INFORMATICA, 2014, Vol. 25, No. 1, 155–184
© 2014 *Vilnius University*
DOI: http://dx.doi.org/10.15388/Informatica.2014.09

Genetic Algorithm for VRP with Constraints Based on Feasible Insertion

Gintaras VAIRA*, Olga KURASOVA
Institute of Mathematics and Informatics, Vilnius University
Akademijos 4, LT-08663 Vilnius, Lithuania
e-mail: gintaras@vaira.net, olga.kurasova@mii.vu.lt

Received: January 2013; accepted: March 2013

Abstract. In the paper we propose a genetic algorithm based on insertion heuristics for the vehicle routing problem with constraints. A random insertion heuristic is used to construct initial solutions and to reconstruct the existing ones. The location where a randomly chosen node will be inserted is selected by calculating an objective function. The process of random insertion preserves stochastic characteristics of the genetic algorithm and preserves feasibility of generated individuals. The defined crossover and mutation operators incorporate random insertion heuristics, analyse individuals and select which parts should be reinserted. Additionally, the second population is used in the mutation process. The second population increases the probability that the solution, obtained in the mutation process, will survive in the first population and increase the probability to find the global optimum. The result comparison shows that the solutions, found by the proposed algorithm, are similar to the optimal solutions obtained by other genetic algorithms. However, in most cases the proposed algorithm finds the solution in a shorter time and it makes this algorithm competitive with others.
Key words: vehicle routing problem, constraints, heuristics, genetic algorithms, feasibility.

1. Introduction

The vehicle routing problem (VRP) is a well known combinatorial problem that attracts researchers to investigate it applying the existing and newly created optimization algorithms. Traditionally, VRP is defined as a routing problem with a single depot, a set of customers, multiple vehicles and the objective in order to minimize the total cost while servicing every customer. A set of constraints can be defined for the VRP problem. In literature we can find different kinds of VRP problems that are grouped according to specific constraints. The well known VRP constrained problems are: VRP with capacity limitations (CVRP), where vehicles are limited by the carrying capacity; VRP with time windows (VRPTW), where a customer can be serviced within a defined time frame or time frames; VRP with multiple depots (MDVRP), where goods can be delivered to a customer from a set of depots; VRP with pick-up and delivery (VRPPD), where rules are defined to visit pick-

*Corresponding author.

up places and later to deliver goods to the drop-off location. Many researches on different heuristic approaches can be found for the solution of the above mentioned problems (Yeun *et al.*, 2008).

In this research we deal with one of metaheuristic algorithms – genetic algorithms. Having constraints in the problem definition, the aim is to find the solution that does not violate any constraint. Such a solution is called a feasible solution or feasible individual. Due to a stochastic characteristic, genetic algorithms generate solutions in the whole search space including the infeasible space. For a constrained problem, the feasible search space is smaller than the whole search space. The search for the feasible solution in a stochastic way can continue very long until the acceptable solution has been found. The common genetic algorithm approaches involve additional repair and improvement methods that are designed for a specific constraint to keep the generated solutions in the feasible search space.

In order to generate solutions in the feasible search space, we propose a genetic algorithm (GA) that is based on insertion heuristics. The random insertion heuristic is considered to preserve a stochastic characteristic of the genetic algorithm and to generate solutions in the feasible space by checking compliance to the defined constraints in the insertion process. Infeasibility is still allowed in the proposed algorithm, because the random insertion approach can create infeasible initial solutions in a highly constrained problem. The defined GA individual includes feasible partial routes and a set of customers that have not been serviced due to constraint violations. In literature we can find the usage of insertion heuristics in the initialisation phase of GA as it belongs to a group of solution construction algorithms. The novelty of the proposed approach is the usage of insertion heuristics in crossover and mutation operators. Differently from other genetic algorithms, the proposed algorithm operators do not construct offspring directly. However, evaluating the information from previous generation, the parts of solutions that should be preserved for the next generation and the weak parts that should be reconstructed are identified in the proposed operators. The crossover and mutation operators are defined to identify such weak parts of the solution. New crossover operators are defined to intersect two solutions in the defined ways and extract a subset of customers. The extracted parts are reinserted back using the defined insertion. The second population is used in the mutation process where the usage of the second population increases the probability that the solution, obtained in the mutation process, will survive in the first population and increases the probability to find the global optimum. In contrast to other approaches, the proposed algorithm does not involve additional local search methods to improve the solution; therefore it does not depend on the local search limitations and can be easily extended with additional constraints.

The rest of the paper is organized as follows. Section 2 describes a VRP problem and constraints, reviews genetic algorithms and the common feasibility handling approaches. Section 3 presents an overview of insertion heuristics and their usage in genetic algorithms. The new approach is introduced and described in Section 4, while Section 5 presents experimental results. The last section concludes the paper.

2. Genetic Algorithms for Solving VRP

The vehicle routing problem has got much attention in recent years. Due to usefulness in real life and innovation in the transportation sector as well as logistics, VRP continues to draw researchers' attention. A number of different exact and heuristic methods have been studied to solve the VRP problem that is known to be NP-hard. Although exact methods give the optimal solution, their computation time considerably increases with the increasing size of the problem. In Dzemyda and Sakalauskas (2011) we can find a survey of heuristic methods for solving problems that are known to be NP-hard. Local searches and heuristic approaches often produce a near optimal solution within a reasonable computation time. These methods may be sensitive to data sets given or require additional training data during the learning process. Metaheuristic is another approach for solving a complex problem that may be too difficult or time-consuming by traditional techniques. In this research we deal with one of metaheuristic approaches – genetic algorithms that have been successfully applied to solve many combinatorial problems. The standard genetic algorithm has limitations in the constrained environment. However, it is able to incorporate other techniques within its framework to produce a hybrid that provides better efficiency (Yeun *et al.*, 2008). In this research the focus is given only to genetic algorithm approaches for solving VRP with constraints.

2.1. *VRP and Constraints*

Traditionally a vehicle routing problem is described as the undirected graph $G = (N, E)$, where $N = \{n_0, n_1, \dots, n_k\}$ is a set of nodes, $d = \{n_0\}$ is a depot node, $N \backslash d$ is a subset of nodes that represent k customers, E is a set of arcs, and $V = \{v_1, \dots, v_t\}$ is a set of vehicles. Usually G is treated as a complete graph, in this case, the set $E = \{e_{ij}\}$, where $i \neq j$, $0 \leqslant i \leqslant k$, $0 \leqslant j \leqslant k$. The objective of VRP is to minimize the total number of required vehicles and the total travelling distance (Solomon, 1987; Berger *et al.*, 1998; Tan *et al.*, 2001a, 2001b; Jung and Moon, 2002; Zhu, 2003; Berger and Barkaoui, 2004; Alvarenga *et al.*, 2005; Ombuki *et al.*, 2006; Tan *et al.*, 2006; Hasle and Kloster, 2007; Yeun *et al.*, 2008; Garcia-Najera and Bullinaria, 2011; Vaira and Kurasova, 2013).

In the VRP problem the Euclidean distance between two nodes in the graph is usually treated as the shortest path value. However, when dealing with real life data, the Euclidean distance between two nodes adds additional inaccuracy to the results. Path calculation can be related to graph data, travelling time or vehicle parameters. Detailed reviews on the existing shortest path algorithms are described in Vaira and Kurasova (2010, 2011).

Various constraints can be added to the VRP. The defined constraints usually refer to real life situations. Let us define a single constraint $c \in C$, where C is a set of all constraints that should not be violated in the final solution.

The most known constraints on the VRP problem are capacity constraints and time window constraints. The capacity constraints $C_c \subseteq C$ are carriage limitations applied to each vehicle. In literature a capacitated vehicle routing problem (CVRP) is defined with equal capacities for all vehicles. However, in real life vehicle fleet with different capacities can be used to solve the delivery problem.

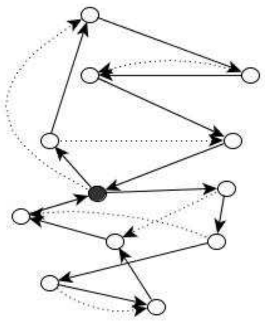

Fig. 1. Pick-up and delivery problem.

Time window constraints $C_{tw} \subseteq C$ define time frames when a customer can be serviced. The problem dealing with time windows constraints is called VRPTW. Single-sided and double-sided windows are specified in terms of time frames that are widely considered in literature. However, real life situations can give a multiple time frame representation, where a customer can be serviced in one of the defined time frames:

1. $[t_1,\ \infty)$ defines a time frame constraint when a vehicle has to arrive no earlier than the time t_1 If a vehicle comes too early, it has to wait until time t_1.
2. $(\infty, t_2]$ defines a time frame constraint when a vehicle has to arrive to a customer no later than the time t_2.
3. $[t_1, t_2]$ defines a double sided time frame constraint, where $t_1 \leqslant t_2$. The constraint includes the limitation from both previously defined constraints.
4. $(\infty, t_1] \cup [t_2, t_3] \cup [t_4, \infty)$ defines multiple time frames, where $t_{i-1} \leqslant t_i$. The multiple time frame constraints can include any of previously defined time window constraint. However, the single constraint from the group needs to be satisfied.

The time window constraint can be added to the depot node to define the overall travelling time limit for a single vehicle. The maximum number of vehicles can be treated as an additional constraint $c_v \in C$, where c_v defines the limit of vehicles in the solution.

Real time situations can give another type of constraints where goods need not only to be brought from a depot to a customer, but also to be picked up from a number of customers and brought back to depot or to any other customer. This problem is known as a pick-up and delivery problem (VRPPD). The set $C_{pd} \subseteq C$ defines pick-up and delivery constraints within the problem, where each $c \in C_{pd}$ is a constraint that defines the delivery of a certain amount of goods from the starting node n_s to the target node n_t. In Fig. 1 the filled circle represents a depot, the empty circles represent customers, the dotted lines represent possible pick-up and delivery constraints, and the solid lines represent a possible routing solution for two vehicles.

Let us define the function $F_c(x)$ that evaluates violation of constraints in the solution and $f_c(x)$ that evaluates violation of the single constraint $c \in C$:

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

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

The objective of the traditional VRP is to find a solution x that minimizes the functions $f_v(x)$ and $f_d(x)$ in the defined order and satisfies the equation $F_c(x) = 0$:

$$f_v(x) = \bigl|\{r_1, \ldots, r_s\}\bigr|, \qquad f_d(x) = \sum_{j=1}^{s} d_l(r_j),$$

$$r_j = (n_{j_1}, \ldots, n_{j_m}), \qquad n_{j_i} \neq n_0, \quad r_j \text{ is a single route with } m \text{ nodes,}$$

$$x = \{r_1, \ldots, r_s\}, \quad s \leqslant c_v, \qquad N_j = \{\forall n_{j_i} \in r_j\}, \qquad |N_j| = m,$$

$$N_1 \cap \cdots \cap N_s = \emptyset, \qquad \{n_0\} \cup N_1 \cup \cdots \cup N_s = N,$$

$$d_l(r_j) = l(n_0, n_{j_1}) + \sum_{i=2}^{m} l(n_{j_{i-1}}, n_{j_i}) + l(n_{j_m}, n_0),$$

$$l(n_{j_{i-1}}, n_{j_i}) \text{ is a distance between nodes}$$

2.2. *Genetic Algorithm and Feasibility Handling*

2.2.1. *Main Genetic Algorithm Principles*

Genetic algorithm approaches are based on the concept of natural selections and genetics. A single solution within a genetic algorithm is described as an individual. In the iterative process new offspring are created from the previous generation by applying stochastic transition operations. "Only the fittest survive" is the main principle of the genetic algorithm. One of the main reasons that distinguishes this approach from sequential local search methods is the maintenance of a large population of candidate solutions instead of producing a sequence of single solutions deterministically (Reid, 2000). The main steps of the genetic algorithm are initialisation, fitness function evaluation, selection of individuals for a new generation, crossover and mutation. The iterative process is performed until one of the stop criteria is met. The stop criterion can be either the maximum time defined for calculation or maximum number of iterations without improvement of the current best solution found (Reid, 2000; Hong *et al.*, 2002; Jung and Moon, 2002; Yeniay, 2005).

The classical genetic algorithm paradigm operates with the encoded form of solutions called chromosomes. A single chromosome describes a solution $x \in S$, where S is the whole search space. Chromosomes usually have the form of a string in an appropriate alphabet (Reid, 2000). The encoded chromosomes do not exploit any useful information about a particular problem domain. However, the standard genetic operators that operate on encoded lines can be applied to such an approach. In Blanton and Wainwright (1993) a possible expression of the VRPTW solution is described where it is expressed as a sequence of customers in the chromosome in the way they are visited. In Potvin and Bengio (1996) the author has proposed a genetic algorithm that applies genetic operators to solutions directly, thus avoiding coding issues.

The fitness function $f(x)$ evaluationallows us to identify the value of each individual. The individual with the best fitness value is treated as the best currently known solution in the population. The selection operator identifies population members that will be used in the reproduction for generating a new offspring by examining the fitness value of each individual in the population.

The crossover operator generates a new offspring from two chosen individuals. Thus, the offspring inherits some characteristics from each parent. It is intended to construct a new offspring by combining the fragments of old solutions that previously have shown good characteristics (Reid, 2000). The traditional crossover operators, i.e. a partially mapped crossover or two-point crossover, can be used when the problem is encoded as a line of characters. In Blanton and Wainwright (1993) two merge crossovers are proposed for the encoded VRP problem, where the new crossover operators are superior to the traditional ones.

The mutation operator generates a new offspring from a single chosen individual. Mutation should increase the probability to find the global optimum instead of the local one. The mutation and crossover operators are applied with the defined probabilities that can vary depending on a problem. A number of guidelines can be found for mutation and crossover probabilities, as well as the studies to define optimal probabilities (Hong *et al.*, 2002). Some researches can be found on the adaptive probabilities for crossover and mutation operators. In Zhang *et al.* (2004, 2007), a fuzzy logic is considered for adjusting the probabilities. Initially the evolutionary algorithms had only selection and mutation, while the genetic algorithms also utilize the crossover operator (Reid, 2000). Both operators play an important role in genetic algorithms due to the success of recombination of the existing solutions into a new one.

2.2.2. *Constraint Handling*

When dealing with constraints, a stochastic approach to find optimal solutions can compute very long, until an acceptable solution has been found (Reid, 2000). The search space that contains only feasible solutions can be defined as $S_F \subseteq S$, where each $x \in S_F$ does not violate any of the defined constraints. The search space that contains all other solutions is the infeasible search space $S_U = S \backslash S_F$. The solution x belongs to the feasible search space S_F, if $F_c(x) = 0$.

Highly constrained problems are those where the feasible search space is very small. Thus the probability to generate solutions in such a space for crossover and mutation operators can be adequately small (Reid, 2000). It is the waste of computation time when infeasible solutions are generated and later eliminated. The author in (Reid, 2000) discusses the possibility of having feasible solutions generated in crossover and mutation operations. In order to handle feasibility in the genetic algorithm, the creation of infeasible solutions is usually allowed, but they are additionally penalized with a modification of the objective function or correction with an additional repair algorithm (Reid, 2000; Yeniay, 2005; Lukasiewycz *et al.*, 2008a, 2008b).

The penalty function $p(x)$ transforms a constrained problem into an unconstrained one. There are two main ways of penalty function application:

- additive: $f_p(x) = f(x) + p(x)$, where $p(x) = 0$, if none of the constraints is violated, and $p(x) > 0$, otherwise;
- multiplicative: $f_p(x) = f(x)p(x)$, where $p(x) = 1$, if none of the constraints is violated, and $p(x) > 1$, otherwise (Yeniay, 2005).

Various penalty functions are considered on the basis of their application characteristics. Some of them can dramatically change the fitness value or completely remove from a population list. The death penalty has the penalty function $p(x) = +\infty$ for each $x \in S_U$. Although it will not allow to have infeasible solutions, it is expected to work well when the feasible search space is a reasonable part of the whole search space (Michalewicz, 1995a). However, for highly constrained problems the algorithm can suffer a degradation when trying to search for feasible solutions and if the feasible solution is found, the search may prevent to find a better one (Yeniay, 2005). For example, adaptive penalties update the parameters for each generation according to information gathered from the population. There are a number of different penalty approaches (Yeniay, 2005). Although penalty functions help to identify infeasible solutions and keep individuals with the best characteristics in the population, they affect the generation of feasible solutions only indirectly and still allow the generation of infeasible solutions. The Pareto fitness ranking scheme for evolutionary multiobjective optimization is proposed in Ombuki *et al.* (2006), Tan *et al.* (2006), Garcia-Najera and Bullinaria (2011) to assign the relative strength of individuals in the population. The ranking mechanism assigns the smallest rank to non-dominated individuals and the dominated individuals are ranked according to the individuals in the population and the defined criteria. In contrast to traditional genetic algorithms, Pareto ranking attempts to assign a single fitness score to a multiobjective problem. However, in literature there can be found Pareto ranking in the genetic algorithm treated as equivalent to the penalty approach (Michalewicz, 1995b). In Alvarenga *et al.* (2005) the author has proposed to use 10 hierarchical criteria to rank individuals in the population.

In paper of Michalewicz (1995a) the author discusses the advantages and disadvantages of having feasible and infeasible solutions in genetic algorithms and how they influence the results. The discussion is carried out on the issue how the feasible and infeasible solutions can be compared. In general, two evaluation functions $f_f(x)$, where $x \in S_F$, and $f_u(x)$, where $x \in S_U$, are considered. Different evaluation functions $f_f(x)$ and $f_u(x)$ are defined because of the ability to compare the solutions in two distinct search spaces. However, the relation between these two functions can be designed via the extended function $q(x)$, where $f_u(x) = f_f(x) + q(x)$, and $q(x)$ can be either the penalty function already discussed or the cost for repairing the solution. Good results are reported, when the penalty function is designed so that feasible results are always treated better than infeasible results. The repair can be designed in two different ways:

- An individual is repaired for evaluation only, where $f_u(x) = f_f(y)$, and y is a repaired (i.e. feasible) version of x. It is the so-called Lamarckian approach (Michalewicz, 1995a; Zhu, 2003; El-Mihoub *et al.*, 2006). The weakness of such an approach is that it depends on the problem and a specific repair algorithm has to be designed (El-Mihoub *et al.*, 2006).

- An individual is repaired and the previous individual is replaced by its repaired version. It is called a Baldwinian approach (Michalewicz, 1995a; Zhu, 2003; El-Mihoub *et al.*, 2006). This method has the same limitation as the previous one. The question of replacement is also widely considered. In some researches the fixed percent of the repaired individuals replace the previous one or this can be dependent on the problem or even on the evolution process.

In Jung and Moon (2002) the author has proposed to use 2D chromosomes for VRP encoding to handle additionally the position of nodes in the 2D Euclidean space. The described crossover operator uses a 2D partitioning to interchange routes between two chromosomes, where each route represents the travelling path of a single vehicle. However, the repair algorithm is considered to connect separate fragments of the route by taking into account additional decision variables. Local search methods such ascrossover, or-opt and relocation with additional modifications that check time constraints are used in the mutation operator to improve the solution. Local improvement methods or repair algorithms are very helpful for a single constrained optimization. However, identification of the parts for solution improvement can be quite complex because of constraints. A problem can arise when the improvement of one objective can lead to a degradation of others.

In Tan *et al.* (2006) the author uses individuals that are composed of a set of routes where each route contains a list of customers. A crossover is defined to exchange the routes between individuals. If the newly added route contains the customer that has already been visited in another route, the customer is removed from the previous one and left in a newly added route (Tan *et al.*, 2006). If individuals selected for crossover are feasible, the offspring, generated from parent individuals, will remain feasible. However, a set of transitions is proposed for feasibility handling in the mutation operator, where constraint violation is evaluated after each transition. If mutation transitions generate an infeasible solution, the original routes are restored (Tan *et al.*, 2006). Such an approach does not help generate feasible solutions, but it helps to avoid infeasibility.

In Alvarenga *et al.* (2005) the author has proposed a crossover where feasible routes from the parent individuals are inserted in the offspring. At first the routes with the maximum number of customers are inserted. After all feasible routes have been inserted in the offspring, the insertion of the remaining customers is tested in the existing routes. If some customers are still not included to any route, a new route is created and a stochastic push-forward insertion heuristic is used to insert customers (Alvarenga *et al.*, 2005).

In literature we can find approaches of using genetic algorithms in a two-phase approach, where in the first phase genetic algorithms are used to solve a single objective and in the second phase different algorithms are used to continue the optimization process (Berger and Barkaoui, 2004; Alvarenga *et al.*, 2005; Ombuki *et al.*, 2006). The fluctuating population size is also considered to keep infeasible solutions in the solution set. It is proposed because some parts of infeasible solutions can still remain significant for crossover and mutation operators (Reid, 2000).

The author in Berger and Barkaoui (2004) has proposed parallel two-population coevolution genetic algorithms, Pop1 and Pop2, for VRPTW. The first population, Pop1, has the objective to minimize the travel distance to a fixed number of vehicles. On the other hand, Pop2 works to minimize the violated time window in order to find at least one feasible individual. In Pop2 the vehicle number is limited to the number obtained by Pop1 minus one. Each time a feasible individual is found, the population Pop1 is substituted by Pop2 and the fixed number of vehicles considered in both populations is decreased by one.

Different mutation and crossover operators can produce different offspring and thus affect the performance of the genetic algorithm. Dynamic genetic algorithms are considered in Hong *et al.* (2002). Since the efficiency of different genetic operators can depend on different problems and also on different stages of the genetic algorithm, the proposed dynamic genetic algorithm is designed to choose different operators as well as to dynamically adjust their application probabilities.

In Reid (2000) the author discusses feasibility handling in a two-point crossover where a set of crossovers with different boundary indices is considered. Probability function is defined to find a feasible crossover for a linear constraint problem. However, for a highly constrained problem where a feasible space is very small as compared to the full search space, only a half-feasible crossover with a single boundary point is discussed. In order to handle feasibility in the mutation process, the proposed mutation operator is based on the crossover operator where the selected individual is crossed with a randomly generated individual (Reid, 2000).

Most of the feasibility handling approaches deal with the population control to preserve feasible individuals. The common approaches like penalty methods or repair algorithms can help rank individuals for the next generation by identifying the infeasible ones. However, the crossover and mutation operators are still organized to generate solutions in the whole search space. It is still time consuming to get an acceptable solution. In literature we can find approaches to define the feasibility preserving operators. Limitations still exist where the constraint violation is evaluated after each step and the original solution is restored in an unsuccessful case. Repair algorithms usually take into account a specific problem or specific constraints.

The aim of this research is to propose a genetic algorithm with operators designed to generate solutions in the feasible search space for the VRP problem with constraints.

3. Insertion Heuristics

Insertion heuristics are popular methods for solving a variety of vehicle routing and scheduling problems. Insertion heuristics were first introduced for a travelling salesman problem (TSP) and belong to a group of route construction algorithms (Rosenkrantz *et al.*, 1977; Campbell and Savelsbergh, 2004). The main principle of an insertion heuristic is to start from a single node that is usually called a seed node and that forms the initial route from depot. Other nodes are inserted one by one evaluating certain functions to a select a node and the place in the route for insertion. The well-known insertion heuristic approaches used in TSP are categorized by the methods used for the node selection to be inserted: random insertion, nearest insertion, farthest insertion and cheapest insertion.

For the farthest and nearest insertion each next node is selected for insertion according to the distance to the already constructed route where the functions for maximization and minimization are defined respectively. The node is inserted by evaluating the cost function $c(n_i, n_k, n_j) = l(n_i, n_k) + l(n_k, n_j) - l(n_i, n_j)$ where n_i, n_j are the nodes in the current constructed route, n_k is the node to be inserted, and $l(n_i, n_j)$ is the distance function. In the random insertion heuristic a node is randomly selected from a set of nodes that are still not included to any route. The place in the route where a randomly selected node has to be inserted is determined by minimizing the same cost function $c(n_i, n_k, n_j)$. In contrast to the random insertion heuristic the cheapest insertion heuristic selects the node for insertion by minimizing the defined function for all nodes and all places in the route.

Solomon (1987) has proposed three types of insertion heuristics. The most successful of them is called I1. The first route is initialised with the seed node which is the farthest one from the depot. Nodes are inserted into the first route until reaching the limit of capacity constraints. If still there are unrouted nodes, a new route is created and the insertion process is repeated until all the nodes are inserted. Two subsequently defined criteria $C_1(n_i, n_u, n_j)$ and $C_2(n_i, n_u, n_j)$ are used to select the node n_u for insertion between the nodes n_i and n_j. The first function determines detour and delay values. The second function generalizes a regret measure over all routes to estimate what could be lost later if the node is not immediately inserted in its best place. The criterion function C_1 depends on the coefficients $(\alpha_1, \alpha_2, \mu)$ and the overall insertion method efficiency depends on them (Potvin and Dubé, 1994). The author in Potvin and Rousseau (1993) has proposed a parallel version of insertion heuristic I1.

Insertion heuristics are popular because they are easy to implement and they show good characteristics in creating feasible solutions (Campbell and Savelsbergh, 2004). However, they still depend on the methods of selecting the nodes and the place in the route for insertion. In this paper the usage of the insertion heuristic together with the genetic algorithm approach, seeking for better efficiency, is considered. As already mentioned, the insertion heuristic is usually used in the initialisation of solutions in the genetic algorithm. In Potvin and Bengio (1996), Jung and Moon (2002), the authors have proposed the usage of Solomon insertion heuristics to create the initial population that is used in the genetic algorithm. The existence of adjustable weights in criteria functions of I1 allows to generate the initial set of different solutions. A similarity of insertion heuristics can be found in or-opt and relocation algorithms used in the mutation operation, proposed in the paper (Jung and Moon, 2002) where the constraint violation is evaluated for the nodes before inserting them in different parts of the solution. In the literature a random node insertion is also considered for creating the initial population for genetic algorithm (Tan *et al.*, 2006).

In Alvarenga *et al.* (2005) the author has proposed a genetic algorithm where stochastic push forward insertion heuristic (PFIH) is used to initialise the population as well as to insert unserviced customers in to route in the crossover operator. In the original push-forward insertion heuristic the first customer in each new route is defined deterministically. Customers then are chosen one by one minimizing the travel distance. The original PFIH is deterministic, but in the stochastic PFIH a random choice is used to define the first customer for each new route. Stochastic PFIH is used for initialisation to produce distinguished individuals for GA (Alvarenga *et al.*, 2005).

In Potvin and Dubé (1994) the genetic algorithm approach is defined to find the best values of coefficients (α_1, α_2, μ) for Solomon insertion heuristic I1. The coefficient values in the range [0, 1] are mapped to values [0, 127] and encoded in 7 symbol substrings as a binary expression and a single point crossover operator is used. The author argues that the results of insertion heuristics can be greatly improved by a careful search for coefficients.

The usage of insertion heuristics in the crossover operator of the genetic algorithm is proposed in Ombuki *et al.* (2006). In the first step a route is chosen randomly in the opposite solution as a reference. Each node that belongs to that route is extracted in the current solution and reinserted back by evaluating constraint violation. If a feasible insertion is not found, an additional route is added to the solution. The route selected from the opposite solution is used only to define which nodes should be extracted. However, such a crossover approach does not share any information between solutions.

From the overview of insertion heuristic usage in genetic algorithms for the VRP problem we can see that usually the insertion heuristics takes place in the initialisation step of GA to create the initial set of solutions. There are some approaches to use insertion heuristics in genetic algorithm operators, but the insertion heuristics used are still treated as the methods to support the main algorithm. The author in Campbell and Savelsbergh (2004) describes the benefits of insertion heuristic in handling constraints and in generating feasible solutions. In contrast to insertion heuristics, genetic algorithms are designed to intensify the search towards an optimal solution. However, genetic algorithms require additional approaches to handle the constraints discussed in Section 2.2.2.

4. New Approach for VRP with Constraints

We propose a genetic algorithm based on insertion heuristics without considering any additional local search methods for the improvement. The definition "genetic algorithm" can describe either a general approach or a set of the specific genetic operators. In this paper the proposed version of genetic algorithm for VRP with constraints will be called "new genetic algorithm" further on to distinguish it from other approaches.

Genetic algorithms and insertion heuristics combine together their best characteristics in search for the optimal solution. It is generally accepted that any genetic algorithm for solving a problem should have basic components, such as a genetic representation of solutions, the way to create the initial solution, the evaluation function for ranking solutions, genetic operators, values of the parameters (i.e. population size, probabilities for applying genetic operators, etc.). In Sections 4.1–4.4 there are described all components of the proposed genetic algorithm in more detail.

4.1. *Incorporating Insertion Heuristics*

As already mentioned in Section 3, in some genetic algorithm approaches the insertion heuristic is used in the initialisation step, where Solomon I1 method has been used with random coefficients for the criterion function. Although such a random insertion is possible, randomization is limited by the defined criterion function. Random insertion heuristic is chosen in the proposed algorithm to maintain stochastic characteristics of the genetic algorithm.

Let us assume that we have a set of nodes $N = \{n_0, \dots, n_k\}$, where $N \backslash \{n_0\}$ are the nodes that should be visited by a single vehicle and n_0 is the depot. The constructed partial solution is $x_0 = (\{n_0\},\ r_0 = \emptyset,\ N_{r0} = N \backslash \{n_0\})$, where r_0 is the empty set of arcs, N_{r0} is a set of unvisited nodes.So the solution contains only the depot n_0.

In the first iteration the random selected node n_{r1} from N_{r0} is inserted into a partial solution x_0. The new constructed partial solution is $x_1 = (\{n_0, n_{r1}\},\ r_1 = \{(n_0, n_{r1}), (n_{r1}, n_0)\},\ N_{r1} = N_{r0} \backslash \{n_{r1}\} = N \backslash \{n_0, n_{r1}\})$. Two new arcs, (n_0, n_{r1}) and (n_{r1}, n_0), have been created in the solution. Assume that the route is feasible and it can be agreed that it would be the shortest route for a single customer problem $\{n_0, n_{r1}\}$.

In the second iteration a random node n_{r2} is selected from N_{r1}. For the newly selected node there exist two possible places for insertion in the solution x_1: either in the arc (n_0, n_{r1}) or in the arc (n_{r1}, n_0). Assume that both insertions are feasible and the arc (n_{r1}, n_0) has a lower insertion cost than the arc (n_0, n_{r1}). So the newly constructed partial solution is $x_2 = (\{n_0, n_{r1}, n_{r2}\},\ r_2 = \{(n_0, n_{r1}), (n_{r1}, n_{r2}), (n_{r2}, n_0)\},\ N_{r2} = N \backslash \{n_0, n_{r1}, n_{r2}\})$. The newly constructed partial solution is feasible and optimal.

In the third iteration another random node n_{r3} is selected from N_{r2} and a new optimal solution x_3 is created.

In each next iteration k a random node n_{rk} is selected from $N_{r(k-1)}$. If there exists such an arc in r_{k-1}, where the inserted node does not violate any constraints and produces a new feasible partial solution x_k,the added node n_{rk} removes the existing arc (n_i, n_j) and adds two new arcs (n_i, n_{rk}) and (n_{rk}, n_j). If we find the optimal partial solution in the iteration $k - 1$, the solution created in iteration k is not necessarily optimal, because two new arcs (n_i, n_{rk}) and (n_{rk}, n_j) are created and there can exist a shorter path to some nodes in the route r_k.

The random insertion heuristic with only one minimization objective, i.e. travelling salesman problem (TSP) with the total travelling path minimization, has a complexity $O(k^2)$ to construct a single solution where the complexity of search for the best arc to insert a single node is $O(k)$. When adding additional constraints, the computation time is affected. Solving VRPTW problem by the insertion heuristic has the complexity $O(k^3)$. The handling time window constraint involves additional check for any violations occurring in a partial route after inserting a new node. So, for each node to be inserted the best arc search has the complexity $O(k^2)$, where the insertion complexity for all nodes is $O(k^3)$.

As already mentioned, when solving the problem with constraints by the genetic algorithm, the constraint violation is checked per solution, usually in the form of penalty or repair cost. In the proposed algorithm the constraint violation is evaluated in the insertion. For each randomly selected node a feasible insertion needs to be determined, where feasible insertion means finding such an arc of a partial solution, where the inserted node as well as all the previously added nodes do not violate any constraints. The partial solution with a new inserted node should remain feasible. Let us define the function $h_c(n, a)$ that evaluates the violation for the certain constraint c with the newly inserted node n in arc a.

The function h_c is similar to the function f_c, where f_c evaluates the whole solution for constraint violation, but h_c is applied to a single node insertion only. The function h_c is defined here as follows: $h_c(n, a) = 0$, if the constraint $c \in C$ is satisfied, and $h_c(n, a) > 0$, otherwise. Insertion of the node n into the arc a is feasible, if $H_c(n, a) = 0$, where

$$H_c(n, a) = \sum_{c \in C} h_c(n, a).$$

Subject to the insertion order and the constraint set, a partial route can be constructed in such a way that no additional nodes can be inserted without violating constraints. So a random insertion heuristics does not always guarantee the creation of a feasible solution, but the feasibility can be preserved in a partial solution. An infeasible insertion would require an additional function definition in order to determine which arc is less infeasible than others. However, such an approach would involve the definition of constraint hierarchy and any decision variables for ranking constraint or different penalty approaches for the evaluation of the constraint violation.

In order to avoid any additional complexities we define the solution x of the genetic algorithm as follows:

$$\begin{aligned}
&x = \big(R = \{r_1, \ldots, r_t\},\ U = \{n_1, \ldots, n_u\}\big),\\
&r_i = (N_i, A_i), \qquad N_i = \{n_0, n_{i_1}, \ldots, n_{i_p}\},\\
&A_i = \big\{a_{i_1} = (n_0, n_{i_1}),\ a_{i_2} = (n_{i_1}, n_{i_2}),\ \ldots,\ a_{i_p} = (n_{i_{p-1}}, n_{i_p}),\ a_{i_{p+1}} = (n_{i_p}, n_0)\big\},\\
&N_v = \{N_1 \cup \cdots \cup N_t\} \backslash n_0
\end{aligned}$$

where each route $r_i \in R$ represents a vehicle travelling path and U is a set of unassigned nodes left due to constraint violation. The single route r_i is represented as a graph, where each arc a_i is the shortest path between two nodes. Set U is part of the solution x, where $N_v \cup U = N$ and $N_v \cap U = \emptyset$. If set U is empty, then the solution x is feasible or infeasible, otherwise.

In the proposed algorithm the insertion is carried out as follows. An initial solution has an empty route list ($R = \emptyset$) and an empty unassigned node list ($U = \emptyset$). The node n_r is randomly selected from the set N. All arcs are checked for feasible insertion of the selected node. All the arcs that pass the constraint violation check are then evaluated by the insertion cost function $c(n_s, n_r, n_t)$, where the path length is the cost.The arc with the smallest value $c(n_s, n_r, n_t)$ is chosen for the insertion of the node n_r. If no arcs pass the violation check, a new route is started. If the maximum number of routes is reached, the node n_r is added to an unassigned node list U. The following pseudo-code describes the insertion process:

```
$N_u = N$
$R = \emptyset$
$U = \emptyset$
**while** $N_u \neq \emptyset$
```

```
        $n_r$ = select a random node from $N_u$
        $A$ = get all arcs from $R$
        for each constraint $c \in C$
            remove arcs from $A$ where insertion of $n_r$ violates constraint $c$
        end for
        if $A = \emptyset$ and $|R| < c_v$
            $r_i$ = new route($\{n_0, n_r\}$)
            add $r_i$ to $R$
        else if ($A \neq \emptyset$)
            find $a \in A$, $a = (n_s, n_t)$ by minimizing the function $c(n_s, n_r, n_t)$
            insert $n_r$ to $a$
        else
            add $n_r$ to $U$
        end if
    end while
```

Figure 2 shows the insertion steps where a filled circle represents the depot node n_0, the arrows and empty circles represent the route r_i, a dotted circle represents the node n_r selected for insertion and the dotted arrows show possible insertion arcs. The maximum vehicle number constraint $c_v \in C$ check is integrated in the insertion process.

Capacity constraints $C_c \subseteq C$ define the maximal allowed amount of goods that can be assigned to a vehicle. As in the VRP problem, the load increases by assigning a new node to the route. Let us define the function $d_c(n_j)$ that evaluates the load of the single node n_j. The condition $c \geqslant \sum d_c(n_j)$, $\forall n_j \in r_i$ and $c \in C_c$, should not be violated in the VRPTW problem. The check of this constraint has the complexity $O(1)$ for a single node insertion. In the VRPPD problem the load varies during travelling and the function $d_c(n_j)$ represents loading or unloading at a specified node n_j, where $d_c(n_j) > 0$ if goods are loaded and $d_c(n_j) < 0$ if goods are unloaded. Node insertion to one place can involve capacity violation in other places. The check for capacity constraint has the complexity $O(k)$ for a single node insertion. The function $g_c(a_i)$ calculates the current capacity space available in the arc a_i. The function $g_{fc}(a_i)$ is used to determine the maximal capacity available for the node insertion in the arc a_i without involving the constraint violation in the subsequent parts of the route.

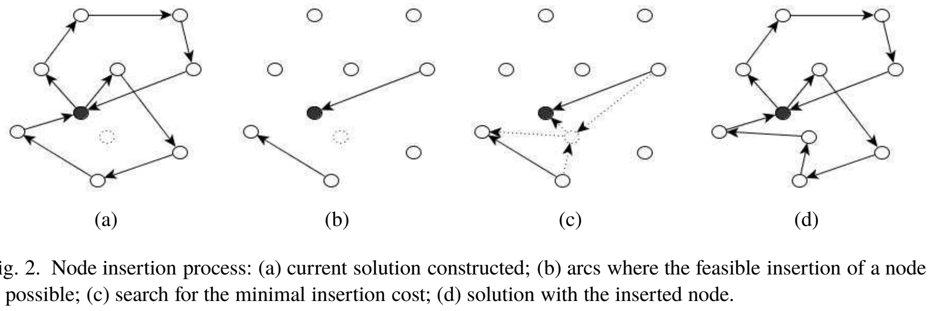

Fig. 2. Node insertion process: (a) current solution constructed; (b) arcs where the feasible insertion of a node is possible; (c) search for the minimal insertion cost; (d) solution with the inserted node.

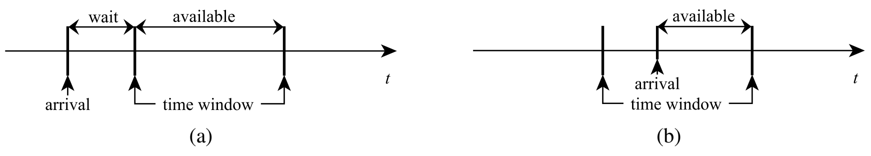

Fig. 3. Arrival at a customer with time window constraint: (a) arrival too early, (b) arrival in the time window.

$$g_{fc}(a_{i_s}) = \begin{cases} \min\{g_{fc}(a_{i_{s+1}}),\, g_c(a_{i_s})\} & \text{if } a_{i_{s+1}} \text{ exist,} \\ g_c(a_{i_s}) & \text{otherwise,} \end{cases}$$

$$g_c(a_{i_s}) = c - \sum_{j=0}^{i_s} d_c(n_j), \quad c \in C_c,\ n_j \in r_i.$$

Time window constraints $C_t \subseteq C$ have characteristics similar to capacity constraints in the VRPPD problem: adding a new node in one place can involve a constraint violation in other places of the current route. Figure 3 represents two possible situations of arrivals at customers. If a vehicle arrives at a customer in the defined time window, the available time is equal to the time left to the end of window (Fig. 3(b)). If a vehicle arrives too early, the waiting time is added to the available time (Fig. 3(a)).

The time window constraint evaluation does not increase the complexity of a single node insertion and it still remains $O(k)$. The function $g_{ft}(a_i)$ is used to determine the maximal amount of time available for the node insertion in the arc a_i without involving a constraint violation in subsequent parts of the route. The constraint $c_{i_s} \in C_{tw}$ defines the end value of the time window.

$$g_{ft}(a_{i_s}) = \begin{cases} \min\{g_{ft}(a_{i_{s+1}}),\, g_t(a_{i_s})\} & \text{if } a_{i_{s+1}} \text{ exist,} \\ g_t(a_{i_s}) & \text{otherwise,} \end{cases}$$

$$g_t(a_{i_s}) = c_{i_s} - \big(t_a(n_{i_s}) + t_w(n_{i_s})\big), \quad c_{i_s} \in C_{tw},\ n_{i_s} \in r_i,$$

$$t_a(n_{i_s}) = \begin{cases} t_t(n_0, n_{i_s}) & \text{if } s = 1, \\ t_a(n_{i_{s-1}}) + t_w(n_{i_{s-1}}) + t_s(n_{i_{s-1}}) + t_t(n_{i_{s-1}}, n_{i_s}) & \text{otherwise,} \end{cases}$$

$t_s(n_{i_s})$ – service time, $\quad t_t(n_{i_{s-1}}, n_{i_s})$ – travel time, $\quad t_w(n_{i_s})$ – waiting time.

The pick-up and delivery constraints $C_{pd} \subseteq C$ connect pick-up and delivery nodes with a logical relation. In order to determine the arcs for a feasible insertion of node n_r, the opposite node n_{op} (pick-up or delivery node) has to be examined. If n_{op} is not yet assigned to the route, all arcs in the partial solution remain competitive for the insertion of the node n_r. If n_{op} has already been assigned to the route, the following rules are applied:

- if n_r is a delivery node, arcs where the insertion of node n_r is possible, are in the same route as the node n_{op} and after the node n_{op};
- if n_r is a pick-up node, arcs, where the insertion of node n_r is possible, are in the same route as the node n_{op} and before the node n_{op}.

In the initialisation of the genetic algorithm an initial population with the above described random insertion process is created, where the creation of single solution has complexity $O(k^2)$. The insertion process still allows to generate infeasible solutions where unassigned node set U is not empty. In the proposed algorithm a feasible solution is always treated better than the infeasible solution. The following pseudocode shows how a better solution $x_{\min}$ is identified from the two solutions x_i and x_j:

```
***If*** $f_u(x_i) \neq f_u(x_j)$, *where* $f_u(x_i) = |U_i|$, $U_i \in x_i$
    ***if*** $f_u(x_i) < f_u(x_j)$
        $x_{\min} = x_i$
    ***else***
        $x_{min} = x_j$
***else if*** $f_v(x_i) \neq f_v(x_j)$
    ***if*** $f_v(x_i) < f_v(x_j)$
        $x_{\min} = x_i$
    ***else***
        $x_{\min} = x_j$
***else***
    ***if*** $f_d(x_i) < f_d(x_j)$
        $x_{\min} = x_i$
    ***else***
        $x_{\min} = x_j$
```

Sections 4.2, 4.3, 4.4 present the proposed genetic algorithm and genetic operators with a defined feasible insertion.

4.2. *Genetic Algorithm*

In the proposed genetic algorithm crossover and mutation operators are defined in the "remove and reinsert" approach. The approach is similar to a single point relocation method where the node is extracted and inserted into a different place. However, reinsertion of a single node in a different place can be unsuccessful, because the constructed routes have reached constraint limits and cannot be extended by an additional node. If a single node has been chosen for reinsertion, there is a large probability that the node will be inserted in the same place from which it has been removed. In order to enable the node reinsertion, multiple nodes have to be extracted. The crossover and mutation operators that follow the idea of node reinsertion are defined in Sections 4.3 and 4.4.

In the proposed algorithm the mutation operator is applied with probability $MP = 0.1$ and the crossover operator is applied to all individuals selected for mating. In the crossover operator new offspring are generated from two selected parent solutions where the first one is selected from the best individuals and the second one is selected randomly from the whole population. The new offspring are added to the population and the worst individuals are removed from the population to keep the same population size in each iteration. The defined mutation operators are based on a random insertion and can produce individuals that will not survive. In order to increase the probability of the mutation operator to generate individuals that will survive, a second population is created. The following pseudo-code represents the proposed genetic algorithm:

```
$Pop_1$ – initial population of size $PS_1$
(create a set of solutions using the feasible insertion method)
while number of iterations without improvement $< IL_1$ and time $< TL_1$
    $sort(Pop_1)$ – sort individuals with a defined comparison function
    remove $(|Pop_1| - PS_1)$ worst individuals from $Pop_1$
    for $i = 1 \ldots PL_1$
        $x_{p11} = Pop_1[i]$ – select first parent
        $x_{p12} = Pop_1[r_i]$, where $r_i = random(1, |Pop_1|)$, $r_i \neq i$ – select second parent
        $x_{c11} = crossover(x_{p11}, x_{p12})$ – generate offspring
        $x_{c12} = crossover(x_{p12}, x_{p11})$
        add $x_{c11}$ and $x_{c12}$ to $Pop_1$
        if $random(0, 1) < MP$ – apply the mutation operator with probability $MP$
            $(x_{mp1}, N_{mp1})$ – create partial solution $x_{mp1}$ and list $N_{mp1}$ by mutating $x_{p11}$
            $Pop_2$ – create population of size $PS_2$ by inserting $N_{mp1}$ to $x_{mp1}$
            while number of iterations without improvement $< IL_2$
                $sort(Pop_2)$ – sort individuals with a defined comparison function
                remove $(|Pop_2| - PS_2)$ worst individuals from $Pop_2$
                for $j = 1 \ldots PL_2$
                    $x_{p21} = Pop_2[j]$ – select the first parent
                    $x_{p22} = Pop_2[rj]$, where $r_j = random(1, |Pop_2|)$, $r_j \neq j$
                    $x_{c21} = crossover(x_{p21}, x_{p22})$
                    $x_{c22} = crossover(x_{p22}, x_{p21})$
                    add $x_{c21}$ and $x_{c22}$ to $Pop_2$
                    if $random(0, 1) < MP$
                        $x_{m2}$ = generate offspring by mutating parent $x_{p21}$
                        add $x_{m2}$ to $Pop_2$
                    end if
                end for
            end while
            $x_{m1}$ = select the best individual from $Pop_2$
            add $x_{m1}$ to $Pop_1$
        end if
    end for
end while
the best solution is first solution in $Pop_1$.
```

Crossover and mutation operators are randomly selected from operators defined in Sections 4.3 and 4.4. In the mutation operator the new population is created in two steps. Firstly, the mutation operator is applied to the solution x_{p11} to create a partial solution x_{mp1} with some routes left as well as the set of nodes N_{mp1} that needs to be reinserted back. Then Pop_2 is created by copying the partial solution x_{mp1} and inserting N_{mp1} using a random insertion. Computation of Pop_2 stops when the best solution is not improved initerations IL_2. The stop criterion in Pop_2 intentionally does not include the maximal time limit. The value IL_2 is chosen to be small to avoid redundant computation in Pop_2. The values used in the experimental evaluation are as follows: $PS_1 = 100$, $PL_1 = 10$, $IL_1 = 50$, $TL_1 = 5$ min, $MP = 0.1$, $PS_2 = 20$, $IL_2 = 5$, $PL_2 = 2$.

4.3. *Crossover Operator*

In the genetic algorithm the crossover operators generate new solutions from the chosen parent solutions x_i and x_j. We propose crossover operators which intersect two solutions in the defined logical ways. The output of such a process is the set of routes $R_{offspring}$ and the set of extracted nodes N_{temp}. All unassigned nodes are inserted back using the defined random insertion. The aim of the crossover is to identify the parts of solution that more probably are optimally constructed than other parts of solution. The benefit of crossover operators that are based on the common part search between parent solutions is shown in Vaira and Kurasova (2013). Two different crossovers are defined to increase the probability of convergence to the global optimum where each crossover produces an offspring by focusing on different information obtained from parents.

In the proposed genetic algorithm, two crossover operators are defined. Each of them produces a single offspring partial solution $x_{offspring}$ from the parent solutions x_i and x_j. The first crossover operator intersects two sets of arcs $A_i \in x_i$ and $A_j \in x_j$:

```
$N_{temp} = U_i$
$x_{offspring}$ – offspring solution
for each arc $a_i \in A_i$
    if $a_i$ exist in $A_j$
        add $a_i$ to $x_{offspring}$
    else
        add node $n_s \in a_i$ to $N_{temp}$, where $n_s$ is the starting node of arc $a_i$
    end if
end for
insert nodes from $N_{temp}$ to $x_{offspring}$ using the defined random insertion
```

The second crossover operator intersects the routes $R_i \in x_i$ and $R_j \in x_j$ according to the visited node sets N_i and N_j:

```
$N_{temp} = U_j$
$x_{offspring}$ – offspring solution
for each route $r_s \in R_j$
    $R_{temp} = \emptyset$ – temporary set of routes
    for each node $n \in r_s$, $n \neq n_0$
        $r_t$ = find route in $R_i$, where $n \in r_t$
        if $r_t$ not found
            add $n$ to $N_{temp}$
        else if $r_{t(tmp)} \in R_{temp}$
```

```
                assign node n to $r_{t(tmp)}$
            else
                $r_{t(tmp)}$ = new route $(\{n_0, n\})$
                add $r_{t(tmp)}$ to $R_{temp}$
            end if
        $r_{best}$ = select a route with the maximal number of nodes visited from $R_{temp}$
        assign all nodes $n \in R_{temp} \backslash r_{best}$ to $N_{temp}$
    end for
    insert nodes from $N_{temp}$ to $x_{offspring}$ using the defined random insertion
```

The second crossover examines all nodes in each route and groups them into partial routes according to the attendance in the routes from the opposite solution. The partial route with the maximum number of nodes is selected to preserve the path. All other partial routes are discarded by adding nodes to the unassigned node list N_{temp}. In the worst case the crossover operators have a complexity of $O(k^2)$. Figure 4 represents behaviour of both crossover operators, where (a), (b) show two parent solutions, (c) shows the offspring obtained by the first crossover, (d) shows the offspring obtained by the second crossover. Dotted circles show unassigned nodes N_{temp} that will be inserted back. Dotted lines in (c) connect intersected arcs, and in (d) intersected node sets displayed as grey circles.

4.4. *Mutation Operator*

Mutation operators deal with a single solution x_i. The proposed mutation operators are similar to the defined crossover operators. The designed mutation operators extract a subset of nodes from the solution in the defined ways and reinsert them back by applying a random insertion. By extracting a set of nodes we aim to preserve one part of the solution and reorganize the other one. A set of mutation operators that are applied by selecting one of them randomly is defined.

The first mutation operator selects a node set for extraction randomly with the limit of $0.5z|N|$, where z is a random number in the range $(0, 1)$. The complexity of random extraction is $O(k)$.

The second operator picks up a random node n_r from x_i and extracts the set of nodes closest to n_r by minimizing the distance function $l(n_r, n_i)$, where $\forall n_i \in R_i$:

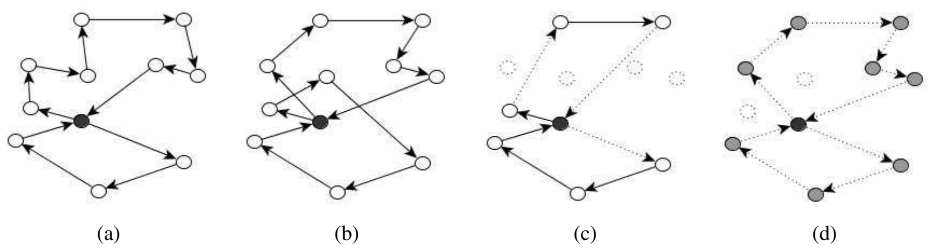

Fig. 4. Crossover operators: (a), (b) two parents; (c) intersection of the first crossover (d) intersection of the second crossover.

```
$x_m = x_i 4$
$N_{temp} = U_m$
$U_m = \emptyset$
$n_r$ = select a random node from $x_m$
extract $n_r$ from $x_i$
add $n_r$ to $N_{temp}$
while not limit reached
        $n_{ri}$ = find the nearest node to $n_r$ in $x_m$
        extract $n_{ri}$ from $x_m$
        add $n_{ri}$ to $N_{temp}$
end while
insert nodes from $N_{temp}$ to $x_m$ using the defined random insertion
```

The number of extracted nodes is limited to $0.5z|N|$, where z is a random number in the range $(0, 1)$. The complexity of the second mutation operator is $O(k^2)$.

The third mutation operator extracts random routes with the limit of $0.5z|R_i|$ routes, where $R_i \in x_i$. The complexity of the described method is $O(k)$.

The fourth mutation operator extracts nodes with the longest detour. The search selects nodes with maximal values of the function $l_r(n_r) = l(n_{r-1}, n_r) + l(n_r, n_{r+1}) - l(n_{r-1}, n_{r+1})$. The number of extracted nodes is limited to $0.5z|N|$, where z is a random number in the range $(0, 1)$. In the worst case the complexity of the fourth mutation operator is $O(k^2)$. The fourth mutation operator is combined with other mutation operators where initially the first mutation operator is applied and then the fourth mutation operator is applied with probability 0.1.

The fifth mutation operator searches for nodes visited around the same time. This mutation operator is similar to the second mutation operator. At first, a random node

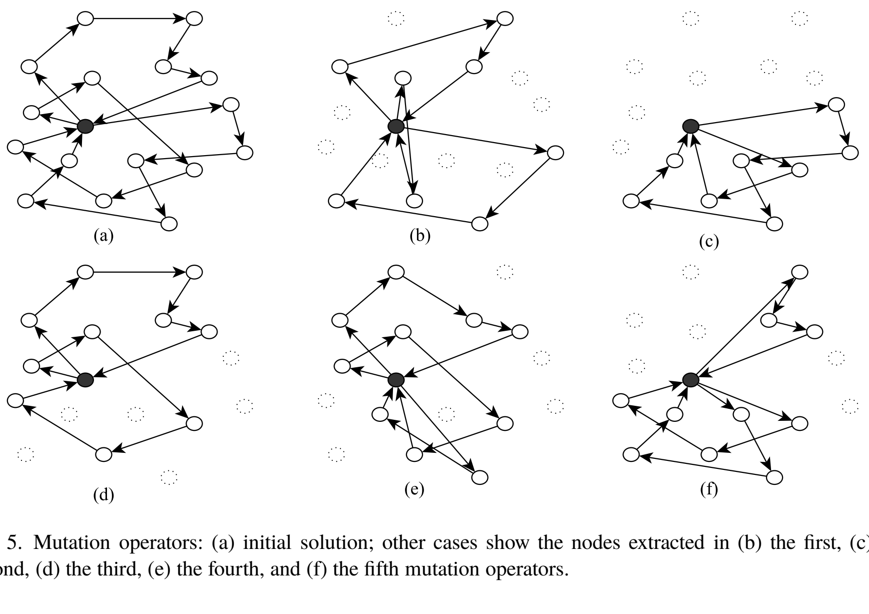

Fig. 5. Mutation operators: (a) initial solution; other cases show the nodes extracted in (b) the first, (c) the second, (d) the third, (e) the fourth, and (f) the fifth mutation operators.

nr is selected, afterwards other nodes are selected by minimizing the function $t_r(n_i) = |t_a(n_r) - t_a(n_i)|$, where $\forall n_i \in R_i$, $R_i \in x_i$. The fifth mutation operator is applied when time constraints are defined in the problem. The complexity of the fifth mutation operator is $O(k^2)$.

In Fig. 5, the behaviour of mutation operators is presented where a filled circle shows a depot, empty circles show visited nodes and dotted circles show the extracted nodes.

5. Experimental Evaluation

The proposed algorithm is tested using two problem sets. The first set includes the well-known Solomon instances of the VRPTW problem where all instances have 100 customers, distributed over the geographical area (Solomon, 1987). 56 VRPTW instances are categorized as:

- set C – customers are located in geographical clusters;
- set R – customers are randomly distributed;
- set RC – some customers are randomly distributed and some customers are located in clusters.

Problem instances are also split into Class 1 (R1, C1, RC1) and Class 2 (R2, C2, RC2), where Class 1 defines problems with a small vehicle capacity and narrow time windows. Class 2 defines problems with a large vehicle capacity and large time windows. Each Solomon problem instance defines the central depot, the maximum vehicle number, limits of vehicle capacity, demands for each node and also the maximum travel time for a single vehicle.

The second set of instances is defined for the VRPPD problem (Li and Lim, 2003). VRPPD instances LC1, LC2, LR1, LR2, LRC1, LRC2 are generated from Solomon problem sets C1, C2, R1, R2, RC1, RC2 respectively. Problem instances have 100 customers. VRPPD instances include the central depot, time window constraints, pick-up and delivery nodes and the maximal travel time for a single vehicle. For both, VRPPD and VRPTW problems, the distance between nodes is taken as a travelling time value.

The proposed algorithm is implemented using the Java programming language. All computations are performed by the personal computer (Intel Core 2 Duo 2.2 GHz CPU, 4 GB RAM). In the experimental evaluation, parameter values in the genetic algorithm are defined as follows: $PS_1 = 100$, $PL_1 = 10$, $IL_1 = 50$, $TL_1 = 5$ min, $MP = 0.1$, $PS_2 = 20$, $IL_2 = 5$, $PL_2 = 2$. All the obtained results are compared with the best results obtained by other genetic algorithms in following papers: [1] (Solomon, 1987); [2] (Berger *et al.*, 1998); [3] (Ho *et al.*, 2001); [4] (Tan *et al.*, 2001a); [5] (Tan *et al.*, 2001b); [6] (Jung and Moon, 2002); [7] (Bent and Hentenryck, 2003); [8] (Li and Lim, 2003); [9] (Zhu, 2003); [10] (Berger and Barkaoui, 2004); [11] (Alvarenga *et al.*, 2005); [12] (Ombuki *et al.*, 2006); [13] (Tan *et al.*, 2006); [14] (Hasle and Kloster, 2007): [15] (Garcia-Najera and Bullinaria, 2011).

In Tables 1–6, the results of VRPTW instances are summarized. The first column defines the problem instance name; three next columns present the best known solutions, the

Table 1
Results of problem set R1.

Problem	Best distance/vehicles			Average distance/vehicle		Average CPU time of the proposed algorithm
	Best solution	Best GA solution	Best solution of the proposed algorithm	Best GA solution	Solution of the proposed algorithm	
R101	1645.79/19	1650.8/19 [15]	**1650.8/19**	1693.23/19.6 [12]	**1650.8/19**	42.5
R102*	1486.12/17	1487.31/17 [15]	**1486.12/17**	1525.46/18.2 [12]	**1487.04/17**	27.27
R103	1292.68/13	1299.18/13 [15]	**1296.29/13**	1281.32/13.8 [12]	**1234.48/13.8**	31.48
R104	1007.24/9	999.82/10 [15]	**982.02/10**	1035.10/10 [12]	**989.96/10**	48.09
R105*	1377.11/14	1377.11/14 [15]	**1377.11/14**	1430.86/14.9 [12]	**1385.56/14**	52.85
R106	1251.98/12	1263.21/12 [15]	**1252.03/12**	1298.27/12.8 [12]	**1259.28/12**	59.64
R107	1104.66/10	1164.14/11 [6]	**1117/10**	1115.87/11 [12]	**1127.04/10**	70.20
R108	960.99/9	**960.99/9** [10]	968.97/9	990.39/10 [12]	**970.18/9**	51.8
R109	1194.73/11	1156.05/12 [15]	**1245.32/11**	1244.87/12.5 [12]	**1175.5/11.8**	21.12
R110	1118.59/10	1119/10 [10]	**1119/10**	1146.11/11.9 [12]	**1091.95/10.9**	43.66
R111	1096.72/10	1084.76/11 [12]	**1096.74/10**	1132.51/11 [12]	**1107.13/10**	75.04
R112	982.14/9	**953.63/10** [6]	962.03/10	1022.51/10.3 [12]	**977.05/10**	52.71

Table 2
Results of problem set R2.

Problem	Best distance/vehicles			Average distance/vehicle		Average CPU time of the proposed algorithm
	Best solution	Best GA solution	Best solution of the proposed algorithm	Best GA solution	Solution of the proposed algorithm	
R201	1252.37/4	1253.32/4 [15]	**1253.23/4**	1313.23/4 [12]	**1262.83/4**	17.32
R202	1191.7/3	1081.6/4 [15]	**1195.3/3**	1114.77/4 [12]	**1196.60/3**	27.76
R203	939.50/3	959.75/3 [15]	**947.09/3**	974.51/3 [12]	**966.71/3**	14.92
R204	825.52/2	760.82/3 [12]	**846.42/2**	777.37/3 [12]	**849.17/2**	60.68
R205	994.42/3	1030.92/3 [15]	**1029.1/3**	1070.66/3 [12]	**1052.89/3**	31.00
R206	906.14/3	919.73/3 [12]	**918.75/3**	949.25/3 [12]	**932.26/3**	26.07
R207*	890.61/2	821.32/3 [12]	**890.61/2**	848.30/3 [12]	**911.02/2**	88.19
R208*	726.82/2	736.47/2 [15]	**726.82/2**	747.98/3 [12]	**734.53/2**	37.43
R209	909.16/3	921.37/3 [15]	**913.14/3**	955.46/4 [12]	**931.54/3**	40.72
R210	939.34/3	954.12/3 [10]	**954.12/3**	999.02/3 [12]	**969.81/3**	29.89
R211	885.71/2	906.19/2 [10]	**900.88/2**	823.34/3 [12]	**929.60/2**	80.99

best known solutions obtained by other genetic algorithms and the best solution obtained by the proposed algorithm, respectively. In the last three columns there are presented the published best average results obtained by other genetic algorithms, the proposed algorithm average results and the average CPU time used in calculation.

Table 7 shows the best results, that are averaged over categories (C, R, RC). The columns show the results from different articles as well as the average results obtained by the proposed algorithm.

In Tables 8–13, the results of VRPPD instances are presented. The first column shows instance names, in the second column the best known results are presented; the third column presents the best results, obtained by the proposed algorithm, and the last two columns show the average results and average CPU time obtained by the proposed algorithm.

Table 3
Results of problem set C1.

Problem	Best distance/vehicles			Average distance/vehicle		Average CPU time of the proposed algorithm
	Best solution	Best GA solution	Best solution of the proposed algorithm	Best GA solution	Solution of the proposed algorithm	
C101*	828.94/10	828.94/10[6]	**828.94/10**	828.94/10[6]	**828.94/10**	12.2
C102*	828.94/10	828.94/10[6]	**828.94/10**	828.94/10[6]	**828.94/10**	13.1
C103*	828.06/10	828.06/10[6]	**828.06/10**	828.06/10[6]	**828.06/10**	15.25
C104*	824.78/10	824.78/10[6]	**824.78/10**	824.78/10[6]	**824.78/10**	16.4
C105*	828.94/10	828.94/10[6]	**828.94/10**	828.94/10[6]	**828.94/10**	12.55
C106*	828.94/10	828.94/10[6]	**828.94/10**	828.94/10[6]	**828.94/10**	12.86
C107*	828.94/10	828.94/10[6]	**828.94/10**	828.94/10[6]	**828.94/10**	12.85
C108*	828.94/10	828.94/10[6]	**828.94/10**	828.94/10[6]	**828.94/10**	13.24
C109*	828.94/10	828.94/10[6]	**828.94/10**	828.94/10[6]	**828.94/10**	14.67

Table 4
Results of problem set C2.

Problem	Best distance/vehicles			Average distance/vehicle		Average CPU time of the proposed algorithm
	Best solution	Best GA solution	Best solution of the proposed algorithm	Best GA solution	Solution of the proposed algorithm	
C201*	591.56/3	591.56/3 [6]	**591.56/3**	591.56/3 [6]	**591.56/3**	12.34
C202*	591.56/3	591.56/3 [6]	**591.56/3**	591.56/3 [6]	**591.56/3**	12.57
C203*	591.17/3	591.17/3 [6]	**591.17/3**	591.17/3 [6]	**591.17/3**	13.29
C204*	590.6/3	590.6/3 [6]	**590.6/3**	590.6/3 [6]	**590.6/3**	15.03
C205*	588.88/3	588.88/3 [6]	**588.88/3**	588.88/3 [6]	**588.88/3**	12.68
C206*	588.49/3	588.49/3 [6]	**588.49/3**	588.49/3 [6]	**588.49/3**	12.85
C207*	588.29/3	588.29/3 [6]	**588.29/3**	588.29/3 [6]	**588.29/3**	12.86
C208*	588.32/3	588.32/3 [6]	**588.32/3**	588.32/3 [6]	**588.32/3**	12.88

The average results are obtained by executing the proposed algorithm 10 times for each problem instance when each time a new initial population is created.

The asterisks at the problem names in Tables 1–6 show which instance solution found by the proposed algorithm is equal to the best known solution. The results are compared in the same way as they have been defined in the objective: firstly, the vehicle numbers found are compared and afterwards the travelling distances found are compared. The best solutions of other genetic algorithms are compared to the best solutions found by the proposed algorithm and the best average results are compared with the average results found by the proposed algorithm. Better values are in bold in Tables 1–6.

The proposed algorithm shows very good results for the problem set C, where the average results are equal to the best known values and the computation time is very small. The results show that for other problem sets R and RC the best values are found only in some cases. However, the results obtained by the proposed algorithm in comparison with other genetic algorithm approaches show that the same or better results are obtained for 51 out of 56 problem instances for the best solutions, and the same or better results for 56 out of 56 problem instances comparing with the best average results published.

Table 5
Results of problem set RC1.

Problem	Best distance/vehicles			Average distance/vehicle		Average CPU time of the proposed algorithm
	Best solution	Best GA solution	Best solution of the proposed algorithm	Best GA solution	Solution of the proposed algorithm	
RC101	1696.94/14	1636.92/15 [12]	**1697.43/14**	1668.52/15.4 [12]	**1649.63/14.8**	76.94
RC102*	1554.75/12	1470.26/13 [13]	**1554.75/12**	1536.04/13.8 [12]	**1547.69/12.4**	63.34
RC103	1261.67/11	**1267.86/11** [13]	1273.81/11	1350.15/12 [12]	**1280.27/11**	84.99
RC104	1135.48/10	1136.81/10 [6]	**1135.83/10**	1184.29/10.4 [12]	**1141.37/10**	46.27
RC105	1629.44/13	**1629.44/13** [10]	1540.18/14	1618.63/15 [12]	**1556.01/14**	88.91
RC106	1424.73/11	**1424.73/11** [10]	1376.26/12	1450.3/12.8 [12]	**1390.15/12**	38.35
RC107	1230.48/11	1235.37/11 [15]	**1230.95/11**	1227.81/12.03 [6]	**1232.78/11**	58.62
RC108*	1139.82/10	1141.34/10 [12]	**1139.82/10**	1135.81/11 [6]	**1151.75/10**	88.54

Table 6
Results of problem set RC2.

Problem	Best distance/vehicles			Average distance/vehicle		Average CPU time of the proposed algorithm
	Best solution	Best GA solution	Best solution of the proposed algorithm	Best GA solution	Solution of the proposed algorithm	
RC201	1406.91/4	1423.73/4 [12]	**1417.45/4**	1492.67/4 [12]	**1435.06/4**	30.06
RC202	1365.65/3	1162.54/4 [15]	**1367.09/3**	1212.49/4 [12]	**1415.48/3**	105.47
RC203	1049.62/3	1058.33/3 [15]	**1058.33/3**	1152.64/3 [12]	**1088.31/3**	44.42
RC204*	798.46/3	801.90/3 [15]	**798.46/3**	826.19/3 [12]	**812.77/3**	28.03
RC205*	1302.42/4	1304.93/4 [15]	**1302.42/4**	1378.44/4 [12]	**1330.06/4**	20.93
RC206*	1146.32/3	1203.7/3 [12]	**1146.32/3**	1164.33/3.3 [12]	**1159.36/3**	43.09
RC207	1061.14/3	1093.25/3 [12]	**1070.85/3**	1052.13/3.7 [12]	**1080.66/3**	77.18
RC208*	828.14/3	834.88/3 [15]	**828.14/3**	938.24/3 [12]	**851.43/3**	28.13

Table 7
Travel distance and the number of vehicles, averaged over categories.

	[4]	[5]	[3]	[9]	[12]	[10]	[11]	[15]	Solution of the proposed algorithm
C1	861/10.1	860.62/10.1	833.32/10	828.9/10	828.48/10	**828.38/10**	**828.38/10**	**828.38/10**	**828.38/10**
C2	619/3.3	624.47/3.3	593/3	**589.86/3**	590.6/3	589.93/3	590.9/3	591.74	**589.86/3**
R1	1227/13.2	1314.79/14.4	1203.32/12.6	1242.7/12.8	1220.92/12.5	**1221.1/11.92**	1224/11.92	1187.32/13.08	1213.66/12.08
R2	980/5	1093.37/5.6	951.17/3.2	1016.4/3	938.75/3.1	975.43/2.73	1012/2.73	897.95/4	**961.44/2.73**
RC1	1427/13.5	1512.94/14.6	1382.06/12.8	1412/13	1386.35/12.12	**1389.89/11.5**	1417/11.5	1348.22/12.63	1370.01/11.75
RC2	1123/5	1282.47/7	1132.79/3.8	1201.2/3.7	1132.12/3.38	1159.37/3.25	1195/3.25	1036.65/5.63	**1126.75/3.25**

The values in bold in Table 7 show the minimal value compared at first according to the vehicle number found and then according to the shortest distance found. The results show that for problem sets C1 and C2 the proposed algorithm finds solutions that are equal to the best results. The proposed algorithm finds solutions that are better than other ones for problem sets R2 and RC2, where problems have large time windows. Better results were obtained in [10] for problems with narrow time windows, R1 and RC1.

Table 8
Results of problem set LR1.

Problem	Best distance/vehicles		Average results	
	Best solution	Solution of the proposed algorithm	Solution of the proposed algorithm	CPU time of the proposed algorithm
LR101	1650.8/19 [8]	**1650.8/19**	**1650.8/19**	15.315
LR102	1487.57/17 [8]	**1487.57/17**	**1487.57/17**	17.115
LR103	1292.68/13 [8]	**1292.68/13**	**1292.68/13**	17.661
LR104	1013.39/9 [8]	**1013.39/9**	**1013.39/9**	44.836
LR105	1377.11/14 [8]	**1377.11/14**	**1377.11/14**	15.959
LR106	1252.62/12 [8]	**1252.62/12**	**1252.62/12**	14.673
LR107	1111.31/10 [8]	**1111.31/10**	**1111.31/10**	20.001
LR108	968.97/9 [8]	**968.97/9**	**968.97/9**	16.577
LR109	1208.96/11 [14]	**1208.96/11**	**1208.96/11**	46.617
LR110	1159.35/10 [8]	**1159.35/10**	1167.55/10.7	50.68
LR111	1108.9/10 [8]	**1108.9/10**	**1108.9/10**	35.007
LR112	1003.77/9 [8]	**1003.77/9**	**1003.77/9**	41.956

Table 9
Results of problem set LR2.

Problem	Best distance/vehicles		Average results	
	Best solution	Solution of the proposed algorithm	Solution of the proposed algorithm	CPU time of the proposed algorithm
LR201	1253.23/10 [14]	**1253.23/10**	**1253.23/10**	13.59
LR202	1197.67/3 [8]	**1197.67/3**	1213.39/3.3	26.806
LR203	949.40/3 [8]	**949.40/3**	**949.40/3**	15.59
LR204	849.05/2 [8]	**849.05/2**	**849.05/2**	20.18
LR205	1054.02/3 [8]	**1054.02/3**	**1054.02/3**	16.68
LR206	931.63/3 [8]	**931.63/3**	**931.63/3**	14.40
LR207	903.06/2 [8]	**903.06/2**	921.41/2.3	29.76
LR208	734.85/2 [8]	**734.85/2**	**734.85/2**	17.75
LR209	930.59/3 [14]	**930.59/3**	939.92/3.1	16.6
LR210	964.22/3[8]	**964.22/3**	999.74/3	22.09
LR211	911.52/2 [14]	**911.52/2**	**911.52/2**	31.3

Table 10
Results of problem set LC1.

Problem	Best distance/vehicles		Average results	
	Best solution	Solution of the proposed algorithm	Solution of the proposed algorithm	CPU time of the proposed algorithm
LC101	828.94/10 [8]	**828.94/10**	**828.94/10**	12.245
LC102	828.94/10 [8]	**828.94/10**	**828.94/10**	12.458
LC103	1035.35/9 [7]	**1035.35/9**	1057.70/9	32.96
LC104	860.01/9 [14]	**860.01/9**	839.31/9.5	27.353
LC105	828.94/10 [8]	**828.94/10**	**828.94/10**	12.38
LC106	828.94/10 [8]	**828.94/10**	**828.94/10**	12.477
LC107	828.94/10 [8]	**828.94/10**	**828.94/10**	12.454
LC108	826.44/10 [8]	**826.44/10**	**826.44/10**	12.609
LC109	1000.6/9 [7]	1036.41/9	896.72/9.7	26.813

Table 11
Results of problem set LC2.

Problem	Best distance/vehicles		Average results	
	Best solution	Solution of the proposed algorithm	Solution of the proposed algorithm	CPU time of the proposed algorithm
LC201	591.56/3 [8]	**591.56/3**	**591.56/3**	12.265
LC202	591.56/3 [8]	**591.56/3**	**591.56/3**	12.307
LC203	585.56/3 [8]	591.17/3	591.17/3	12.479
LC204	590.60/3 [14]	**590.60/3**	**590.60/3**	13.166
LC205	588.88/3 [8]	**588.88/3**	**588.88/3**	12.432
LC206	588.49/3 [8]	**588.49/3**	**588.49/3**	12.546
LC207	588.29/3 [8]	**588.29/3**	**588.29/3**	12.516
LC208	588.32/3 [8]	**588.32/3**	**588.32/3**	12.475

Table 12
Results of problem set LRC1.

Problem	Best distance/vehicles		Average results	
	Best solution	Solution of the proposed algorithm	Solution of the proposed algorithm	CPU time of the proposed algorithm
LRC101	1708.80/14 [8]	**1708.80/14**	**1708.80/14**	18.2
LRC102	1558.07/12 [14]	**1558.07/12**	**1558.07/12**	20.383
LRC103	1258.74/11 [8]	**1258.74/11**	**1258.74/11**	19.47
LRC104	1128.40/10 [8]	**1128.40/10**	**1128.40/10**	16.787
LRC105	1637.62/13 [8]	**1637.62/13**	**1637.62/13**	23.077
LRC106	1424.73/11 [14]	**1424.73/11**	**1424.73/11**	36.105
LRC107	1230.15/11 [8]	**1230.14/11**	**1230.14/11**	19.488
LRC108	1147.43/10 [14]	**1147.43/10**	1168.4/10.7	25.531

Table 13
Results of problem set LRC2..

Problem	Best distance/vehicles		Average results	
	Best solution	Solution of the proposed algorithm	Solution of the proposed algorithm	CPU time of the proposed algorithm
LRC201	1406.94/4 [14]	**1406.94/4**	**1406.94/4**	42.016
LRC202	1374.27/3 [8]	**1374.27/3**	1392.59/3.6	37.966
LRC203	1089.07/03 [8]	**1089.07/03**	**1089.07/03**	18.329
LRC204	818.66/3 [14]	**818.66/3**	**818.66/3**	15.961
LRC205	1302.20/4 [8]	**1302.2/4**	**1302.20/4**	29.366
LRC206	1159.03/3 [14]	**1159.03/3**	**1159.03/3**	21.527
LRC207	1062.05/3 [14]	**1062.05/3**	**1062.05/3**	22.121
LRC208	852.76/3 [8]	**852.76/3**	**852.76/3**	18.563

The numbers in bold in Tables 8–13 for the VRPPD problem show where the best solutions, obtained by the proposed algorithm, are equal to the best known solutions. The results for VRPPD instances show that the solutions, found by the proposed algorithm, are equal to the best known solutions for 54 out of 56 problem instances and the average results found are equal to the best known solutions for 45 out of 56 problem instances. For

VRPTW and VRPPD instances the minimal computation time is for problem set C, where customers are located in clusters. It is worth mentioning that the results, obtained by the proposed algorithm, were identified on average in 38.97 seconds for VRPTW instances. The results obtained in Alvarenga *et al.* (2005) were found in 15 minutes by Pentium IV 2.4 GHz and in Berger and Barkaoui (2004) the presented results were found in 30 minutes by Pentium 400 MHz.

6. Conclusions

The proposed new genetic algorithm is based on insertion heuristics for the vehicle routing problem with constraints, where the insertion is combined with genetic algorithm operators. As the results show, the proposed genetic algorithm finds solutions that in most cases are better than the ones found by other genetic algorithms. Although the solutions are not equal to the best known solutions in all cases, they are found in a reasonably short time. That makes the proposed algorithm competitive with other known algorithms.

The proposed algorithm can be applied to any problem that can be expressed as a graph. Mutation and crossover operators of the proposed genetic algorithm are based on a random insertion heuristic. The operators are not designed to a certain specific problem and can be applied to different problems. It allows to apply the proposed algorithm in general cases. The proposed algorithm has been applied to two different problems (VRPTW, VRPPD). However, no additional improvement/repair algorithms or local search algorithms are used here.

In order to apply the proposed algorithm to problems with additional constraints, the check of constraints can be easily added to the node insertion process. As the experimental results show, properly defined genetic algorithm operators with the feasibility check allow to obtain good solutions in some acceptable computation time. Different selection methods of nodes for reinsertion could be investigated further. Methods could be investigated for ranking the population as well as for selecting arcs in the node insertion process. Various improvement methods can be considered to improve the results. However, the proposed algorithm does not include improvement methods with a view to present the results obtained only by the genetic algorithm with a feasible insertion.

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.

Bent, R., Hentenryck, P.V. (2003). A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. In: *Proceedings of the 9th International Conference on the Principles and Practice of Constraint Programming (CP 2003)*, *Lecture Notes in Computer Science*, Vol. 2833. Springer, Berlin, pp. 123–137.

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

Berger, J., Salois, M., Begin, R. (1998). A hybrid genetic algorithm for the vehicle routing problem with time windows. In: *Proceedings of the12th Biannual Conference of the Canadian Society for ComputationalStudies of Intelligence*, *Lecture Notes in Computer Science*, Vol. 1418. Springer, Berlin, pp. 114–127.

Blanton, J. L., Wainwright, R. L. (1993). Multiple vehicle routing with time and capacity constraints using genetic algorithms. In: *Proceedings of the 5th International Conference on Genetic Algorithms (ICGA)*, Morgan Kaufmann, San Mateo, pp. 452–459.

Campbell, A.M., Savelsbergh, M.W.P. (2004). Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation Science*, 38(3), 369–378.

Dzemyda, G., Sakalauskas, L. (2011). Large-scale data analysis using heuristic methods. *Informatica*, 22(1), 1–10.

El-Mihoub, T.A., Hopgood, A.A., Nolle, L., Battersby, A. (2006). Hybrid genetic algorithms: a review. *Engineering Letters*, 13(2), 124–137.

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

Hasle, G., Kloster, O. (2007). Industrial vehicle routing problems. In: Hasle, G., Lie, K.-A., Quak, E. (eds) *Geometric Modelling, Numerical Simulation, and Optimization – Applied Mathematics at SINTEF*, Springer, Berlin, pp. 397–432.

Ho, W.-K., Ang, J.C., Lim, A. (2001). A hybrid search algorithm for the vehicle routing problem with time windows. *International Journal on Artificial Intelligence Tools*, 10(3), 431–449.

Hong, T.-P., Wang, H.-S., Lin, W.-Y., Lee, W.-Y. (2002). Evolution of appropriate crossover and mutation operators in a genetic process. *Applied Intelligence*, 16(1), 7–17.

Jung, S., Moon, B. R. (2002). A hybrid genetic algorithm for the vehicle routing problem with time windows. In: *Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2002)*, pp. 1309–1316.

Li, H., Lim, A. (2003). A metaheuristic for the pickup and delivery problem with time windows. *International Journal on Artificial Intelligence Tools*, 12(2), 173–186.

Lukasiewycz, M., Glass, M., Haubelt, C., Teich, J. (2008a). A feasibility-preserving local search operator for constrained discrete optimization problems. In: *Proceedings of IEEE Congress on Evolutionary Computation (CEC 2008)*, pp. 1968–1975.

Lukasiewycz, M., Glass, M., Teich, J. (2008b). A feasibility-preserving crossover and mutation operator for constrained combinatorial problems. In: *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature (PPSN)*, *Lecture Notes in Computer Science*, Vol. 5199. Springer, Berlin, pp. 919–928.

Michalewicz, Z. (1995a). Do not kill unfeasible individuals. In: *Proceedings of the 4th Intelligent Information Systems Workshop (IIS'95)*, pp. 110–123.

Michalewicz, Z. (1995b). A survey of constraint handling techniques in evolutionary computation methods. *Evolutionary Programming*, 135–155.

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.

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.

Potvin, J.-Y., Dubé, D. (1994). Improving a vehicle routing heuristic through genetic search. In: *Proceedings of the 1st IEEE Conference on Evolutionary Computation*, pp. 194–199.

Potvin, J.-Y., Rousseau, J.M. (1993). A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66, 331–240.

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

Rosenkrantz, D.J., Stearns, R.E., Lewis II, P.M. (1977). An analysis of several heuristics for the traveling Salesman problem. *SIAM Journal on Computing*, 6(3), 563–581.

Solomon, M.M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2), 254–265.

Tan, K.C., Chew, Y.H., Lee, L.H. (2006). A hybrid multiobjective evolutionary algorithm for solving vehicle routing problem with time windows. *Computational Optimization and Applications*, 34(1), 115–151.

Tan, K.C., Hay, L.L., Ke., O. (2001a). A hybrid genetic algorithm for solving vehicle routing problems with time window constraints. *Asia-Pacific Journal of Operational Research*, 18(1), 121–130.

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

Vaira, G., Kurasova, O. (2010). Modified bidirectional shortest path Dijkstra‘s algorithm based on the parallel computation. In: Barzdins, J., Kirikova, M. (eds.), *Proceedings of the 9th International Baltic Conference – Baltic DB&IS 2010*. University of Latvia Press, Riga, pp. 205–217.

Vaira, G., Kurasova, O. (2011). Parallel bidirectional Dijkstra's shortest path algorithm. In: *Databases and Information Systems VI*, *Frontiers in Artificial Intelligence and Applications*, Vol. 224. IOS Press, Amsterdam, pp. 422–435.

Vaira, G., Kurasova, O. (2013). Genetic algorithms and VRP: the behaviour of a crossover operator. *Baltic Journal of Modern Computing*, 1(3–4), 161–185.

Yeniay, O. (2005). Penalty function methods for constrained optimization with genetic algorithms. *Mathematical and Computational Applications*, 10(1), 45–56.

Yeun, L.C., Ismail, W.R., Omar, K., Zirour, M. (2008). Vehicle routing problem: models and solutions. *Journal of Quality Measurement and Analysis (JQMA)*, 4(1), 205–218.

Zhang, J., Chung, H.S.-H., Hu, B.J. (2004). Adaptive probabilities of crossover and mutation in genetic algorithms based on clustering technique evolutionary computation. In: *Proceedings of IEEE Congress on Evolutionary Computation (CEC2004)*, Vol. 2, pp. 2280–2287.

Zhang, J., Chung, H.S.-H., Lo, W.-L. (2007). Clustering-based adaptive crossover and mutation probabilities for genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 11(3), 326–335.

Zhu, K.Q. (2003). A diversity-controlling adaptive genetic algorithm for the vehicle routing problem with time windows. In: *Proceedings of the 15th IEEE International Conference on Tools for Artificial Intelligence (ICTAI 2003)*, pp. 176–183.

G. Vaira is a PhD student at the Institute of Mathematics and Informatics, Vilnius University, Lithuania. He obtained BSc degree from Vilnius University in 2006, and MSc degree from Vilnius Gediminas Technical University in 2009, both in the field of informatics. His research interests include exact and heuristic optimization algorithms and their application in routing problems.

O. Kurasova received the doctoral degree in computer science (PhD) from Institute of Mathematics and Informatics jointly with Vytautas Magnus University in 2005. Recent employment is at the System Analysis Department of the Vilnius University, Institute of Mathematics and Informatics as senior researcher, and at the Informatics Department of Lithuanian University of Educational Sciences as associate professor. Research interests include data mining methods, optimization theory and applications, artificial intelligence, neural networks, visualization of multidimensional data, multiple criteria decision support, parallel computing. She is the author of more than 40 scientific publications.

Genetinis algoritmas krovinių gabenimo uždaviniams spręsti, paremtas leistinu įterpimu

Gintaras VAIRA, Olga KURASOVA

Šiame straipsnyje yra pasiūlytas genetinis algoritmas krovinių gabenimo uždaviniams spręsti, kuris yra paremtas įterpimo euristika. Atsitiktinio įterpimo euristika yra naudojama pradinių sprendinių sukonstravimui bei esamų sprendinių rekonstravimui. Pasiūlyti genetinio algoritmo kryžminimo bei mutacijos operatoriai yra apjungti su atsitiktinio įterpimo euristikos metodu ir taip išlaiko stochastines genetinio algoritmo savybes. Skirtingai nuo kitų genetinių algoritmų, pasiūlyto genetinio algoritmo operatoriai nekonstruoja naujo sprendinio iš parinktų sprendinių dalių, tačiau, įvertinus ankstesnės generacijos sprendinius, identifikuoja tas sprendinio dalis, kurios turėtų išlikti, ir tas dalis, kurios turėtų būti perkurtos. Mutacijos operatoriuje yra sukuriama antra sprendinių populiacija ir vykdomas genetinis algoritmas. Gauti sprendiniai skiriasi nuo paprastos mutacijos sugeneruotų sprendinių tuo, kad šiems sprendiniams gauti buvo atliekama optimizacija. Tokie sprendiniai pirmoje populiacijoje turi didesnę tikimybę būti parinkti sekančios generacijos kūrimui taip padidinant diversifikavimą visame genetiniame algoritme. Skirtingai nuo kitų genetinių algoritmų, pasiūlytame algoritme nėra naudojami papildomi lokalios paieškos, pagerinimo ar taisymo metodai, kurie galėtų apriboti naujų sąlygų įtraukimą į algoritmą. Eksperimentiniai rezultatai rodo, kad sprendiniai rasti pasiūlytu algoritmu yra panašūs į kitų genetinių algoritmų rezultatus. Tačiau, daugeliu atveju pasiūlytas genetinis algoritmas suranda sprendinį per trumpesnį laiką.