Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 192 (2021) 1451–1460

Procedia Computer Science

www.elsevier.com/locate/procedia

25th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems

Feasibility-Preserving Genetic Operators for Hybrid Algorithms using TSP solvers for the Inventory Routing Problem

Krzysztof Michalak[a,*]

[a] *Department of Information Technologies, Wrocław University of Economics and Business, Wrocław, Poland*

Abstract

In this paper feasibility-preserving genetic operators for hybrid algorithms using TSP solvers for the Inventory Routing Problem (IRP) are studied. The IRP is a problem of jointly optimizing delivery schedules and routes for vehicles transporting products from a supplier to a number of retailers. This optimization problem is highly constrained, because limits on inventory levels as well as the vehicle capacity have to be taken into account. Moreover, the IRP is a generalization of the TSP and solving the TSP effectively is an important part of obtaining good solutions to the IRP. In this paper evolutionary algorithms are used for solving the IRP, but finding good routes is delegated to a state-of-the-art TSP solver. Therefore, genetic operators used in this paper focus on constructing the delivery schedule and not on optimizing the routes. In the experimental part of the paper an evolutionary algorithm using feasibility-preserving genetic operators is compared to the Infeasibility Driven Evolutionary Algorithm (IDEA) that uses the selective pressure to obtain feasible solutions. Presented results suggest that designing good feasibility-preserving genetic operators is important, because allowing the optimization algorithm to generate infeasible solutions and handling infeasibility in IDEA using the selective pressure leads to inferior results.

Keywords: Evolutionary Algorithms; Concorde Solver; Infeasibility Driven Evolutionary Algorithm
2020 MSC: 68W50; 90B06; 90C27; 90C29; 90C59

1. Introduction

The Inventory Routing Problem (IRP) introduced in [9] is an optimization problem involving joint optimization of delivery schedules and vehicle routes. In the most commonly investigated version of the IRP a single commodity is distributed from a supplier to n retailers using a homogeneous fleet of vehicles of capacity C, and deliveries have to be planned over a finite horizon H. On each day $t = 1, \ldots, H$ the supplier produces $r_{0,t}$ units of the product and retailer $s \in 1, \ldots, n$ sells $r_{s,t}$ units. In some problem formulations it is assumed that the production and sales are constant

E-mail address: krzysztof.michalak@ue.wroc.pl

1877-0509
10.1016/j.procs.2021.08.149

in time and thus denoted r_0 and r_s respectively. Inventory levels at time t are denoted $I_{s,t}$, where $s = 0, \ldots, n$ with the minimum and maximum allowed values constant in time, but possibly different for each of the retailers and the supplier, and denoted L_s and U_s, respectively (with an often assumed infinite storage capacity at the supplier, that is $U_0 = \infty$). Thus, inventory level constraints are: $\forall s = 0, \ldots, n \forall t = 1, \ldots, H : L_s \leq I_{s,t} \leq U_s$.

A solution to the IRP can be described as a set of routes $\pi_t : t = 1, \ldots, H$ assigned to each vehicle on consecutive days with the number of units of the product to be delivered $x_{s,t}$ specified for each retailer s for each date t. This solution representation can further be simplified if a replenishment policy is adopted, which allows determining the number of product units to deliver to each retailer. For example, in the up-to-level replenishment policy if a retailer is visited its inventory is filled to the maximum, so the amount of goods shipped to retailer s at time t is $x_{s,t} = U_s - I_{s,t}$ if $s \in \pi_t$ or 0 if $s \notin \pi_t$, where, as stated above, $I_{s,t}$ is the current inventory level at retailer s at time t. The up-to-level replenishment policy was to date used in numerous works on the IRP, among others in the paper [3] in which a well known set of benchmark IRP instances was defined. Regardless of how the delivered quantities are determined the capacity of the vehicle cannot be exceeded, so $\forall t = 1, \ldots, H : \sum_{s=1}^{n} x_{s,t} \leq C$.

Inventory costs are denoted h_s for $s = 0, \ldots, n$ and are assumed to be constant in time, but can be different at different locations. Transportation costs, in the most general case, are given for each pair of locations $i, j \in 0, \ldots n$ and are denoted $c_{i,j}$. In some problem formulations, however, they are calculated as Euclidean distances from locations given as coordinates on the real plane $\mathbb{R}^2$. For a given route $\pi_t = [\pi_{t,1}, \ldots, \pi_{t,k}]$ the transportation cost is $c(\pi_t) = c_{0,\pi_{t,1}} + \sum_{i=1}^{k-1} c_{\pi_{t,i},\pi_{t,i+1}} + c_{\pi_{t,k},0}$. Inventory costs and transportation costs are added together and thereby an evaluation of the given solution to the IRP is obtained.

The IRP problem variants studied in the literature include single- [3] and multi-depot [5] IRP, time windows [8, 12] and transshipments [11]. Methods used to solve the IRP include genetic algorithms (GA) [4], particle swarm optimization (PSO) [7, 16], simulated annealing (SA) [19], tabu search (TS) [1], variable neighbourhood search (VNS) [10, 14, 15], Greedy Randomized Adaptive Search Procedure (GRASP) [2], heuristic methods [12] and mathematical programming [6].

From the IRP definition presented at the beginning of this section it follows that a part of solving this optimization problem consists in tackling the Travelling Salesman Problem (TSP). For the TSP well-known problem solvers exist, such as the Concorde TSP Solver [20]. Solvers dedicated to the TSP are known to handle even very large instances efficiently. For example the Concorde TSP Solver found the optimal solutions to all 110 of the TSPLIB instances [18], the largest having 85900 locations. It seems, therefore, promising to study metaheuristic algorithms that focus on finding good delivery schedules and delegate solving the TSP to the dedicated solvers. From the point of view of the design of such metaheuristics, and especially the genetic operators this means treating the routes π_t as sets of locations with the ordering determined by the TSP solver. In this paper several genetic operators are studied that, instead of optimizing the routes, focus on finding good delivery schedules. The operators used with SGA preserve feasibility of solutions to the IRP.

The rest of this paper is structured as follows. In Section 2 genetic operators used for solving the IRP are described. Section 3 presents the evolutionary algorithms used in this paper. In Section 4 experiments and their results are discussed. Section 5 concludes the paper.

2. Solution Representation and Genetic Operators

This section presents the solution representation used in this paper, the genetic operators and the population initialization procedure. Because finding good routes is delegated to the TSP solver, the genetic operators used in this paper do not attempt to construct good routes, focusing instead on the assignment of retailers to particular dates. In this paper two types of genetic operators are used: preserving solution feasibility (denoted as "feasible" later in the text) and allowing infeasible solutions (denoted "infeasible"). Generally, for each infeasible operator a feasible counterpart is defined with the exception of two infeasible mutation operators one adding and the other removing retailers for which one feasible equivalent was defined which adds *or* removes a retailer depending on which of the moves preserves the feasibility of the solution. Presented operators use the following functions:

$URand(A)$ - draws an element from the given set with uniform probability. For example, $URand([0, 1])$ draws a real number uniformly from the [0, 1] range and $URand(\{1, 2, 3\})$ draws one of the three numbers with uniform probability.

$RandPerm(A)$ - returns the elements of a given set A in a random order.

$StockoutDate(S, s, t_0)$ - finds the first date on which a stockout occurs if no deliveries are made to the retailer s from the day t_0 onwards (inclusive).

2.1. Solution Representation and Evaluation

Solving the IRP requires determining which retailers to visit at each date $t = 1, \ldots, H$ and what quantities of the product $x_{s,t}$ to deliver for $s = 1, \ldots, n$. In this paper the up-to-level replenishment policy is used, so it suffices to store the vehicle route π_t for each date $t = 1, \ldots, H$ and the quantities $x_{s,t}$ are determined using this policy. Consequently, in the evolutionary algorithms studied in this paper solutions to the IRP are represented as the so-called jagged arrays which are a generalization of rectangular arrays allowing the "rows" to be of different length. Each solution $S = \{\pi_t : t = 1, \ldots, H\}$ consists of H vectors, each representing the route for the day $t = 1, \ldots, H$. Before evaluation, the routes in S are optimized using the Concorde TSP Solver [20]. Then, the evaluation procedure calculates:

- Daily inventory levels for all retailers $I_{s,t}$, where $s = 1, \ldots, n$, $t = 1, \ldots, H$.
- Daily vehicle loads V_t, where $t = 1, \ldots, H$.
- Total inventory costs $f_i(S) = \sum_{s=0}^{n} \sum_{t=1}^{H+1} h_s I_{s,t}$
- Total transportation costs $f_t(S) = \sum_{t=1}^{H} c(\pi_t)$, where $c(\pi_t)$ is calculated as described in Section 1.
- Total costs $f(S) = f_i(S) + f_t(S)$.
- Constraints $g_i(S), i = 1, 2, 3$ with $g_1(S) = -\sum_{t=1}^{H} \max(V_t - I_{0,t}, 0)$ and $g_1(S) < 0$ representing a stockout at the supplier, $g_2(S) = -\sum_{t=1}^{H} \max(V_t - C, 0)$ and $g_2(S) < 0$ representing the vehicle capacity being exceeded, $g_3(S) = -\sum_{t=1}^{H} \sum_{s=1}^{n} \max(L_s - I_{s,t}, 0)$ and $g_3(S) < 0$ representing the minimum inventory level constraint at the retailers being violated.

For a feasible solution S the predicate *IsFeasible* returns true: $IsFeasible(S) \equiv g_1(S) \geq 0 \wedge g_2(S) \geq 0 \wedge g_3(S) \geq 0$.

2.2. Supply at the Latest Date heuristic

The Supply at the Latest Date (SLD) heuristic determines the days at which the delivery has to be made to a given retailer s in order to avoid stockout. It works under the assumption that in order to lower the inventory costs it is profitable to deliver the goods as late as possible, so that they do not occupy storage space for long. The working of the SLD heuristic is presented in Algorithm 1. In the presented version the heuristic checks if the vehicle capacity C is not exceeded in order to preserve feasibility of generated solutions.

2.3. Crossover Operator

The crossover operator used in this paper builds two offspring solutions $O^{(1)}$ and $O^{(2)}$ by uniformly mixing delivery schedules for individual retailers obtained from two parent solutions $S^{(1)}$ and $S^{(2)}$. For each retailer s in turn (in a random order) the dates when the deliveries are made to retailer s are obtained from $S^{(1)}$ and $S^{(2)}$ and they are used to add retailer s to the delivery schedules in $O^{(1)}$ and $O^{(2)}$. The working of the crossover operator is presented in Algorithm 2. The *noInfeasible* parameter allows permitting or disallowing infeasible offspring to be generated by the crossover operator.

The crossover operator presented here also uses the parameter P_{cross}, which is the probability that the crossover procedure described in Algorithm 2 is used to generate the offspring for a given pair of parents. With probability $1 - P_{cross}$ the offspring are clones of the parents.

2.4. Mutation Operators

Mutation operators perform one of three operations: moving retailers from one day to another, adding a retailer or removing one. Mutation is applied to each solution with the probability P_{mut} and it makes at most $N_{mut} = \alpha_{mut} \cdot n$ changes to the solution. The α_{mut} parameter can be interpreted as mutation intensity, determining how profoundly the mutation operator changes the solution it is applied to. There are five variants of the mutation operator, which are

Algorithm 1: The Supply at the Latest Date (SLD) heuristic.

Inputs:
- $S = \{\pi_t : t = 1, \ldots, H\}$ - A solution to the IRP
- $V_t, t = 1, \ldots, H$ - Vehicle loads for the solution S
- s - A retailer for which to plan deliveries
- d - The day from which to start planning

Output:
- $S' = \{\pi'_t : t = 1, \ldots, H\}$ - A solution with the deliveries for retailer s planned

Precondition:
- $\forall t = d, \ldots, H : s \notin \pi_t$ - The retailer s is *not* included in the routes from the day d onwards.

$t_{prev} := d$ - 1
$t := d$
while $t \leq H$ **do**
  $q := 0$
  if $I_{s,t} - r_{s,t} < L_s$ **then** *// Stockout at day t*
    for $t_{ins} \in \{t, \ldots, t_{prev} + 1\}$ **do** *// Try to deliver on dates decreasing from t to $t_{prev} + 1$*
      $q := U_s - I_{s,t_{ins}}$ *// Delivered quantity determined using the up-to-level policy*
      if $V_{t_{ins}} + q \leq C$ **then** *// The vehicle is not overloaded*
        $\pi_{t_{ins}} := \pi_{t_{ins}} \cup \{s\}$
        $V_{t_{ins}} := V_{t_{ins}} + q$
        $t_{prev} := t_{ins}$
        break
  $I_{s,t} := I_{s,t} - r_{s,t} + q$
  $t := t + 1$

described in this section: feasible date change, infeasible date change, feasible retailer addition/removal, infeasible retailer addition, infeasible retailer removal.

Feasible date change mutation (FDCM) The feasible date change mutation operator considers retailers in a random order selecting, for each retailer, one date when it is visited, moving this retailer to some other feasible date and planning subsequent deliveries using the SLD heuristic. Details of this operator are presented in Algorithm 3.

Infeasible date change mutation The infeasible date change mutation operator performs N_{mut} changes to the solution $S = \{\pi_t : t = 1, \ldots, H\}$ by:

1. Randomly selecting a removal date t^-, such that $\pi_{t^-} \neq \emptyset$.
2. Randomly selecting a retailer $s \in \pi_{t^-}$.
3. Randomly selecting an addition date t^+, such that $s \notin \pi_{t^+}$.
4. Removing the retailer s from the route on the day t^-.
5. Adding the retailer s to the route on the day t^+.

All random selections are performed with uniform probability. If the original solution S contains only empty routes, a random retailer is added to one of them. If $\forall t : s \in \pi_t$ then it is not possible to find the t^+ to add the retailer at, so only the removal on the date t^- is performed.

Feasible retailer addition/removal mutation The feasible retailer addition/removal mutation attempts to perform N_{mut} changes to the solution S by creating, for each retailer s, candidate solutions with retailer s added or removed. From the generated candidate solutions one feasible solution is selected at random. The working of this mutation operator is presented in Algorithm 4.

Infeasible retailer addition mutation The infeasible retailer addition mutation applies N_{mut} changes to a solution $S = \{\pi_t : t = 1, \ldots, H\}$. Each change is made by randomly selecting a date $d \in \{1, \ldots, H\}$ and a retailer $s \notin \pi_d$. The retailer s is added to the route on the date d.

Algorithm 2: The crossover operator.

Inputs:
- $S^{(S1)} = \{\pi_t^{(S1)} : t = 1, \ldots, H\}$ - Parent 1
- $S^{(S2)} = \{\pi_t^{(S2)} : t = 1, \ldots, H\}$ - Parent 2
- $noInfeasible$ - Indicates whether to disallow building infeasible offspring

Output:
- $O^{(1)}$ and $O^{(2)}$ - Offspring solutions

$O^{(1)} = \{\pi_t^{(O1)} := \emptyset : t = 1, \ldots, H\}$ // *All routes in the offspring are empty at first*
$O^{(2)} = \{\pi_t^{(O2)} := \emptyset : t = 1, \ldots, H\}$

for $s \in RandPerm(\{1, \ldots, n\})$ **do** // *Consider retailers in a random order*
- $T_1 := \{t : s \in \pi_t^{(S1)}\}$ // *Get delivery schedules from parent solutions*
- $T_2 := \{t : s \in \pi_t^{(S2)}\}$
- **if** $URand([0, 1]) < 0.5$ **then** // *Swap with probability* $\frac{1}{2}$
  - $T_1 \leftrightarrow T_2$
- **for** $t = 1, \ldots, H$ **do** // *Try adding delivery schedules*
  - **if** $t \in T_1$ **then** // *from parent solutions to the offspring*
    - $\pi_t^{(O1)} := \pi_t^{(O1)} \cup \{s\}$
  - **if** $t \in T_2$ **then**
    - $\pi_t^{(O2)} := \pi_t^{(O2)} \cup \{s\}$
- **if** $noInfeasible$ **then**
  - **if** $\neg IsFeasible(O^{(1)})$ **then** // *If infeasible offspring not allowed*
    - $O^{(1)} := SLD(S^{(S1)}, s, 1)$ // *replace using the SLD heuristic*
  - **if** $\neg IsFeasible(O^{(2)})$ **then**
    - $O^{(2)} := SLD(S^{(S2)}, s, 1)$

Infeasible retailer removal mutation The infeasible retailer removal mutation applies N_{mut} changes to a solution $S = \{\pi_t : t = 1, \ldots, H\}$. Each change is made by randomly selecting one element from S which represents a retailer s being visited on a day d. The retailer s is removed from π_d. Note, that retailers for removal are selected with the uniform probability over the entire solution S so if a retailer is present in several routes the probability of selecting this retailer for removal increases.

2.5. Population Initialization

Population initialization is performed by first generating a base solution using the SLD heuristic and then mutating this base solution using the feasible date change mutation (FDCM) operator with $P_{mut} = 1.0$ and $\alpha_{mut} = 1.0$. The population initialization procedure is presented in Algorithm 5.

3. Evolutionary Algorithms

The evolutionary algorithms used in this paper are the classical Simple Genetic Algorithm (SGA) with:

Algorithm 3: The feasible date change mutation (FDCM).

Inputs:
- $S = \{\pi_t : t = 1, \ldots, H\}$ - Solution to mutate
- N_{mut} - Number of changes to make to S

Output:
- S' - Mutated solution if feasible modifications found or unchanged otherwise

$S' := S$
for $s \in RandPerm(\{1, \ldots, n\})$ **do** // *Consider retailers in a random order*
  $T := \{t : s \in \pi_t\}$ // *Get delivery days for retailer s*
  if $T = \emptyset$ **then**
    continue
  $d := URand(\{1, \ldots, |T|\})$ // *Select a date to move and the range of movement*
  if $d > 1$ **then**
    $t_0 := T[d-1] + 1$
  else
    $t_0 := 1$
  $t_{max} := \min(H, StockoutDate(S, s, t_0))$
  if $t_{max} = t_0$ **then** // *The date cannot be moved, or a stockout occurs*
    continue
  for $t^+ \in RandPerm(\{t_0, \ldots, t_{max}\} \setminus \{T[d]\})$ **do** // *Try moving the delivery*
    $S'' := S'$ // *to available dates in a random order*
    $\pi''_{T[d]} := \pi''_{T[d]} \setminus \{s\}$
    $\pi''_{t^+} := \pi''_{t^+} \cup \{s\}$
    $\forall t > t^+ : \pi''_t := \pi''_t \setminus \{s\}$
    $S'' := SLD(S'', s, t^+ + 1)$
    if $IsFeasible(S'')$ **then**
      $S' := S''$
      $N_{mut} := N_{mut} - 1$
      break
  if $N_{mut} = 0$ **then** // *Stop when N_{mut} changes were made*
    break
return S'

- Fitness calculated based on the solution costs normalized to [0, 1]. That is $fitness(S) = 1 - Norm(f(S), 0, 1)$, where the $Norm$ function normalizes the solution costs in the current population to [0, 1]. SGA tries to maximize $fitness(S)$ thereby minimizing the cost $f(S)$.
- Elitism mechanism, preserving the best solution found so far.
- Mating pool selection based on a binary tournament with respect to the fitness.
- The crossover operator described in Section 2.3 with the *noInfeasible* parameter set to *true* allowing only feasible solutions to be generated.
- Feasible date change and retailer addition/removal mutation operators described in Section 2.4.

and the Infeasibility Driven Evolutionary Algorithm (IDEA) [17] with:

Algorithm 4: The feasible retailer addition/removal mutation.

Inputs:
- $S = \{\pi_t : t = 1, \ldots, H\}$ - Solution to mutate
- N_{mut} - Number of changes to make to S

Output:
- S - Solution S mutated if feasible modifications found or unchanged otherwise

for $s \in RandPerm(\{1, \ldots, n\})$ **do** *// Consider retailers in a random order*
  $\mathcal{M} := \emptyset$
  for $d \in \{1, \ldots, H\}$ **do**
    $S' := S$
    if $s \in \pi_d$ **then**
      $\pi'_d := \pi'_d \setminus \{s\}$
    else
      $\pi'_d := \pi'_d \cup \{s\}$
    if $IsFeasible(S')$ **then**
      $\mathcal{M} := \mathcal{M} \cup \{S'\}$
  if $\mathcal{M} \neq \emptyset$ **then**
    $S := URand(\mathcal{M})$
    $N_{mut} := N_{mut} - 1$
  if $N_{mut} = 0$ **then** *// Stop when N_{mut} changes were made*
    break
return S

Algorithm 5: The population initialization procedure.

Inputs:
- N_{pop} - Number of solutions to generate
- P_{mut} - Mutation probability ($P_{mut} = 1.0$ by default)
- α_{mut} - Mutation intensity ($\alpha_{mut} = 1.0$ by default)

Output:
- $\mathcal{P}$ - A population with N_{pop} solutions

$S_0 := \emptyset$ *// Generate the base solution*
for $s \in RandPerm(\{1, \ldots, n\})$ **do**
  $S_0 := SLD(S_0, s, 1)$

$\mathcal{P} := \emptyset$
for $i \in \{1, \ldots, N_{pop}\}$ **do** *// Population contains N_{pop} copies of the base solution S_0*
  $\mathcal{P} := \mathcal{P} \cup \{FDCM(S_0, P_{mut}, \alpha_{mut})\}$ *// mutated using the feasible date change mutation (FDCM) operator*
return $\mathcal{P}$

- Mating pool selection based on a binary tournament with respect to the non-dominated front number and crowding distance.
- The crossover operator described in Section 2.3 with the *noInfeasible* parameter set to $false$ allowing infeasible solutions to be generated.
- Infeasible date change, retailer addition and retailer removal mutation operators described in Section 2.4.

Note, that IDEA is a multi-objective optimization algorithm, but in this paper it is used to solve a single-objective optimization problem. The second objective used in IDEA is the constraint violation measure calculated from the constraints $g_i(S), i = 1, 2, 3$ (Section 2.1). IDEA was used in this paper as a baseline with which to compare SGA with operators preserving feasibility, because IDEA aims at handling infeasibility, but allowing the operators to produce infeasible solutions and using the selective pressure to obtain feasible ones. Both algorithms are hybridized with the Concorde TSP Solver [20] to which the optimization of the delivery routes is delegated. Because the algorithms use more than one mutation operator a mechanism for autoadaptation of operator probability based on success rates [13] is used for deciding which operator to apply when solutions are mutated. Both algorithms start from populations generated using the population initialization procedure described in Section 2.5. This allows SGA to keep the population feasible using feasible genetic operators and provides IDEA with a feasible part of the population. Infeasible solutions in IDEA are obtained in subsequent generations by applying infeasible genetic operators.

The algorithms are parameterized by setting the population size N_{pop}, operator probabilities P_{cross} and P_{mut}, and mutation intensity α_{mut}. These parameters were tuned using the grid-search approach with the tested values: $N_{pop} \in \{$ 50, 100, 200, 500 $\}$, $P_{cross} \in \{$ 0.2, 0.4, 0.6, 0.8, 1.0 $\}$, $P_{mut} \in \{$ 0.02, 0.04, 0.06, 0.08, 0.10, 0.20, 0.50, 0.75, 1.00 $\}$ and $\alpha_{mut} \in \{$ 0.1, 0.2, 0.5, 1.0 $\}$. The selected values of the parameters were $N_{pop} = 50$, $P_{cross} = 0.2$ for SGA and $N_{pop} = 500$, $P_{cross} = 0.4$ for IDEA. For both algorithms $P_{mut} = 0.75$ and $\alpha_{mut} = 0.1$ were found to be the best ones. The infeasible population fraction for IDEA was set to $\alpha_{infeasible} = 0.2$ as in the original paper [17].

4. Experiments and Results

In the experiments the two algorithms presented in Section 3 were compared thereby providing a comparison between an algorithm with feasibility-preserving operators and an algorithm that allows the operators to produce infeasible solutions and aims at using selective pressure to obtain feasible solutions. The comparison was performed on the low-cost instances with $H = 3$ presented in [3]. The maximum number of solution evaluations $max_{FE} = 10000$ was used as the stopping criterion and for each problem instance 10 repetitions of the test were performed.

4.1. Comparison of best results

Figure 1 presents the relative difference between the best results attained by IDEA and SGA. The bars present the average of the value $\frac{f^*_{IDEA}(K) - f^*_{SGA}(K)}{f^*_{SGA}(K)}$, calculated over five instances K of IRP with the given size n. In this equation $f^*(K)$ denotes the best result attained by each of the algorithms on the problem instance K. Clearly, the evolutionary algorithm using feasibility-preserving operators has an advantage over IDEA especially for medium-size instances ($n = 20, \ldots, 30$). Table 1 compares numerical values of the best solutions found by the two evolutionary algorithms. The "Comp." column shows a comparison of the results produced by SGA using feasibility-preserving operators with the results produced by IDEA. In this column the '=' sign represents equally good results and the '+' sign represents those instances when the results produced by SGA were superior to those produced by IDEA, that is, when lower cost was attained by SGA. Note, that for none of the tested instances IDEA attained better results than SGA (hence, no '-' in the "Comp." column).

Results presented in Table 1 clearly show that utilizing feasibility-preserving operators allows attaining better results and IDEA using the selective pressure towards feasible solutions is not able to produce competitive results.

4.2. Analysis of operator success rates

In this paper a mechanism for autoadaptation of operator probability based on success rates [13] is used for deciding which operator to apply when solutions are mutated. Success rates are defined as the ratio of the number of improvements b_i in the criteria used for evaluating solutions to the number of times n_i a given mutation operator was applied and changed the solution: $s_i = b_i/n_i$. The number n_i is affected by the mutation probability P_{mut}, the mutation being selected or not by the autoadaptation mechanism and the fact that feasible operators may be unable to find a modification of the original solution that preserves the feasibility and they leave the initial solution unchanged. For SGA the b_i is increased by 1 if the mutated solution has lower total cost and for IDEA the b_i is increased by 1 when solution cost is decreased or when the constraint violation measure is decreased. Therefore, the theoretical maximum

Table 1: The best solutions found by the evolutionary algorithms.

Instance (K)	SGA $f^*_{SGA}(K)$	IDEA $f^*_{IDEA}(K)$	Comp.
abs1n5	1281.68	1281.68	=
abs2n5	1176.63	1176.63	=
abs3n5	2020.65	2020.65	=
abs4n5	1449.43	1449.43	=
abs5n5	1165.40	1165.40	=
abs1n10	2167.37	2167.37	=
abs2n10	2510.13	2510.13	=
abs3n10	2099.68	2099.68	=
abs4n10	2188.01	2535.73	+
abs5n10	2178.15	2178.15	=
abs1n15	2271.68	2380.88	+
abs2n15	2506.21	2673.84	+
abs3n15	2841.06	3162.73	+
abs4n15	2540.77	3294.42	+
abs5n15	2453.50	3011.59	+
abs1n20	2874.56	3848.83	+
abs2n20	2867.75	3408.01	+
abs3n20	3102.47	3713.05	+
abs4n20	3572.61	4427.20	+
abs5n20	3393.90	4113.09	+
abs1n25	3640.98	4284.99	+
abs2n25	3557.53	4511.36	+
abs3n25	3733.75	4647.06	+
abs4n25	3438.58	4747.81	+
abs5n25	3766.45	4288.21	+

Instance (K)	SGA $f^*_{SGA}(K)$	IDEA $f^*_{IDEA}(K)$	Comp.
abs1n30	4446.07	4878.97	+
abs2n30	4143.44	4645.87	+
abs3n30	4204.12	4881.15	+
abs4n30	3785.60	4701.47	+
abs5n30	3483.37	4476.47	+
abs1n35	4694.30	4994.55	+
abs2n35	4786.66	5116.12	+
abs3n35	5335.28	5375.80	+
abs4n35	4444.84	4953.90	+
abs5n35	4256.25	5322.19	+
abs1n40	5044.20	5491.36	+
abs2n40	5038.71	5461.69	+
abs3n40	4976.14	5554.43	+
abs4n40	4857.85	5651.99	+
abs5n40	4762.78	5379.00	+
abs1n45	5423.70	5886.08	+
abs2n45	5444.81	5585.80	+
abs3n45	5234.45	6063.41	+
abs4n45	5417.41	5763.73	+
abs5n45	4679.25	5537.58	+
abs1n50	5581.44	6072.69	+
abs2n50	6301.82	6453.06	+
abs3n50	5997.67	6383.36	+
abs4n50	6228.11	6571.09	+
abs5n50	5751.47	6264.02	+

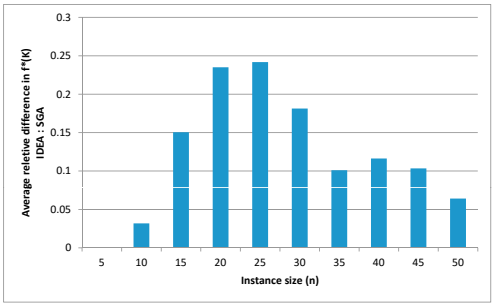

Fig. 1: The relative difference between the best results attained by IDEA and SGA (Section 4.1) averaged over five instances with the given size n for each bar. The larger the value, the bigger the advantage of SGA using feasibility-preserving operators over IDEA.

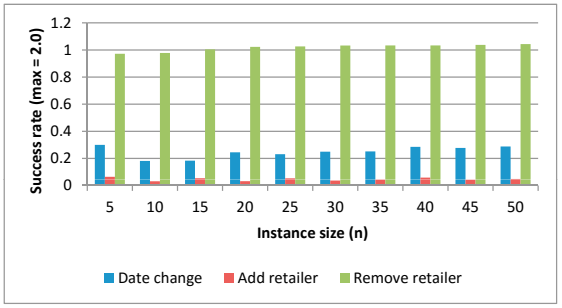

Fig. 2: Success rates of the three mutation operators used in IDEA. Note, that the maximum possible value of the success rates is 2.0 for this algorithm, because improving each of the two criteria counts as a 'success'.

of the success rates is $max(s_i) = 1$ in SGA and $max(s_i) = 2$ in IDEA (in an unrealistic scenario in which *every single application* of the i-th operator decreased *both* the cost *and* the constraint violation measure).

Figure 2 presents success rates for the operators used in IDEA. Obviously, the infeasible retailer removal mutation is very likely to improve the cost criterion, but only sometimes it is able to also improve the feasibility of the solution, namely when the retailer s removed from the route π_t does not really have to be visited on that date. The other two operators are less likely to achieve at least one success, because they both may, or may not, decrease the cost and diminish constraint violation severity. In the case of SGA the feasible date change mutation operator achieved success rates between 0.34 and 0.48 for all instance sizes, however, no plot is shown because of space limitation.

5. Conclusion

In this paper an evolutionary algorithm with feasibility-preserving operators for the inventory routing problem (IRP) was compared to the Infeasibility Driven Evolutionary Algorithm (IDEA) which allows the operators to generate infeasible solutions and tries to attain feasible ones using a split population and the selective pressure towards decreasing the constraint violation measure. Costs produced by an evolutionary algorithm with feasibility-preserving operators are lower than those attained by IDEA for all instances with $n \geq 15$. Presented results show that designing good feasibility-preserving operators is very important in the context of the considered problem. Clearly, applying the selective pressure towards feasible solutions is not enough to solve the IRP effectively. Presented work motivates further study on feasibility-preserving genetic operators for the inventory routing problem.

Acknowledgment

This work was supported by the Polish National Science Centre under grant no. 2015/19/D/HS4/02574. Calculations have been carried out using resources provided by Wroclaw Centre for Networking and Supercomputing (http://wcss.pl), grant No. 407.

References

[1] Alinaghian, M., Tirkolaee, E.B., Dezaki, Z.K., Hejazi, S.R., Ding, W., 2020. An augmented tabu search algorithm for the green inventory-routing problem with time windows. Swarm and Evolutionary Computation 60, 100802.
[2] Alkaabneh, F., Diabat, A., Gao, H.O., 2020. Benders decomposition for the inventory vehicle routing problem with perishable products and environmental costs. Computers & Operations Research 113, 104751.
[3] Archetti, C., Bertazzi, L., Laporte, G., Speranza, M.G., 2007. A branch-and-cut algorithm for a vendor-managed inventory-routing problem. Transportation Science 41, 382–391.
[4] Azadeh, A., Elahi, S., Farahani, M.H., Nasirian, B., 2017. A genetic algorithm-taguchi based approach to inventory routing problem of a single perishable product with transshipment. Computers & Industrial Engineering 104, 124–133.
[5] Bertazzi, L., Coelho, L.C., De Maio, A., Lagana, D., 2019. A matheuristic algorithm for the multi-depot inventory routing problem. Transportation Research Part E: Logistics and Transportation Review 122, 524–544.
[6] Coelho, L.C., De Maio, A., Lagana, D., 2020. A variable mip neighborhood descent for the multi-attribute inventory routing problem. Transportation Research Part E: Logistics and Transportation Review 144, 102137.
[7] Dabiri, N., J. Tarokh, M., Alinaghian, M., 2017. New mathematical model for the bi-objective inventory routing problem with a step cost function: a multi-objective particle swarm optimization solution approach. Applied Mathematical Modelling 49, 302–318.
[8] De, A., Kumar, S.K., Gunasekaran, A., Tiwari, M.K., 2017. Sustainable maritime inventory routing problem with time window constraints. Engineering Applications of Artificial Intelligence 61, 77–95.
[9] Federgruen, A., Zipkin, P., 1984. A combined vehicle routing and inventory allocation problem. Operations Research 32, 1019–1037.
[10] Gruler, A., Panadero, J., de Armas, J., Moreno Prez, J.A., Juan, A.A., 2018. Combining variable neighborhood search with simulation for the inventory routing problem with stochastic demands and stock-outs. Computers & Industrial Engineering 123, 278–288.
[11] Lefever, W., Aghezzaf, E.H., Hadj-Hamou, K., Penz, B., 2018. Analysis of an improved branch-and-cut formulation for the inventory-routing problem with transshipment. Computers & Operations Research 98, 137–148.
[12] Liu, S.C., Lee, W.T., 2011. A heuristic method for the inventory routing problem with time windows. Expert Systems with Applications 38, 13223–13231.
[13] Michalak, K., 2015. The Sim-EA algorithm with operator autoadaptation for the multiobjective firefighter problem, in: Ochoa, G., Chicano, F. (Eds.), Evolutionary Computation in Combinatorial Optimization. Springer, Cham. volume 9026 of *LNCS*, pp. 184–196.
[14] Mjirda, A., Jarboui, B., Macedo, R., Hanafi, S., Mladenovi, N., 2014. A two phase variable neighborhood search for the multi-product inventory routing problem. Computers & Operations Research 52, 291–299.
[15] Popovi, D., Vidovi, M., Radivojevi, G., 2012. Variable neighborhood search heuristic for the inventory routing problem in fuel delivery. Expert Systems with Applications 39, 13390–13398.
[16] Rau, H., Budiman, S.D., Widyadana, G.A., 2018. Optimization of the multi-objective green cyclical inventory routing problem using discrete multi-swarm pso method. Transportation Research Part E: Logistics and Transportation Review 120, 51–75.
[17] Ray, T., Singh, H.K., Isaacs, A., Smith, W., 2009. Infeasibility driven evolutionary algorithm for constrained optimization, in: Mezura-Montes, E. (Ed.), Constraint-Handling in Evolutionary Optimization. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 145–165.
[18] Reinelt, G., 1991. TSPLIBA Traveling Salesman Problem Library. INFORMS Journal on Computing 3, 376–384.
[19] Shaabani, H., Kamalabadi, I.N., 2016. An efficient population-based simulated annealing algorithm for the multi-product multi-retailer perishable inventory routing problem. Computers & Industrial Engineering 99, 189–201.
[20] William Cook, 2020. Concorde TSP solver. http://www.math.uwaterloo.ca/tsp/concorde.html. Online: accessed 2021.02.02.