An evolutionary approach to time-constrained routing problems

C.H.M. van Kemenade and J.N. Kok

Computer Science/Department of Software Technology

CS-R9547 1995

Report CS-R9547
ISSN 0169-118X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

An Evolutionary Approach to Time Constrained Routing Problems

C.H.M. van Kemenade

CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
kemenade@cwi.nl

J.N. Kok

Leiden University
Department of Computer Science
P.O. Box 9512, 2300 RA, Leiden, The Netherlands
joost@wi.leidenuniv.nl

Abstract

Routing problems are an important class of planning problems. Usually there are many different constraints and optimization criteria involved, and it is difficult to find general methods for solving routing problems. We propose an evolutionary solver for such planning problems. An instance of this solver has been tested on a specific routing problem with time constraints. The performance of this evolutionary solver is compared to a biased random solver and a biased hillclimber solver. Results show that the evolutionary solver performs significantly better than the other two solvers.

AMS Subject Classification (1991): 68T20
CR Subject Classification (1991): G.1.7, I.2.8, J.m
Keywords & Phrases: evolutionary computation, genetic algorithms, constrained optimization, vehicle routing
Note: Paper was presented at the Nordic Workshop on Genetic Algorithms, Vaasa 1995

1. Introduction

Evolutionary Algorithms EAs have been applied successfully to difficult numerical and discrete optimization problems. In order to solve a problem the EA uses a population of individuals, where each individual represents a potential solution to the given problem. Many theoretical and experimental results exist for individuals represented by vectors of floats, bitstrings and order based representations. There are many problems where these representations are appropriate, but there are also numerous problems where it is difficult to find a mapping between solutions and these simple representations. In this paper we use a different kind of representation, and we discuss the various issues involved.

Because there are so many different planning problems, it is important to find general methods which can be applied to a wide variety of planning problems. EAs have shown to be robust methods for solving a wide range of different optimization tasks with different types of constraints. EAs do not require much problem specific knowledge, because they only need some kind of measure to determine the quality of potential solution. On the other hand, if problem specific knowledge is available, then the method allows the usage of this knowledge by the incorporation of heuristics, problem specific representations and problem specific operators. In this way EAs try to get the best of both worlds. Another interesting property of EAs is that these methods perform an iterated improvement procedure on their population. There is always a solution available, and if there still is some time left then the EA tries to improve on its current best solution. This way interactive decision making becomes feasible. For these reasons we think that EAs are interesting options for handling planning problems.

In this paper we carry out a case study to show that a planning tool based on evolutionary algorithms is useful for a specific problem involving time constraints. This problem will be a routing problem. In routing problems optimal routes have to be obtained for a fleet of moving objects. There is a wide variety of routing problems with different constraints and different objectives. An example of a routing problem is the Traveling Salesmen Problem (TSP). In a TSP one tries to find the shortest tour to visit a given set of locations. The TSP is known to belong to the class of NP-complete problems. Even more difficult problems can arise if there are additional constraints which have to be satisfied. There can be many different constraints, such as time-constraints, capacity constraints, compound trucks and priority constraints. There can also be different objectives such as the minimization of the traveled distance, the minimization of delay until service, the optimization of the spread of vehicles over the area, and the avoidance of traveling during rush-hours. All combinations of constraints and goals give rise to different planning problems.

The rest of the paper is structured as follows. In section 2 we introduce the routing problem with time-constraints, in section 3 we show how an EA can be applied to this constrained routing problem, section 4 shows the experimental setup used to determine the strength of the method and the actual results, and finally we draw some conclusions and give some directions for further research.

2. The Routing Problem

In this section we define the routing problem which we will study in the rest of the paper.

The problem is to find a Just In Time (JIT) delivery schedule for a number of customers served by a single distribution center. Each customer has a limited stock capacity, and is assumed to consume the delivered cargo at a known rate. The limited stock capacity combined with the consumption rates can be translated in a kind of time constraints, that give the latest time a delivery can take place without the customer running out of stock.

In this problem we are given a set of customers C, a set of trucks T, and a distance matrix D. Each customer c is a tuple

$$c = (q_0, f_c, cap),$$

where q_0 is the initial stock the customer has a time $t = 0$, f_c is the function describing the instantanious consumption as a function of time, and *cap* is the total available stock capacity. A truck *tr* is represented by

$$tr = (cap),$$

where *cap* is the capacity of the truck. At time $t = 0$ all trucks are assumed to be completely filled and located at the distribution center. The distribution center is assumed to have enough production capacity to serve all customers. A visit v is represented by the tuple

$$v = (t, q)$$

where t is the time of visit v, and q is the quantity of cargo being delivered during visit v.

The distance matrix D contains the travel distances between locations, and we assume that the travel time is independent of the vehicle and of rush-hours. This limitation is for convenience and does not correspond to an actual limitation of the method.

The goal is to find a routing schedule, such that no customer runs out of stock, and such that the total number of trucks is minimized.

A schedule for a truck is represented by a list of customers, which have to be visited in sequence. For convenience the distribution center is assigned the label 0, so a return to the distribution center to refill a truck with cargo can be modeled as an ordinary visit to the distribution center. A tour is defined as the list of visits between two subsequent visits to the distribution center. For example, the the first tour in the schedule described by the list $(0, 5, 8, 3, 0, 2, 3 ...)$ refers to visits to customers

5, 8 and 3 in sequence, and then returns to the distribution center to get new cargo. Next it visits customer 2, costumer 3, etc. A complete planning consists of a set of such lists, one for each truck.

Using such a schedule and the distance matrix D the time $t(v)$ of visit v can be calculated easily. The virtual stock $f_{virt.stock}$ is defined as

$$f_{virt.stock}(t) = q_0 - \int_0^t f_c \mathrm{d}t + \sum_{v\,:\,t_v < t} q_v \,.$$

Note that the virtual stock can become negative. Now, given a visit v, the maximal size of a delivery to the customer $f_{max.del}$, can be calculated:

$$f_{max.del}(t) = cap_c - f_{stock}(t),$$

where

$$f_{stock}(t) = \max\{f_{virt.stock}, 0\}.$$

For each visit v the size of the delivery q_v is constrained to

$$0 \leq q_v \leq f_{max.del}(t_v).$$

The amount of cargo a truck can carry is limited too. Furthermore the formula

$$f_{loss} = \max\{-f_{virt.stock}, 0\}$$

is needed. A positive value of this function means that a customer has run out of stock. As the production of the customer has to be stopped, this will result in a loss.

The problem is a real problem, given to us by a company that specializes in planning. According to this company, this problem is a difficult constrained optimization problem. Previous efforts to solve this problem using constraint languages only gave reasonable results for problems involving a small number of customers. The main problem is the dynamic character of the time constraints. The time window for a certain visit v depends on the time t_w and quantity the q_w of all preceding visits w to the same customer.

3. Evolutionary route planning

In order to handle the problem of the previous section with an evolutionary algorithm, we have to decide on three important issues: the representation, the fitness measure and which parts of the problem will be solved by means of evolution.

When introducing problem specific knowledge in the representation it is important to realize that this might lead to a less general planning method. Furthermore experience has shown that it tends to become more difficult to find good recombination operators as the representation gets more problem-specific. In complex representations the information tends to get more context dependent, thereby introducing more epistasis (that is, correlations between parts of the representation). The larger the context of a piece of information, the more difficult it becomes to recombine this information.

Furthermore it is desirable that individuals are likely to represent feasible solutions. If many individuals are infeasible this will result in a slow search. In general 0–1 constraints are undesirable as these kinds of constraints do not carry much information regarding the severeness of a constraint violation, and therefore give no information about the distance to a feasible region.

Which part of the problem should be solved by means of evolution and which part by means of heuristics is strongly dependent on the kind of representation. Representations incorporating much problem specific knowledge often need heuristics and repair mechanisms to maintain valid individuals.

We should be careful with heuristics which can damage the generality of the method. Often the constraints are more difficult to handle than optimization criteria. Therefore it might be advantageous to remove constraints as much as possible, even if this makes the optimization part of a problem a little bit more difficult.

Using these aims it seems reasonable to use a decoder approach together with a simple representation. Decoder approaches have been applied successfully on JSP, see for example Bagchi et al. [BUMK91] and Nakano [Nak91]. In order to do so a simple greedy algorithm is defined. Such a greedy algorithm takes local optimal decisions based on partial knowledge. On problems of interesting size it will be unlikely that this results in a global optimal plan being found, but the greedy algorithm provides us with an interesting starting point for further search. Next, this greedy approach is enhanced by evolving its input in order to get a better plan.

The greedy algorithm, when given a partial plan and a visit to a customer to be scheduled, creates a new partial plan that contains the visit. Hence given a queue of visits to be scheduled and starting with an empty initial plan, such an algorithm can be used to construct a reasonable plan. The plan which comes out of such an algorithm is strongly dependent on the order in which visits are added to the schedule. A reasonable first estimate is to calculate ideal times for all visits and determining the order based on these times.

The pseudo-code for the greedy algorithm is shown in Figure 1. The algorithm is relatively

```
locate all truck
   at the distr.center

for all visit
  determine the ideal time
```

```
sort all visit on time

for all visit
  for all truck
    calculate time of arrival
  select truck with earliest time
  assign visit to this truck
  update customer
  update truck
  if (load truck < min.load)
    return truck to distr.center
```

Figure 1: Pseudo code for the greedy algorithm

straightforward. The first part takes care of finding reasonable values, the second part does the actual scheduling. It will yield a reasonable solution with local near-optimal decisions. This algorithm does not guarantee that a truck will arrive in time. It also does not look for global optimal solutions, as it just uses local information about the current distribution of trucks.

The main problem with the greedy algorithm is that the choices it makes are not guaranteed to be globally optimal. In order to make better decisions more information needs to be processed. Processing more information will slow down the computation significantly. To obtain a global optimum the complete information is needed, so in a way we get back to the point were we started, in that we have a problem that seems to be unsolvable within a reasonable amount of time. This is the point where we add some evolution to the system. In order to do so a representation has to be chosen. After that we define a mutation operator, and a recombination operator that act upon this representation.

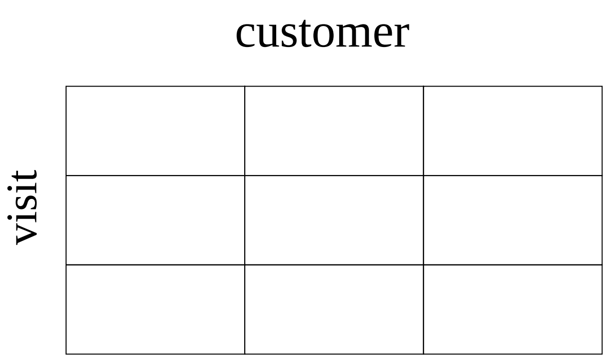

Figure 2: An individual interpreted as a table of estimated times

An individual is defined as a table of estimated visiting times I. Each column in this table represents the visits to a customer. Field $I(j,c)$ represents the estimated time of the j^{th} visit to customer c. An example of an individual is given in Figure 2. The first part of the greedy algorithm in Figure 1 is combined with heavy mutation to create an initial population. The second part of the greedy algorithm is applied to an individual I to generate the actual route plan. In order to apply the greedy algorithm to such an individual we define a list of tuples $(I(j,c),c)$, one for each field in individual I. By sorting the list of tuples on the first field (i.e. on the estimated visiting time) we get a list of visits to customers c. This sorting step corresponds to the start of the second part of the greedy algorithm. The quality of this route plan determines the fitness of the individual $I(j,\ c)$. The representation has some special properties, which are exploited by the evolutionary operators. By sorting the list of estimated times a list of ordered visits is obtained. The sorting step can be left out if we use an order based representation, where each individual contains an ordered list of visits. There are two advantages of this more complex representation. First, the problem has a two dimensional structure. A visit v to customer c is stronger correlated to other visits to the same customer, and to visits which are nearby in time. The reason for using estimated times instead of an order based representation is that the estimated times contain additional information. If two visits are close in time, it is more likely that the interchange of these visits will give a better result. As distance in time does not have to be linearly related to the actual distance in the list, and information would be lost by choosing an order based representation.

The task of the mutation operator is to introduce a (small) modification in an individual, which might lead to a better schedule. The pseudo-code for the mutation operator is shown in Figure 3. In

```
for all $customer$
  for all $visit$
    if $(random(0,1) < P_{mut})$
      $I(visit, customer)$ += $N(0,\sigma)$
```

Figure 3: Pseudo code for mutation operator

order to prevent mutations from being too disruptive, a Gaussian distributed mutation with a small variance is used. This results in that an exchange of visits v and w gets less likely when the the distance in time between the two visits increases.

The task of the recombination operator is to combine good parts of different individuals. It is also important for the recombination operator not to be too disruptive [MdWS91]. This can also be seen intuitively, because the success of certain partial solutions will depend on the context. So in order to combine good partial solutions to a obtain better solution, part of the context has to be preserved by the recombination operator. A recombination operator which is very disruptive is not able to preserve such a context, but will instead create a new (random) plan. Given the two dimensional structure of the individuals there are at least two ways to preserve information. The first is to perform a column-wise recombination. This corresponds to taking the plan for different sites from different parents. The pseudo code for this column-wise recombination operator is shown in Figure 4. A second possibility is

```
for all $j$
  determine parent $p$
  for all $i$
    $I_{child} = I_{par,p}(i,j)$
```

Figure 4: Pseudo code for the column-wise recombination operator

to do a row-wise recombination. Such a recombination combines the first n rows from the first parent with the remaining rows from the second parent. This way one tries to start routing according to the plan of the first parent and later switches to the second parent. In order to reduce the amount of disruptiveness of this recombination operator an even more elaborate strategy was used. Instead of pivoting on a row, a time-pivot is used. The pseudo-code of the row-wise recombination operator is shown in Figure 5.

```
choose $\textit{time-pivot} \in [0..\textit{max-time}]$
for all $i$
  for all $j$
    if $(I_{par,1}(i,j) < \textit{time-pivot})$
      $I_{child}(i,j) = I_{par,1}(i,j)$
    else
      $I_{child}(i,j) = I_{par,2}(i,j)$
```

Figure 5: Pseudo code for the column-wise recombination operator

The greedy algorithm does not give guarantees that the constraints are not violated. It only constructs a reasonable schedule based on a list of visits. The fitness function is used to give a penalty to bad solutions. The fitness function also contains the optimization criteria. We use the following fitness function:

$$f(I) = \frac{\sum\limits_{v \in V} q(v)}{\sum\limits_{v \in V} D(site(v), site(succ(v)))} - M \times \sum_{c \in C} \int_0^{t_{plan}} f_{loss}(t)$$

where V is the set of all visits, C is the set of all customers, $site(v)$ returns the customer belonging to visit v, and $succ(v)$ returns the first visit after visit v. The first term in this formula describes the ratio of the amount of the delivered cargo versus the total traveled distance. A higher value of this term corresponds to a more effective use of the resource, i.e. the use of the trucks. The second term gives a penalty to deliveries that are too late.

A Steady-State Genetic Algorithm is used. Steady-State GAs can be tuned more easily than a generational GA, and by choosing appropriate values for the parameters the Steady-State GA can have the same behavior as a generational GA [Sys91].

Usually Steady-State algorithms use a biased selection scheme for reproduction, while unbiased (uniform) selection is used for the reduction phase. During the reduction phase it is decided which individual will be removed from the population to make room for new individuals. In the description of Genitor [Whi89] it is suggested that the application of a bias in the reduction phase too, can result in a faster search. We go even one step further. We use an unbiased selection of parents for reproduction, so the biased selection for reduction is our prime guide during the evolution. This modified scheme performs better than the standard scheme when a high bias is used, because using a high bias in the standard approach will result in much emphasis being put on the best few individuals. This results in a reduction of the effective population size, which can easily lead to premature convergence. Using an extremely large bias only the best few individuals are able to reproduce. In the scheme the use of a high bias will result in a kind of truncation selection. The probability of premature convergence taking place is much lower as the selection of parents is still done unbiased, so the time it takes the best few individuals to take over the complete population remains quite long. Such a scheme comes closer to the scheme used in Evolution Strategies, which have been shown to be outperform Genetic Algorithms on function optimization tasks [Bäc94]. During previous work on application of an evolutionary computation method to air traffic control [vKHHK95], and work on function optimization, this scheme resulted in a better performance of the evolutionary algorithm.

Using fitness proportionate selection does not make much sense, as it is not known beforehand whether fitness is a linear measure in this case. Therefore we use a rank based approach, where the fitness is used to determine the rank of an individual within the population. The actual lifetime of an individual depends on its rank, and not on its actual fitness. This rank based approach helps in preventing scaling problems, due to which a superindividual can take over the complete population.

4. Experiments

The complexity of a routing problem does not only depend on the size of the problem. The actual distribution of the customers, and the initial stock of different customers can also have a large influence. Therefore all tests have to be performed on a number of different problem instances. The different problem instances are generated at random, in order to prevent an unwanted bias in the test set. A problem instance is obtained by selecting random values for all parameters according to a uniform distribution over the allowed parameter range. The ranges of the different parameters are:

parameter	lower bound	upper bound
storage cap.	$12\frac{1}{2}$	25
consump. rate	stor. cap./4	stor. cap./2
x-coordinate	-50	50
y-coordinate	-50	50
initial stock	0	stor. cap./2
truck cap.	50	50

The coordinates of a customer describe the relative geographic position of the customer with respect to the distribution center. The fields of matrix D are calculated using the Euclidian distances between the locations. For convenience the function f_c describing the consumption of a customer as a function

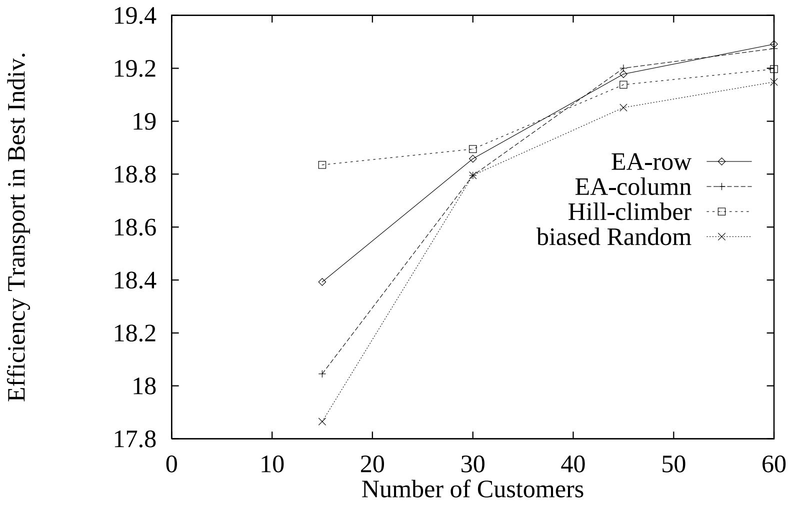

Figure 6: Efficiency of transport (delivered/traveled)

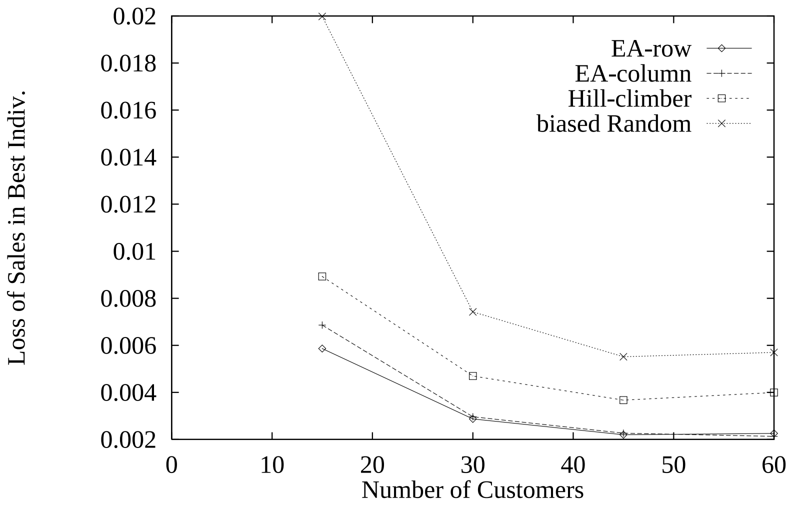

Figure 7: Fractional loss (shortage/delivered)

of time is assumed to be a block function which has a constant value during working hours. Then the amount of stock is decreasing at a constant rate, and in case of a shortage, the loss is proportional to the time the customer has to suspend its production.

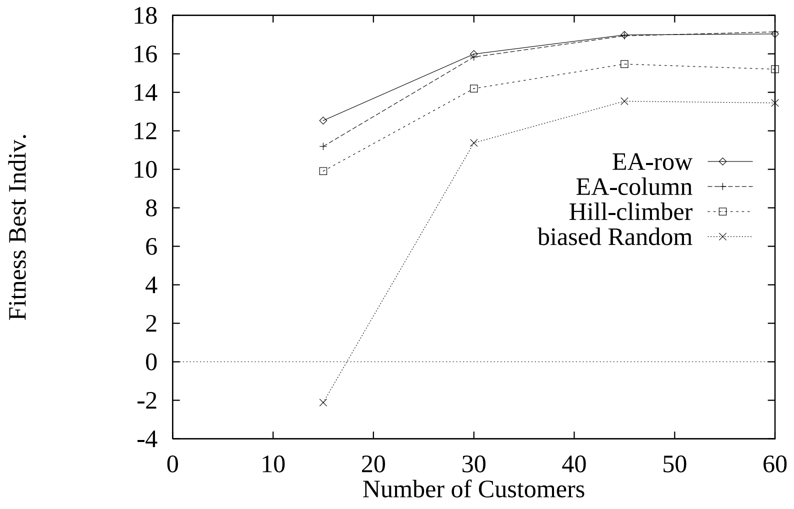

Figure 8: Fitness Best

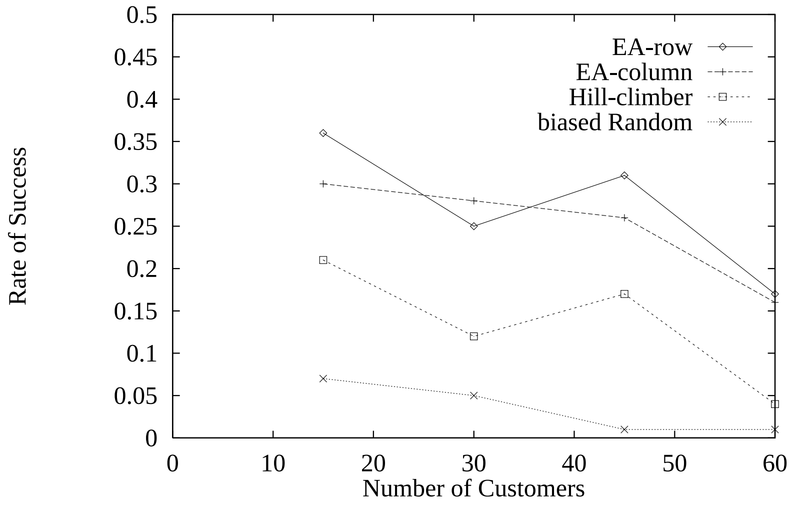

Figure 9: Rate of success

A typical test problem will contain up to 60 customers, 5 visits for each customer, and a fleet of 20 trucks. A typical test problem will contain up to 300 visits to be scheduled. The number of trucks used is linearly related to the number of customers.

The row-wise and column-wise recombination operator are tested separately. To see whether evolution really plays a role two other simple solver methods are defined.

biased random Takes the estimated visiting times, and adds some Gaussian noise to it.

biased hill-climber Uses the estimated visiting times as an initial guess, and then applies the mutation operator to it. The best of the original and the new solution is kept.

Both methods use the same greedy algorithm and fitness function as the EA in order to evaluate the solutions. The biased random method tries to find an enhancement to the estimated visiting times, which are predicted in a naive way. The biased hill-climber tries to find a good solution by improving its current best solution. In order to get a fair comparison all methods are assigned the same number of fitness evaluations.

For the investigation of the scaling properties of our method a sequence of test problems, with an increasing number of customers, is generated. The number of trucks is linearly related to the number of customers.

exp.	customers	trucks
I	15	5
II	30	10
III	45	15
VI	60	20

For each type of problem 100 independent instances are generated at random. A single runs of the algorithm is done for each such instance. The results shown in this paper are averages over these 100 instances. The number of function evaluations is set to 3000.

Next we compare the four different methods. Figure 6 shows the average efficiency of transportation. This efficiency is represented by the ratio of the total amount of delivered cargo and the total traveled distance. On problems I and II the hill-climber approach finds the most efficient solutions. On problem III and IV the EAs perform better when using this measure. It can be expected that the fitness landscape corresponding to the efficiency measure is not too rugged. Minimizing traveled distance can be done relatively easy by means of successive local optimizations, as can also be seen observed in many TSP heuristics. On the larger problem instances better efficiency rates can be obtained because the number of customers is larger, and therefore the distance to the nearest neighbors is smaller.

Figure 7 shows the fractional loss as a function of the problem size. A loss appears when a customer runs out of stock. Because we are assuming that the consumption function f_c is constant, the total loss is proportional to the total time clients are out of stock. Using this measure the EAs perform much better than the other two methods. The random method performs especially bad on the smallest problem instances. This is a result of the quality of the estimated visiting times, which are the basis for each separate solution of the random method. These naive estimated times are expected to be less good for the small problem instances. The average loss can get smaller for larger problem instances as the customers are closer to one another in these larger instances.

Figure 9 shows the percentage of tests where the loss was less than 0.001 as a function of the problem size. The EAs perform better than the other two methods. The average loss gets smaller for the larger problem instances as can be seen in Figure 7. Unfortunately this does not result in an increase of the rate of success. Although average loss is decreasing, the standard deviation is decreasing too, which corresponds to a more consistent value of this measure.

Figure 8 shows the average fitness of the best individual. As this fitness is a weighted sum of the efficiency and loss, features of Figures 6 and 7 can be seen in Figure 8 too.

5. Conclusions

The two EAs perform better than the other two methods using the discussed measures. Given the current results it seems that the row-wise recombination operator performs slightly better than the column-wise recombination operator, though the differences are small. The hill-climber and random approach also seem to be quite powerful, given the size of the problems used. The power of these methods is mainly a result of the power of the simple greedy algorithm used in our approach.

The combination of simple greedy algorithms with some randomness seems to be promising direction for solving planning problems. In our approach the main purpose of the greedy algorithm is to act as a decoder, that is as a mapping between the search space and the actual solutions to the problem. The structure of the search space, should be as simple as possible without loosing too much of the structure of the original problem. Furthermore the greedy algorithm should do part of the constrained optimization process by making appropriate local optimizations. It is important to find a balance between the amount of work done by the greedy algorithm and by the evolution. Putting too much emphasis on the greedy algorithm might lead to a situation in which the actual global optimal solution becomes unreachable, because this solution can not be constructed by the greedy algorithm. Furthermore the greedy algorithm might become too slow.

When trying to solve complex planning problems the combination of a simple greedy algorithm with some randomness can give good results. Especially an evolutionary approach might be interesting, as these methods have shown to be successful on exploring complex search spaces with unknown dependencies between the different variables.

The current results look interesting, but further research is needed. The dynamic behavior of the method is important if there are some small changes in the existing situation, such as an extra customer to be added or a truck which ends up in a traffic-jam, and there is the need for a new plan. Probably some extra constraints will be added such as minimizing the deviations with respect to the current plan.

Another important issue is the relation between the number of trucks and the relative quality of an individual. In the current model different runs have to be done for different numbers of trucks in order to find the minimal number of trucks. If the relative quality of an individual is not too strongly dependent on the number of trucks then the minimal number of trucks can be determined easily in a single run of the method.

Another important issue is the handling other types of (complex) constraints, and to make a comparison to methods from Operations Research on these problems. Especially problems containing non-linear constraints and/or objective functions might be interesting candidates as these problems are often difficult to solve by means of the standard techniques.

References

[Bäc94] Th. Bäck. *Evolutionary Algorithms in Theory and Practice.* Dissertation, Universität Dortmund, 1994.

[BUMK91] S. Bagchi, S. Uckun, Y. Miyabe, and K. Kawamura. Exploring problem-specific recombination operators for the job shop scheduling. In *Fourth International Conference on Genetic Algorithms*, 1991.

[MdWS91] B. Manderick, M. de Weger, and P. Spiessens. The genetic algorithm and the structure of the fitness landscape. In *Fourth International Conference on Genetic Algorithms*, 1991.

[Nak91] R. Nakano. Conventional genetic algorithms for job shop problems. In *Fourth International Conference on Genetic Algorithms*, 1991.

[Sys91] G. Syswerda. A study of reproduction in generational and steady–state genetic algorithms. In G. Rawlins, editor, *Foundations of Genetic Algorithms - 1.* Morgan Kaufmann, 1991.

[vKHHK95] C.H.M. van Kemenade, C.F.W. Hendriks, H.H. Hesselink, and J.N. Kok. Evolutionary computation in air traffic control planning. In *Sixth International Conference on Genetic Algorithms*, 1995.

[Whi89] D. Whitley. The genitor algorithm and selective pressure. In *Third International Conference on Genetic Algorithms*, pages 116–121, 1989.