if a very large space is destroyed (Pisinger and Ropke, 2009). The destruction method should be used such that explores the search space where the global optimum is expected to be found.

The BCRC crossover, mentioned in the previous section, involves the destruction of the parent individuals to build offspring, but the exploration depends on the route from the second parent individual. The tasks in the route from the second individual could be assigned depending on their time window constraint. Thus, it means that the removed tasks can have a low probability to change their positions in the solution. Other crossovers (SBX, RBX, LRX) convey the *union* of the solutions, where some parts from both individuals are combined with the intention to find a better solution. Such crossovers explore only a small neighbourhood and only in the cases, where additional unassigned tasks are left during the recombination of parents.

Other than the *union* crossover operators, the proposed crossover is designed to preserve common parts of the two selected individuals. The common parts could be the tasks assigned to the same vehicle, the tasks assigned to the vehicle starting from the same depot, or the tasks with the same cargo. By removing the tasks that do not belong to the common parts of solutions, the common neighbourhood of two solutions is identified. A size of the neighbourhood is inversely proportional to the size of the common parts. In the VRP problem, where the objective is to minimize the vehicle number and the path length, the sequence of tasks in the route is important. If the initial individuals in a genetic algorithm are created in a stochastic way, removal of the tasks that do not belong to the same sequence of the tasks, most probably, will remove the tasks that prolong the overall path, where long paths can lead to a larger number of routes.

The target of the proposed crossover operators is to identify the common sequence in the parent individuals, preserve it in the intermediate solution and reconstruct it in the

offspring individual. Two crossover operators (common arc crossover and longest common sequence crossover) are defined to handle the same sequence of the tasks in both parents. Each crossover produces only one offspring from two parents.

The *common arc crossover* (CAX) preserves arcs in the first parent solution, if the corresponding arcs exist in the second parent solution, where the corresponding arc has the same start and the same end.

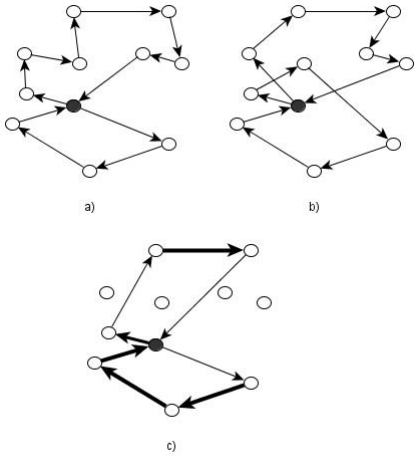

Fig. 4. Common arc crossover: a) and b) parent solutions, c) intermediate solutions with highlighted arcs common in both parents

The common arc crossover preserves the sequence between two tasks in the graph. The complexity of this crossover is O(p), where p is the total number of tasks in the problem. The example of behaviour of the first crossover is displayed in Figure 4, where a) and b) are parent solutions, and c) is the intermediate solution, where the common arcs from both parents are displayed with highlighted arrows. The arrows that are not highlighted are new arcs that connect the tasks according to their position in the first parent solution to form the route. The algorithm to find the common arcs in two parent solutions is as follows:

1. $x'_{o1} = \varnothing$;
2. $T_{temp} = \varnothing$;
3. for $\forall\ t \in x_{p1}$;
4. if the task t is the start or the end of the common arc, add the task t to x'_{o1}, otherwise, add the task t to T_{temp};
5. continue from step 3) until all the tasks in x_{p1} are evaluated.
6. create x_{o1} by inserting $\forall\ t \in T_{temp}$ into x'_{o1} by evaluating the insertion cost function.

The *Longest common sequence crossover* (LCSX) is the second crossover operator, proposed in this paper. It examines the two parent solutions by searching for the longest common sequences in all the routes. An example of the longest common sequence between two routes is displayed in Figure 5, where a) displays the first route with the indexed tasks in the route (literal string displays the indexed sequence); b) for all the tasks in the second route indexes are assigned according to the route in a); c) displays the longest common sequence solution example, where the solution is found by solving the

longest increasing subsequence (LIS) (Schensted, 1961; Yang et al., 2005; Chan et al., 2007) for the index line, identified in b).

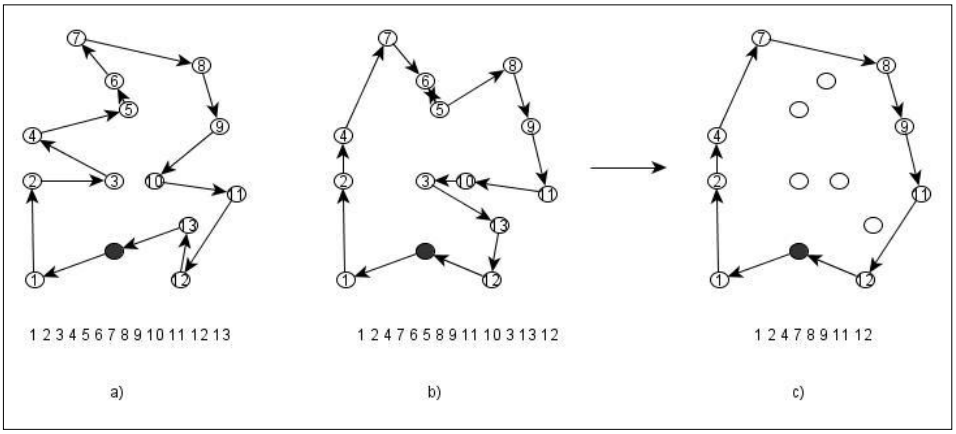

Fig. 5. The longest common sequence between two routes

The longest increasing sequence presented in c) is not the only one, there exist different increasing sequences that have the same length as in c). All the possible longest increasing sequences are as follows:

1 2 4 5 8 9 10 12
1 2 4 5 8 9 10 13
1 2 4 5 8 9 11 12
1 2 4 5 8 9 11 13
1 2 4 6 8 9 10 12
1 2 4 6 8 9 10 13
1 2 4 6 8 9 11 12
1 2 4 6 8 9 11 13
1 2 4 7 8 9 10 12
1 2 4 7 8 9 10 13
1 2 4 7 8 9 11 12
1 2 4 7 8 9 11 13

For LCSX, all the longest common sequences are identified and a single sequence is chosen randomly as the longest common sequence for the offspring. In the large neighbourhood search the parts of solution are destroyed by evaluating a single solution. In the LCSX crossover, some parts of solution are destroyed by evaluating the selected solution and another solution taken from the population.

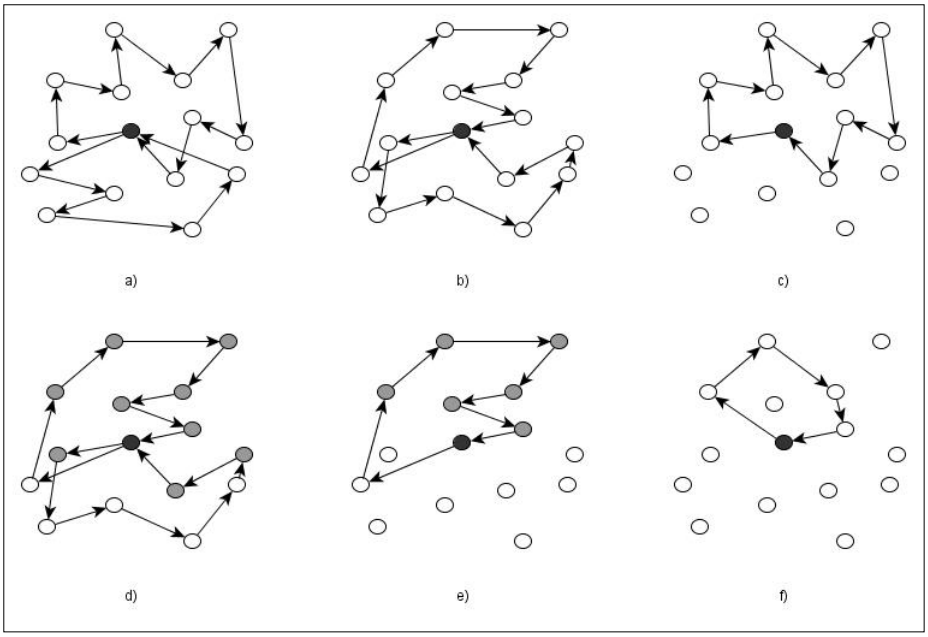

Fig. 6. Identification of the longest common sequence in all routes: a) and b) two parent solutions; c) the selected route from the first individual for evaluation; d) the routes in the second individual are identified that handle the same tasks as in the c) selected route; e) the route with the largest number of tasks is selected from the routes in d); f) the longest common sequence between the routes from c) and e) is identified

In Figure 6, an example of finding common sequences among more than one route in a solution is presented. For each route r_i in the first individual, the routes in the second individual are identified that handle at least one task belonging to r_i (Figure 6, d)). Then the route with the largest number of tasks is selected (Figure 6, e)) and the longest common sequence between routes in c) and e) is found. In e), the removed routes could also be evaluated for the longest common sequence, however, search for the longest common sequence in all of the routes increases the complexity of the crossover. To avoid extra complexity, the longest increasing sequence is identified for the routes with the largest number of common tasks. The same method is applied to all the routes in the first individual to get the intermediate solution x'_{o1} (Figure 7):

1. $x'_{o1} = \varnothing$;
2. $T_{temp} = \varnothing$;
3. for $\forall r_i \in x_{p1}$;
4. find the intersecting route $r_{int} \in x_{p2}$ that has the largest number of common tasks;
5. SEQ = find the longest common sequence between the routes r_i and r_{int};
6. for $\forall\ t \in r_i$, if t exist in the sequence SEQ, add the task t into x'_{o1}, otherwise, add t to T_{temp};
7. create x_{o1} by inserting $\forall t \in T_{emp}$ into x'_{o1} by evaluating the insertion cost function.

The CAX crossover preserves the sequence only for each the two subsequent tasks in the route, where the LCSX crossover preserves the longest common sequence between

two solutions. In both crossover operators, if the route has one task, it is removed and the task is added to the list of unassigned tasks. The complexity of LCSX is $O(p^2)$ in the worst case including computation of the longest common sequence.

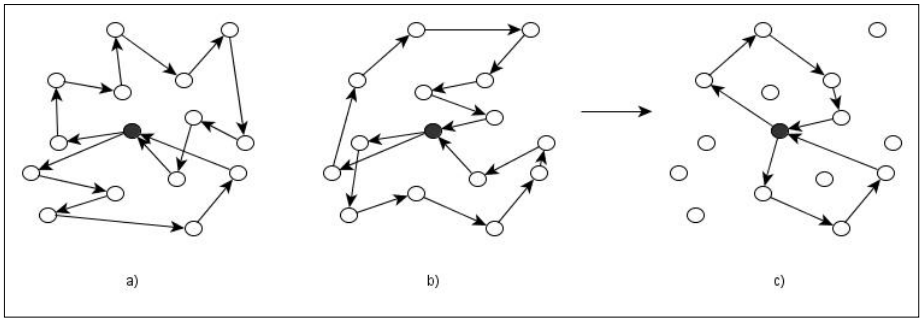

Fig. 7. The longest common sequence crossover: a) and b) two parent solutions; c) the intermediate solution found by the crossover

Insertion. A random insertion heuristic is chosen for reconstruction to preserve the stochastic approach of the genetic algorithm. A task is chosen from the list of unassigned tasks randomly and inserted into the route by evaluating the feasibility and minimising the insertion cost functions $f_v(a_i, t_m)$ and $f_d(a_i, t_m)$ defined in the previous section. If the crossover operators search for the common parts in the solutions, the random task insertion involves a diversification in a population.

The proposed crossover operators preserve common parts in all routes. In most cases, the crossovers will create an intermediate offspring that has the same number of routes as in the parent solution. To minimize the number of routes, as it is defined in the objective, the number of routes should be reduced in the intermediate offspring by removing a randomly selected route. The last step of each proposed crossover is processed as follows:

1. while $f_v(x'_{o1}) > f_v(x_p) - \delta$, remove the randomly selected route from x'_{o1} and insert all the tasks from the route to T_{temp};
2. create x_{o1} by inserting $\forall t \in T_{emp}$ into x'_{o1} by evaluating the cost functions $f'_v(a_i,t_m)$ and $f_d(a_i,t_m)$

In step 1), the value δ can have values from 0 to $f_v(x_p)$-1, where x_p is a parent solution participating in the crossover operation. If $\delta = 0$, the task will be inserted in the place with the minimal detour, however, the crossover will not try to minimize the number of routes. For the proposed crossover operators, the insertion function is used where δ is a random value from the set {0,1}, if $f_v(x_p)>1$, and $\delta = 0$ otherwise.

In the next section, the new crossover operators are compared to other crossover operators, described in this paper.

5. Experimental evaluation

The proposed crossover operators are implemented with the Java programming language in order to compare it with other crossover operators. The other crossover operators, described in the paper, are also implemented for comparison. For these crossover operators that use insertion heuristics, construction of solutions is the same as in the

algorithm definition. For comparison of crossovers, other parts of the genetic algorithm are common:

- Population initialization is performed by randomly selecting tasks for insertion and inserting them by evaluating the feasibility and minimising the cost. The same population is used for a single experiment with all the crossover operators. The population size is equal to 100.
- For evolution strategy, the *k-tournament selection* of the size k = 2 is chosen. In each iteration 10 new offspring are created.
- Mutation. To identify the features of a crossover better two groups of experiments are carried out. The first group of experiments does not involve the mutation operator. In the second group of experiments, the mutation operator is applied. The mutation operator used extracts $0.5z$ tasks from the solution and reconstructs the solution by the same reconstruction method that is used in the crossover, z is the random value within the range (0,1). Mutation is applied with the probability 0.15.
- Computation is stopped, when the best solutions are not improved for 300 iterations or when the maximum computation time (5 minutes) is reached.

The experiments are carried out using the well-known Solomon instances of the VRPTW problem (Solomon, 1987), where all the instances have 100 customers, distributed over the geographical area. The Solomon problem consists of 6 different problem sets R1, R2, C1, C2, RC1, RC2. The nodes in the sets R1 and R2 are randomly distributed over the geographical area; in the sets C1, C2, they are located in geographical clusters; in the sets RC1, RC2, some nodes are located in clusters and the other ones are distributed randomly. The sets R1, C1, RC1 define the problems with a large vehicle capacity and large time windows. The sets R2, C2, RC2 define the problems with a small vehicle capacity and narrow time windows.

For the evaluation of crossovers, we choose two problem instances from each set of problem. For each instance, the genetic algorithms with different crossovers are run 10 times, and each time, a new initial population is created**.** Computations are performed on a laptop PC with Intel Core 2 Duo 2.2 GHz CPU and 4GB RAM. Tables 1-2 summarize the results, where computations are performed without applying the mutation operator, and Tables 3-4 summarize the results, where computations involve the mutation operator. In Tables 1 and 3, the best results identified are presented, and, in Tables 2 and 4, the averaged results for each crossover operators are presented. The results in the tables show the difference from the best known solutions, reported in the papers (Solomon, 1987; Potvin and Bengio, 1996; Tan et al., 2001; Jung and Moon, 2002; Ombuki et al., 2002; Berger and Barkaoui, 2004; Alvarenga et al., 2005; Ombuki et al., 2006; Tan et al., 2006; Garcia-Najera and Bullinaria, 2011). The results are displayed in the form "*difference of the total path length / difference of the route number*" Problem instances used in the experiments and the best known solutions are as follows:

- C104 – vehicles 10, total path length 824.78;
- C106 – vehicles 10, total path length 828.94;
- C204 – vehicles 3, total path length 590.6;
- C207 – vehicles 3, total path length 588.29;
- R101 – vehicles 19, total path length 1645.79;
- R105 – vehicles 14, total path length 1377.11;
- R205 – vehicles 3, total path length 994.42;

- R209 – vehicles 3, total path length 909.16;
- RC101 – vehicles 14, total path length 1696.94;
- RC107 – vehicles 11, total path length 1230.48;
- RC201 – vehicles 4, total path length 1406.91;
- RC208 – vehicles 3, total path length 828.14;

Table 1. The difference between the best results, found by using crossover operators without applying mutation, and the best known solutions

	BCRC	RBX	LRX	CAX (proposed)	LCSX (proposed)
C104	22.27/0	22.98/0	907.52/0	**0/0**	**0/0**
C106	**0/0**	**0/0**	886.69/2	**0/0**	**0/0**
C204	**0/0**	10.61/0	708.94/1	**0/0**	**0/0**
C207	**0/0**	8.49/0	696.19/1	**0/0**	**0/0**
R101	46.0/0	46.3/0	186.5/0	*17.32/0*	**7.74/0**
R105	30.35/1	58.77/1	286.72/1	*10.38/0*	**0/0**
R205	*46.04/0*	82.61/0	698.16/1	73.010/0	**27.32/0**
R209	*34.35/0*	61.04/3	485.63/1	41.92/0	**19.43/0**
RC101	-8.5/1	30.55/1	125.31/2	*-52.68/1*	**-53.07/1**
RC107	*62.43/0*	54.62/1	438.83/2	268.73/1	**3.81/0**
RC201	22.61/0	34.94/0	724.57/1	*10.86/0*	**6.61/0**
RC208	39.38/0	70.4/0	922.42/1	*18.1/0*	**4.22/0**

Table 2. The difference between the averaged results, found by using crossover operators without applying mutation, and the best known solutions

	BCRC	RBX	LRX	CAX (proposed)	LCSX (proposed)
C104	61.88/0	61.03/0	1292.83/0	*20.16/0*	**8.98/0**
C106	47.53/0	91.65/0.1	1043.65/2.2	**0/0**	**0/0**
C204	1.69/0	27.96/0	860.58/3	**0/0**	**0/0**
C207	**0/0**	23.48/0	1078.35/1	**0/0**	**0/0**
R101	83.64/0.3	83.56/0.5	194.31/0.5	*37.58/0.3*	**15.93/0**
R105	73.68/1	134.69/1.1	322.18/1.9	*77.18/0.3*	**1.94/0.6**
R205	*80.76/0*	122.82/0	722.36/1	119.46/0	**65.15/0**
R209	*58.99/0*	110.87/0	646.42/1	75.48/0	**44.18/0**
RC101	33.07/1.7	98.23/1.5	186.03/2.1	*4.4/1.1*	**-38.89/1.3**
RC107	*71.15/0.8*	1253.52/1	488.63/2.9	303.1/1.1	**25.9/0.5**
RC201	*76.01/0*	128.65/0	894.62/1.1	93.9/0	**30.99/0**
RC208	*59.51/0*	98.21/0	982.84/1	63.43/0	**17.69/3**
Averaged CPU time, s	32.82	94.84	7.37	82.99	35.75

Table 3. The difference between the best results, found by using crossover operators and mutation also being applied, and the best known solutions

	BCRC	RBX	LRX	CAX (proposed)	LCSX (proposed)
C104	**0/0**	15.17/10	94.75/0	0/0	**0/0**
C106	**0/0**	**0/0**	**0/0**	**0/0**	**0/0**
C204	**0/0**	**0/0**	0.57/0	**0/0**	**0/0**
C207	**0/0**	**0/0**	**0/0**	**0/0**	**0/0**
R101	8.51/0	8.97/0	36.58/0	*6.78/0*	**5.22/0**
R105	0.66/1	106.17/0	56.61/0	*10.38/0*	**0/0**
R205	45.69/0	38.91/0	99.77/0	**28.97/0**	*38.83/0*
R209	25.19/0	44.51/0	62.35/0	*13.01/0*	**6.0/0**
RC101	–40.5/1	–24.87/1	-14.91/1	*-40.62/1*	**-53.95/1**
RC107	28.92/0	32.35/1	109.31/0	*5.53/0*	**3.53/0**
RC201	22.83/0	56.88/0	99.72/0	**6.61/0**	*8.09/0*
RC208	32.49/0	45.69/0	128.38/0	*17.47/0*	**0.87.01/0**

Table 4. The difference between the averaged results, found by using crossover operators and mutation also being applied, and the best known solution

	BCRC	RBX	LRX	CAX (proposed)	LCSX (proposed)
C104	31.95/0	61.83/0	313.5/0	*0.19/0*	**0/0**
C106	29.71/0	34.78/0	3.81/0	**0/0**	**0/0**
C204	0.74/0	5.69/0	38.79/0.3	**0/0**	**0/0**
C207	**0/0**	**0/0**	56.23/0.3	**0/0**	**0/0**
R101	33.29/0.1	33.06/0.3	76.54/0.3	*14.47/0*	**14.34/0**
R105	41.33/1	62.68/0.9	83.43/0.5	*23.59/0.1*	**5.18/0**
R205	82.19/0	78.21/0	189.15/0.6	**56.66/0**	*66.98/0*
R209	52.97/0	79.53/0	192.97/0.7	*49.47/0*	**24.33/0**
RC101	-2.48/1.4	21.69/15.4	26.81/1.3	*-39.06/1*	**-41.44/1**
RC107	67.39/0.7	96.1/1.1	122.5/0.8	*78.58/0.3*	**36.39/0.3**
RC201	69.73/0	82.74/0	147.17/0.7	*33.53/0*	**18.95/0**
RC208	50.89/0	70.6/0	167.23/0.3	*44.25/0*	**23.48/0**
Averaged CPU time, s	39.78	150.79	110.44	92.74	43.29

The results are compared according to the defined objective: at first, the route number differences are compared, and afterwards differences of the total path length are compared. The best values are bold in Tables 1-4, and the second best values are displayed in italics. The results show that the LCSX crossover, proposed in this paper, has found better solutions than the other crossover operators in 45 cases out of 48. In the experiments where mutation was not applied, LCSX has found better solutions in all the

cases. In 3 cases out of 48, the solutions found by LCSX are the second better comparing to the other crossover operators. For the RC101 problem instance, the path length difference identified is negative, however, the route number difference is positive. It means that a better path length is identified for the RC101 problem, but the number of routes was not minimized to the best known number. The best CPU time in the cases, where mutation was not applied, belongs to crossover LRX, however, the solutions found by this crossover are worst. The computation with the LRX crossover stopped early without finding better solutions, so it leads to a short CPU time and not so good solutions found. The LRX crossover showed better characteristics when mutation was applied, but the results are still worst comparing to other crossovers.

The best computation time in the cases, where mutation is applied, belongs to BCRC. The computation time of LCSX is the second best one and is quite similar to the computation time of BCRC. However, the solutions found by LCSX are better than that found by BCRC. The genetic algorithm with CAX has found the second best solutions in most cases, where mutation was applied, and in some cases, where mutation was not applied. The CPU time of CAX is longer than LCSX because removal of the common arcs at the beginning produces more unhandled tasks and requires more insertion trials while searching for a solution that could be competitive in the population. Although the CPU time of BCRC is shorter than LCSX and CAX, BCRC cannot be applied to the problems, where the solution is one route.

It is worth mentioning, that crossovers, defined by other authors (BCRC, RBX, LRX), can be dependent on other parts in the genetic algorithm, i.e. on the created initial population or the selection operator, etc. LCSX has showed better results when computing without mutation or with mutation that randomly removes and reinserts tasks.

6. Conclusions

In the paper, new crossover operators are proposed, based on the common part search between parent solutions and the insertion heuristic is used for reconstruction. The crossover operators are proposed that intensify the search by identifying the common sequence of tasks in parent solutions. All unhandled tasks are reinserted back by using random insertion heuristic to preserve stochastic characteristics of genetic algorithm and to involve a diversification in the population. The new crossover operators are compared to other crossover operators that also deal with insertion heuristics for constructing feasible solutions. The crossover operators proposed are applied to VPRTW problem instances for comparison. The experimental evaluation shows that the new crossover operators, in most cases, find better solutions than other crossover operators. The computation time of the new crossover operator LCSX is similar to that of other crossovers, however, the solutions found are more accurate as compared to that found by the other crossovers.

The solutions are found in the experimental evaluation by applying the mutation operator that randomly removes parts of the solutions and reconstructs the solution. However, such a mutation operator was chosen just for comparison of crossover operators. Different mutation operators could be used to find better solutions. Also, it is worth mentioning that some solutions are equal to the best known solutions even in the cases, where mutation was not applied, however, no additional improvement approaches are used. The new crossover operators proposed do not depend on the number of routes and they are designed for problems where not only the number of routes is the objective.

The genetic algorithm together with the new crossover operators can be applied to different VRP problems or other problems, that can be expressed as a graph, and depend on the sequence of elements.

The proposed crossovers could be investigated further in genetic algorithms with different ranking approaches, i.e. multi-objective genetic algorithms, where Pareto ranking is used. Different ranking would also affect the reconstruction method where the delta value is taken into account.

References

Alvarenga, G. B., de A. Silva, R. M., Sampaio, R. M. (2005). A Hybrid Algorithm for the Vehicle Routing Problem with Time Window, INFOCOMP Journal of Computer Science, 4(2), 9–16.

Berger, J., Barkaoui, M. (2004). A parallel hybrid genetic algorithm for the vehicle routing problem with time windows. Computers & Operations Research, 31, 2037–2053..

Chan, W.., Zhang, Y., Fung, S. , Ye, D., Zhu, H. (2007). Efficient algorithms for finding a longest common increasing subsequence. J. Comb. Optim., 13(3), 277–288.

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

Deep, K., Adane, H. M. (2011), New Variations of Order Crossover for Travelling Salesman Problem. IJCOPI 2(1) , 2–13.

Garcia-Najera, A., Bullinaria, J. A. (2011). An improved multi-objective evolutionary algorithm for the vehicle routing problem with time windows. Computers & OR, 38, 287–300.

Holland, J. (1975). Adaptation in Natural and Artificial Systems: An Introductory Analysis with applications to biology, Control and Artificial Intelligence. The University of Michigan Press.

Hong, T., Wang, H., Lin, W., Lee, W. (2002). Evolution of Appropriate Crossover and Mutation Operators in a Genetic Process. Appl. Intell., 16(1), 7–17.

Jih, W., Chen, Y., Hsu, Y. (1996). A Comparative Study of Genetic Algorithms for Vehicle Routing with Time Constraints. Proceedings of the 1996 International Computer Symposium, 17–24.

Jih, W., Hsu, Y. (2004). A family competition genetic algorithm for the pickup and delivery problems with time window. Bull. Coll. Eng. N.T.U. 90, 89–98.

Jung, S., Moon, B. R. (2002). A Hybrid Genetic Algorithm For The Vehicle Routing Problem With Time Windows. GECCO 2002, 1309–1316.

Kumar, N., Karambir, Kumar, R. (2012). A Comparative Analysis of PMX, CX and OX Crossover operators for solving Travelling Salesman Problem. International Journal of Latest Research in Science and Technology, 1(2), 98–101.

Lukasiewycz, M., Glaß, M., Teich, J. (2008). A Feasibility-Preserving Crossover and Mutation Operator for Constrained Combinatorial Problems. Volume 5199 of Lecture Notes in Computer Science, Springer, 919–928.

Misevičius, A., Kilda, B. (2005). Comparison of crossover operators for the quadratic assignment problem. Information Technology and Control, 34(2), 109–119.

Nagata, Y., Bräysy, O. (2009). Edge assembly-based memetic algorithm for the capacitated vehicle routing problem. Networks, 54(4), 205–215.

Ombuki, B. M., Nakamura, M., Maeda, O. (2002). A hybrid search based on genetic algorithms and tabu search for vehicle routing. In 6th IASTED Intl. Conf. On Artificial Intelligence and Soft Computing (ASC 2002), edited by A.B. Banff, H Leung, ACTA Press, 176–181.

Ombuki, B. M., Ross, B., Hanshar, F. (2006). Multi-Objective Genetic Algorithms for Vehicle Routing Problem with Time Windows. Applied Intelligence, 24(1), 17–30.

Pisinger, D., Ropke, S. (2009). Large neighborhood search. Handbook of Metaheuristics, 2nd edition, M. Gendreau and J.-Y. Potvin(eds).

Potvin, J.-Y., Bengio, S. (1996). The Vehicle Routing Problem with Time Windows Part II: Genetic Search. INFORMS Journal on Computing, 8(2), 165–172.

Reid, D. J. (2000). Feasibility and Genetic Algorithms: the Behaviour of Crossover and Mutation. Land Operations Division of DSTO Electronics and Surveillance Research Laboratory.

Schensted, C. (1961), Longest increasing and decreasing subsequences. Canad. J. Math. 13, 179–191.

Solomon, M.M. (1987). Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. Operations Research, 35(2), 254–265.

Srinivas, M., Patnaik, L. M. (1994). Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics, 24(4), 656–667.

Tan, K. C., Chew, Y. H., Lee, L. H. (2006). A Hybrid Multiobjective Evolutionary Algorithm for Solving Vehicle Routing Problem with Time Windows. Comput. Optim. and Appl. 34 (1), 115–151.

Tan, K. C., Lee, L. H., Zhu, K. Q., Ou, K. (2001). Heuristic methods for vehicle routing problem with time windows. AI in Engineering, 15(3), 281–295.

Thangiah, S., Nygard, K., Juell, P. (1991). GIDEON: A genetic algorithm system for vehicle routing with time windows. In Seventh Conference on Artificial Intelligence Applications, 322–328.

Vaira, G., Kurasova, O. (2010). Modified bidirectional shortest path Dijkstra's algorithm based on the parallel computation. Proceedings of the Ninth International Baltic Conference – Baltic DB&IS 2010 (J. Barzdins, M. Kirikova (eds.)), University of Latvia Press, 205–217.

Vaira, G., Kurasova, O. (2011). Parallel Bidirectional Dijkstra's Shortest Path Algorithm. DB&IS, Volume 224 of Frontiers in Artificial Intelligence and Applications, IOS Press, 422–435.

Yang, I, Huang, C., Chao, K. (2005). A fast algorithm for computing a longest common increasing subsequence. Inf. Process. Lett., 93(5), 249–253.

Yeun, L. C., Ismail, W. R., Omar, K. and Zirour, M. (2008). Vehicle Routing Problem: Models and Solutions. Journal of Quality Measurement and Analysis (JQMA), 4(1), 205–218.

Received November 24, 2013, accepted December 10, 2013