1 Introduction

Although evolutionary algorithms have often been used to address VRPs and VRPTWs [1], applications to other routing problems such as period routing and inventory routing are less common. In this paper, a versatile hybrid evolutionary algorithm is developed that can be used for cyclic inventory routing and a variety of capacitated and periodic vehicle routing problems. First, a random insertion heuristic and three well-known local search (LS) operators are used to generate a pool of high quality solutions. An evolutionary algorithm (EA) then manages solution quality and diversity in the pool by using disruptive crossover and mutation operators combined with the same three local search operators. The versatility and performance of the approach is illustrated on a set of benchmark problems on the cyclic inventory routing problem (CIRP), the periodic vehicle routing problem (PVRP) and the classical capacitated vehicle routing problem (CVRP).

The remainder of this paper is structured as follows. The three routing problems are briefly presented in Section 2. Section 3 presents the evolutionary algorithm and Section 4 describes the specifics of the local search procedures used. Computational experiments are reported in Section 5, followed by conclusions and suggestions for further developments in Section 6.
2 Description of the routing problems

2.1 Cyclic inventory routing

In the cyclic inventory routing problem, a set of customers $S$ with constant demand rates $d_i (i \in S)$ and limited storage capacities $\kappa_i (i \in S)$ have to be replenished from a single depot. Because of the constant demand rates, a cyclic planning policy is adopted and the minimization of the total cost rate (cost per period), is the objective. For this total cost rate, we consider the following four components: (1) a fixed vehicle dispatching cost $\phi$ per route, (2) a transportation cost per kilometer $\delta$, (3) a drop-off cost $\varphi_i$ per delivery at customer $i \in S$ and (4) an inventory holding cost $\eta_i$ per unit per period at customer $i \in S$.

The following interrelated subproblems then have to be solved simultaneously:

- Designing routes to minimize dispatching and transportation costs.
- Determining cycle times and delivery quantities for the customers to balance transportation and drop-off costs with holding costs.

In the long run, the number of trucks can be adjusted and fixed vehicle costs need to be taken into account. This further adds to complexity because it induces another subproblem to be solved [2, 3]:

- Assigning routes to vehicles to minimize the vehicle fleet.

However, the fleet sizing component of the cyclic inventory routing problem is not considered in this paper.

For a cyclic route $r$ replenishing a given subset of customers $S_r$, the following needs to be done. First, the maximal cycle time of the route is determined by looking at the vehicle’s loading capacity $\kappa$ and customers’ storage capacities $\kappa_i$:

$$T_{\text{max},r} = \min \left( \frac{\kappa}{\sum_{i \in S_r} d_i}, \min_{i \in S_r} \left( \frac{\kappa_i}{d_i} \right) \right)$$

Next, the customers in the route should be sequenced such that the travel distance of the route $D_r$ is minimized. If the duration of the route is longer than a given limit (typically eight hours), the route is infeasible. Finally, the cycle time of the route, $T_r$, is chosen such that the route’s cost rate $C_r$ is minimized.

$$C_r = \frac{1}{T_r} \left( \phi + \delta D_r + \sum_{i \in S_r} \varphi_i \right) + T_r \left( \sum_{i \in S_r} \frac{\eta_i d_i}{2} \right)$$

Determining the optimal cycle time $T_r$ is nothing more than evaluating a capacitated economic order quantity (EOQ) that balances the transportation and inventory holding costs.
2.2 Periodic vehicle routing

In the PVRP, a set of customers $S$ requesting given quantities $q_i$ ($i \in S$) needs to be serviced within a given period of $T$ days at minimal transportation costs. This problem can be considered as a cyclic inventory routing problem in which (i) the per unit per day holding cost rate is zero for all customers, $\eta_i = 0$ ($i \in S$); (ii) customer daily demand rates are equal to the requested quantity divided by the time horizon, $d_i = q_i/T$ ($i \in S$); and (iii) customer storage capacities are equal to the requested quantities, $\kappa_i = q_i$ ($i \in S$). As a result, all route cycle times $T_r$ will be equal to the given time horizon $T$.

2.3 Capacitated vehicle routing

The classical capacitated vehicle routing problem is a PVRP with a time horizon of a single day ($T = 1$). So, it can be considered as a cyclic inventory routing problem in which (i) the per unit per day holding cost rate is zero for all customers, $\eta_i = 0$ ($i \in S$); and (ii) daily demand rates and customer storage capacities are equal to the requested quantities, $d_i = \kappa_i = q_i$ ($i \in S$). All route cycle times $T_r$ will then be equal to one day. To limit the number of routes, an artificially high vehicle dispatching cost $\phi$ can be quoted.

3 The evolutionary algorithm

The purpose of the evolutionary algorithm presented in this section is to manage the solution quality and diversity in a pool of solutions. It does so with the use of crossover and mutation operators that create new solutions by partially destroying existing solutions and subsequently improving them with various local search operators.

The size of the solution pool is set to 30 solutions. Initial solutions are constructed using a random sequential insertion heuristic, followed by the local search procedure explained below. In the insertion heuristic, customers are randomly inserted into the solution one by one, each into a route (and a position in that route) that least increases the total cost rate, or into a new route if no feasible insertion is possible. The subsequent local search phase is explained below in Section 4. To safeguard diversity in the pool, only unique individuals are allowed, so the mutation operator (see below) is applied if a solution is constructed that is already present in the population, again followed by the local search phase.

Per generation, 30 crossovers are performed. This means that after crossover, 60 solutions are available (30 ‘parents’ and 30 ‘children’). To maintain a population of unique individuals, mutation is applied to a child if it is the same as a solution already present in the population or in the set of newly generated children. The 30 best of these 60 solutions will constitute the next generation of the population.

After a number of generations, it may happen that none of the 30 children being generated during crossover is better than any of the 30 parents, which indicates that the population is converging. Therefore, the following stopping criterion is adopted: if no child enters the population for 25 consecutive generations, the algorithm stops and the best solution is returned.
3.1 Fitness

To increase the chances of the best solutions being selected for crossover, the fitness calculation
is not based on the absolute objective function values, but on values relative to the best
found solution instead. These values are calculated as \( \text{obj}_\text{value} - \alpha \cdot \text{best}_\text{obj}_\text{value} \), with
the parameter \( \alpha \) smaller than 1. With \( \alpha \) equal to zero, absolute values are used and there
may not be large difference in the fitness of the best and the worst solutions. Increasing \( \alpha \)
increases the fitness of the better solutions, while decreasing that of the worse solutions in the
population. However, when \( \alpha \) approaches 1, the fitness of the best solution goes to 100%, while
all other fitnesses go to zero. This means that, for high values of \( \alpha \), the probability is very
high that the same solutions are always selected for crossover, which would lead to premature
convergence of the population. The value of the parameter \( \alpha \) can therefore be varied from
generation to generation (possibly in a self-adaptive manner) to switch between intensification
and diversification of the search. However, in the current implementation, the value of \( \alpha \) has
been fixed to 95% after some computational validations.

3.2 Crossover

Since powerful local search operators are available for optimizing a given solution, the main
task of the crossover operator is to perturb parent solutions while preserving their common
arcs. Two different parents are selected from the population using the well-known roulette
wheel mechanism based on the fitness distribution. During the crossover, common arcs of the
two parent solutions are first identified and all customers at the edges of these common arcs are
being marked. Then, two children are constructed as follows. Initially, the first child is a copy
of the first parent, and the second child is a copy of the second parent. Then, all unmarked
customers (i.e. not at the edge of a common arc) that are before the crossover point of the
second parent are removed from the first child and vice versa. See Figure 1 for an illustrative
example.

<table>
<thead>
<tr>
<th>Parent 1:</th>
<th>Parent 2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 0 6 7 8 9 0</td>
<td>0 7 9 1 6 0 4 3 8 2 5 0</td>
</tr>
</tbody>
</table>

Common arcs: (3, 4), (5, 0) and (0, 6) ⇒ Marked customers: 3, 4, 5, and 6

<table>
<thead>
<tr>
<th>Child 1:</th>
<th>Child 2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent 1 without customers 7, 9 and 1.</td>
<td>Parent 2 without customers 1 and 2.</td>
</tr>
<tr>
<td>0 2 3 4 5 0 6 8 0 7 9 1</td>
<td>0 7 9 6 0 4 3 8 5 0 1 2</td>
</tr>
</tbody>
</table>

(Customer 3 is kept since it was marked.)

Figure 1: Illustration of the crossover.

The crossover point is after the \( n \)'th customer in the solution, with \( n \) randomly generated
as follows: \( n = 1 + \lfloor r^2 \cdot (|C| - 1) \rfloor, r \in [0, 1[. \) Note that the randomly generated number \( r \) is squared, such that a small number of customers is removed from the children more often than a large number of customers.

When the unmarked customers have been removed from the children, a mutation in which some additional customers are removed (see below) is applied to both children with a probability of 1%. After this, the children are optimized as follows. First, a partial solution is constructed with all removed customers (using the same sequential insertion heuristic as for the initial solutions) and then local search is applied to the whole solution (i.e. the remaining part of the parent solution plus the newly constructed partial solution with removed customers). Of both children, the better one is eventually returned as the result of the crossover.

### 3.3 Mutation

Similar to the crossover operator, the mutation operator removes a random number \( n \) of customers from the solution, with \( n = 1 + \lfloor r^2 \cdot (|C| - 1) \rfloor, r \in [0, 1[. \) There are two possibilities for the mutation: either the customers to be removed are randomly selected, or a ratio is attributed to the customers, and those with the highest ratios are removed. This ratio must therefore reflect how well the solution is for a particular customer. The idea is that the situation for customers with high ratios improves after removing them and subsequently reinserting them, followed by the local search operators.

In the CIRP, the objective is to find the best balance between transportation and holding cost rates. The customer ratio therefore compares the customer’s holding cost rate \( H \) to the incremental transportation cost rate \( TR \) for reaching the customer (approximated by the average of the transportation costs of both arcs incident to the customer). If \( H \) is much bigger than \( TR \), or vice versa, this means that no good balance is obtained. The ratio that is therefore assigned to a customer is \( \max \left( \frac{TR}{H}, \frac{H}{TR} \right) \). In the PVRP and CVRP, there is no holding cost component \( (H = 0) \), so the ratio there is given by \( TR \).

### 4 The local search procedure

The local search procedure that is applied after the construction phase and throughout the genetic algorithm is actually a variable neighbourhood search (VNS), consisting of various well-known local search operators that have been generalized for the cyclic inventory routing problem. Each of these local search operators is applied in a first-accept manner, meaning that as soon as an improvement is identified, it is immediately implemented in the solution and the local search operator is restarted.

The local search operators that are used are the following:

1. Relocate: remove a customer and try inserting it into another position (either in the same route, in a different route or in a separate new route).

2. Exchange: switch the positions of two customers (either from the same route or from two different routes).
3. 2-opt: remove two arcs (either from the same route or from two different routes) and replace them by two other arcs, such that all routes are closed again and no subtours are created.

5 Computational experiments

In this section, some preliminary computational results are presented. However, further computational testing is currently being performed, and more results and a more elaborate discussion of these results will be available shortly.

5.1 The cyclic inventory routing problem

First, the EA is applied to the cyclic inventory routing problem benchmark instances of Raa and Aghezzaf (2007) [3]. It turns out that the EA considerably outperforms the multi-start heuristic presented there for almost all instances (see Table 1).

The same benchmark instances were also solved with a multistart two-phase heuristic that uses the same construction and improvement heuristics as those adopted for the EA, and that is run with computation times as long as those of the EA. The results, also in Table 1, show that the EA is always better, which proves the value of the proposed crossover and mutation operators.

Table 1: CIRP computational results.

<table>
<thead>
<tr>
<th>CCAP</th>
<th>HC</th>
<th>NR</th>
<th>[3]</th>
<th>EA-LS</th>
<th>Improvement</th>
<th>Multistart</th>
<th>EA-LS</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>80c</td>
<td>[30,70]</td>
<td>1297.03</td>
<td>1247.97</td>
<td>3.78%</td>
<td>1248.06</td>
<td>1247.97</td>
<td>0.01%</td>
</tr>
<tr>
<td>No</td>
<td>80c</td>
<td>[80,120]</td>
<td>2822.56</td>
<td>2622.96</td>
<td>7.07%</td>
<td>2642.95</td>
<td>2622.96</td>
<td>0.76%</td>
</tr>
<tr>
<td>No</td>
<td>8c</td>
<td>[30,70]</td>
<td>714.77</td>
<td>716.65</td>
<td>−0.26%</td>
<td>716.73</td>
<td>716.65</td>
<td>0.01%</td>
</tr>
<tr>
<td>No</td>
<td>8c</td>
<td>[80,120]</td>
<td>1562.11</td>
<td>1553.53</td>
<td>0.55%</td>
<td>1559.49</td>
<td>1553.53</td>
<td>0.38%</td>
</tr>
<tr>
<td>Yes</td>
<td>80c</td>
<td>[30,70]</td>
<td>1632.45</td>
<td>1564.37</td>
<td>4.17%</td>
<td>1567.55</td>
<td>1564.37</td>
<td>0.20%</td>
</tr>
<tr>
<td>Yes</td>
<td>80c</td>
<td>[80,120]</td>
<td>3335.86</td>
<td>3079.35</td>
<td>7.69%</td>
<td>3121.19</td>
<td>3079.35</td>
<td>1.34%</td>
</tr>
<tr>
<td>Yes</td>
<td>8c</td>
<td>[30,70]</td>
<td>1324.01</td>
<td>1267.93</td>
<td>4.24%</td>
<td>1270.58</td>
<td>1267.93</td>
<td>0.21%</td>
</tr>
<tr>
<td>Yes</td>
<td>8c</td>
<td>[80,120]</td>
<td>2633.03</td>
<td>2430.30</td>
<td>7.70%</td>
<td>2484.64</td>
<td>2430.30</td>
<td>2.19%</td>
</tr>
</tbody>
</table>

5.2 The periodic vehicle routing problem

Next, the EA is applied to a limited set of PVRP benchmark problems. In our solution approach, each customer is always replenished by the same route, and the cycle time of the customer is always equal to the cycle time of the route serving the customer. In many PVRP instances however, serving customers in different routes on different days offers much potential for improvement. E.g., if customer A must be visited every day, and customers B and C only every two days, then the best solution probably is to visit A and B in a single route on odd days, and A and C in another single route on even days. Customer A is then served by more
than one route. Since this is not possible in our solution approach (yet), only PVRP instances
are being considered in which all customers have the same cycle time.

The results for these instances are shown in Table 2. Note that the EA finds four best
known solutions, among two of which are new best solutions.

Table 2: PVRP computational results.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Best-known</th>
<th>EA-LS</th>
<th>Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>p01-2</td>
<td>524.61</td>
<td>524.61</td>
<td>0.00%</td>
</tr>
<tr>
<td>p03-5</td>
<td>524.61</td>
<td>524.61</td>
<td>0.00%</td>
</tr>
<tr>
<td>p04-2</td>
<td>835.43</td>
<td>835.26</td>
<td>−0.02%</td>
</tr>
<tr>
<td>p06-10</td>
<td>836.37</td>
<td>835.26</td>
<td>−0.13%</td>
</tr>
<tr>
<td>p07-2</td>
<td>826.14</td>
<td>827.39</td>
<td>0.15%</td>
</tr>
<tr>
<td>p09-8</td>
<td>826.14</td>
<td>828.77</td>
<td>0.32%</td>
</tr>
</tbody>
</table>

5.3 The capacitated vehicle routing problem

Finally, the EA is applied to a set of benchmark instances for the CVRP. The preliminary
results, shown in Table 3, look promising although they are not always excellent. We believe
that some further work on the EA, including the self-adaptive tuning of the parameter α, can
still significantly improve our results. However, we can already conclude that the EA which
was designed for more general routing problems is already capable of delivering competitive
results for the very thoroughly studied CVRP.

Table 3: CVRP computational results.

<table>
<thead>
<tr>
<th>Instance</th>
<th>EA-LS</th>
<th>Gap</th>
<th>Instance</th>
<th>EA-LS</th>
<th>Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>tai75a</td>
<td>1618.35</td>
<td>0.00%</td>
<td>gol-n241-k22</td>
<td>708.75</td>
<td>0.14%</td>
</tr>
<tr>
<td>tai75b</td>
<td>1355.92</td>
<td>0.84%</td>
<td>gol-n253-k27</td>
<td>864.08</td>
<td>0.58%</td>
</tr>
<tr>
<td>tai75c</td>
<td>1291.00</td>
<td>0.00%</td>
<td>gol-n256-k14</td>
<td>587.80</td>
<td>0.76%</td>
</tr>
<tr>
<td>tai75d</td>
<td>1365.42</td>
<td>0.00%</td>
<td>gol-n301-k28</td>
<td>1009.59</td>
<td>1.09%</td>
</tr>
<tr>
<td>tai100a</td>
<td>2047.90</td>
<td>0.32%</td>
<td>gol-n321-k30</td>
<td>1096.07</td>
<td>1.36%</td>
</tr>
<tr>
<td>tai100b</td>
<td>1940.61</td>
<td>0.04%</td>
<td>gol-n324-k16</td>
<td>748.93</td>
<td>0.93%</td>
</tr>
<tr>
<td>tai100c</td>
<td>1415.28</td>
<td>0.65%</td>
<td>gol-n361-k33</td>
<td>1388.59</td>
<td>1.59%</td>
</tr>
<tr>
<td>tai100d</td>
<td>1594.46</td>
<td>0.84%</td>
<td>gol-n397-k34</td>
<td>1373.04</td>
<td>2.07%</td>
</tr>
<tr>
<td>tai150a</td>
<td>3058.83</td>
<td>0.12%</td>
<td>gol-n400-k18</td>
<td>932.36</td>
<td>1.51%</td>
</tr>
<tr>
<td>tai150b</td>
<td>2732.51</td>
<td>2.86%</td>
<td>gol-n421-k41</td>
<td>1874.41</td>
<td>2.92%</td>
</tr>
<tr>
<td>tai150c</td>
<td>2391.62</td>
<td>2.13%</td>
<td>gol-n481-k38</td>
<td>1646.23</td>
<td>1.45%</td>
</tr>
<tr>
<td>tai150d</td>
<td>2669.31</td>
<td>0.90%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tai385</td>
<td>25512.7</td>
<td>4.43%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6 Conclusion

This paper presents a hybrid evolutionary algorithm (EA) for generalized routing problems. The algorithm combines highly disruptive crossover and mutation operators with some powerful local search (LS) operators. Preliminary results on benchmark instances of CIRP, PVRP and CVRP show the versatility and effectiveness of both the EA and the LS components.

In the further development of this solution approach, two main ideas will be investigated:

- Allowing customers to be served by different routes, which offers a savings potential for periodic routing with multiple delivery frequencies.
- Tuning of the different parameters of the EA, such as the mutation probability, the parameter $\alpha$ used for fitness calculations, the population size, . . .

References

