Optimization of a Container Transport System with the Breeder Genetic Algorithm

U. Bartling
H. Mühlenbein
RWCP*Theoretical Foundation GMD Laboratory
GMD - Forschungszentrum Informationstechnik
D-53754 St. Augustin, Germany

Abstract

In the field of vehicle routing and scheduling the problem of container transportation attracted less attention than one would expect when taking the economical importance into account. Enormous amounts of goods are tranported in containers of various kind. Different networks have been established for the distribution of spare parts in the automobile industry, parcels, deep-frozen food and many more. For both economical and ecological reasons it would make sense to join at least a few of these networks. But besides technical problems like sharing containers between different companies the large number of depots and containers makes the problem so complex that it exceeds the capabilities of human planners. In this paper, we present an approach to optimize the distribution problem with some important constraints like automated generation of balance tours and to quickly produce similar schedules for similar problem instances.

Keywords: vehicle routing problem, adaptive strategy selection, incremental search

1 INTRODUCTION

The problems of finding the minimum number of vehicles needed to visit a certain set of nodes build the class of *vehicle routing problems* (VRP). In the past mainly the single depot VRP (SDVRP) has been investigated. A survey can be found in [3]. Recently, the emphasis has changed to the multiple depot VRP (MDVRP). Here a large number of customers are served by a small number of depots. The MDVRP is a multilevel combinatorical optimisation problem. At the first level the boundaries for each depot service area have to be defined. At the second level, a VRP has to be solved for each depot. At the third level a classical Travelling Salesman Problem (TSP) needs to be solved for each route. Because of its difficulty, the MDVRP has been seldom investigated. A new heuristic has been proposed in [4].

The Container Transportation Problem (CTP) also belongs to the class of node-based scheduling and routing problems. But it has different characteristics than the aforementioned VRP. Here, a large number of containers has to be transported between geographically dispersed depots. In general, a pair of origin/destination depots is given and at least one container must be transported

*Real World Computing Partnership

from each origin to its corresponding destination. A vehicle visits only a few depots, typically two or three. A fleet of trucks is hired for the purpose of transporting the containers. Furthermore, depots do not send out the same number of containers as they receive. Balance tours have to be organized in order to meet the needs for empty containers to be loaded the next day. For a logistics company involved in this business the automated generation of schedules including balance tours for those large networks is highly desirable. The optimization goal is to minimize both the fleet size and the overall driving distance. For both economical and ecological reasons it would make sense to join networks still separated. However, the complexity of a single networks often exceeds the capabilities of human planners.

Typical applications are parcel distribution systems, shipment of clothing, spare parts in the automobile industry, and many more. Despite its high economic importance, the CTP has been investigated seldomly.

Most real-world vehicle routing applications are still fairly different from the ideal mathematical VRP. They can be distinguished into *static, incremental* and *dynamic* problems. In static problems, all demands are fixed and known beforehand. In incremental problems, some fluctuations in demand occur every once in a while. The task is to generate a new schedule which is very similar to the existing one. Experience shows that drivers are accustomed to the tours they made in the past and tend to make errors whenever a new schedule is implemented. The dynamic problem combines dispatching and scheduling. New demands arrive when the cars are underway and the routes are changed accordingly.

In the scientific literature mainly the static problem has been dealt with. But this rarely occurs in practice. A recent survey about dynamic routing problems can be found in [6]. For the CTP both the static and the incremental problem are important. We have developed an optimizer called Cargo/CT addressing exactly these tasks.

The basic design of our optimizer and first results were published in [1]. In this paper we describe the most important extensions to the basic model. First, we sketch out the basic optimization ideas. In chapters 4 and 5 we present the key features of our optimizer. The Adaptive Strategy Selection controls the usage of the elementary optimization strategies. In order to keep consecutive master schedules similar we developed the the Incremental Search.

2 Problem Description

2.1 Real-life problem

We are given a network of depots. Containers are to be transported between these depots. The term *depot* is used in a more general sense here than in other vehicle routing problems. It is not restricted to a classical depot namely a large building with lots of gates where trucks can be (un)loaded. It might also be a railway station or a small company's site where an empty container is dropped off in the morning and picked up loaded in the evening. Each container to be transported is characterized by a place of origin, a destination, and a time window during which the transportation must be carried out. These trips are serviced by an inhomogeneous fleet of vehicles consisting of trucks and trucks with trailers. So we have two types of vehicles, one carrying one container and one carrying two containers.
There exist legal obligations concerning driving times we have to obey. Depending on the tour length there are vehicles with one or two drivers.

A *route* is a sequence of trips carried out by one vehicle. Routes may begin and end in different depots, although it is highly desireable to produce round trips. The majority of the transportation

requests have to be carried out over night. The range of planning covers 24 hours.

Between certain depots there might be no traffic at all. In the case of deep-frozen food traffic takes place between cold-storage depots only.

The following properties clearly distinguish our Container Transportation Problem from other VRP described in the literature:

- **No TSP variant**

  When combining trips to routes there exists no requirement to make it a shortest path route. Moreover, potential users strongly favour commuter traffic between nearby depots.

- **Balance tours**

  Mail order services, for example, deliver a huge amount of parcels every day. However, they do not receive any. In other words, loaded containers will not arrive at their depots. To ensure a sufficient number of empty containers to meet tomorrows need, empty containers have to be transported to these depots. In other words, we have sources and sinks with respect to containers. The task is to find an appropriate set of trips to balance the number of containers in both sources and sinks. Therefore, the set of trips to be carried out is not completely known beforehand.

- **Stop-over points**

  If the distance between the origin and the destination exceeds a certain limit a single driver cannot perform a return trip. A transportation request might be split up in such a way the the first vehicle carries the container to a stop-over depot where the container is dropped off. The vehicle then returns to its home base. Later, the container is picked up by another vehicle and transported to its destination. With an appropriate choice of the stop-over point both vehicles may carry out return trips. Once again, the set of trips might be manipulated by the optimizing process.

For the purpose of this paper, we neglect mounting times. We assume the whole fleet of vehicles to be rented for the purpose of transportation.

2.2 Definitions

The set V of depots and the set A of arcs denoting the possible connections between the depots make up a directed and completely connected graph $G = (V, A)$ without loops. With each arc $(i, j) \in A$ are associated non-negative values

- d_{ij}: driving distance between depot i and j
- t_{ij}: driving time from depot i to j and
- k_{ij}: costs for driving from depot i to j.

The corresponding matrices are $D = (d_{ij})$, $T = (t_{ij})$, and $K = (k_{ij})$. A constant factor λ is needed to compute the costs for the time a driver is on duty. The graph G together with D, T, λ, and K defines the *network*.

The maximum time M two drivers can be on duty is seventeen hours in our case. The range of planning cover a 24 hour time interval.

We define a *trip* c for a container to be a 8-tuple

$$c = (o, d, s, l, w_r, w_a, w_d, id)$$

It describes the task to move a container from the source depot $o \in V$ to the destination depot $d \in V$ along the arc $(o, d) \in A$. The status $s \in \{\mathsf{empty}, \mathsf{loaded}, \mathsf{nil}\}$ describes what kind of container to move. The first two states are self-explaning, the nil-state is needed to describe a trip without any container, also called an unproductive trip. Since there are various types of containers we need a component l to denote the type of a container. The set of types is defined by the problem instance. Normally, container differ by length.

Finally, $[w_r, w_a]$ is the time window during which the trip must be carried out. At time w_r the container is ready for operation, at time w_a it should have arrived at its destination. We require the network and the transportation requests to be consistent, i.e. $w_r + t_{o,d} \leq w_a$, $t_{o,d} \leq 17$ hours. So the trip can be executed in time.

For loaded containers, both r and a are given. An unproductive trip can be carried out anytime. In that case, we have $r = 0$ and $a = P$. For a trip with status empty only the arrival time is given. A ready time, however, is not known in advance. There are times when in a certain depot at least one empty container is available and there are also time intervals when there is none. To overcome this difficulty the ready time is always assumed to be 0. With respect to the real life application this assumption is justified for two reasons. First, each depot has a certain number of empty containers in stock. Second, if a trip with an empty container is scheduled at a time when no one is available the depot manager will rent one from the free market. The side constraint in this case is to return the container at the end of the planning period. The rent for additional containers will increase the overall costs of a schedule only slightly so we can neglect them within the scope of this paper.

Optionally, the trips for loaded containers are assigned a unique transportation request identification number *id*. Balance trips all have the same identification number 0.

For a given trip c let $\mathtt{o}(c)$ denote its origin, $\mathtt{d}(c)$ its destination, $\mathtt{w_r}(c)$ its ready time, $\mathtt{w_d}(c)$ its departure time, $\mathtt{w_a}(c)$ its arrival time, and $\mathtt{s}(c)$ its status.

A vehicle can be a truck with or without a trailer. In the vast majority of cases trucks with trailer will be used. That is why we define a *route* ϱ as two sequences of trips. In a matrix-like notation it is

$$\begin{pmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,n} \\ c_{2,1} & c_{2,2} & \dots & c_{2,m} \end{pmatrix}$$

We shall refer to the trips $c_{1,i}$ as the trips for a truck and to $c_{2,i}$ as the trips for its trailer. For $m = 0$ one can think of the vehicle as being a single truck. Without loss of generality we can assume $n \geq m$.

In real world applications, the two sequences may have different lengths. It allows for more complex combinations of trips to routes. To overcome notational problems we describe the transportation via a stop-over point by splitting up a trip $c = (o, d, s, t, r, a)$ in two sub-trips:

$$c^1 = (o, x, s, t, r, a - \mathtt{t}(x, d))$$
$$c^2 = (x, d, s, t, r + \mathtt{t}(x, d), a)$$

If we assume the trip c to be carried out by the trailer then there exists a trip $\bar{c}$ on the truck with $\mathtt{o}(\bar{c}) = o$ and $(\bar{c}) = x$. Thus, we have $m = n$. At the depot x the container c^1 will not be dropped off.

For each route ϱ we require the following conditions to be true:

$$\mathtt{o}(c_{i,j+1}) = \mathtt{d}(c_{i,j}), j = 1, \dots n-1; i = 1, 2 \tag{1}$$

$$\sum_{j=1}^{n} t_{\mathtt{o}(c_{i,j}),\mathtt{d}(c_{i,j})} \leq M \quad i = 1,2 \tag{2}$$

$$\begin{array}{rcl} \mathtt{o}(c_{1,j}) & = & \mathtt{o}(c_{2,j}) \\ \mathtt{d}(c_{1,j}) & = & \mathtt{d}(c_{2,j}) \end{array} \quad j = 1,\ldots n-1 \tag{3}$$

A follow-up trip starts where its predecessor ends (1) and the tour length constraint is fulfilled (2). The sequence of trips for the truck and its trailer has to be parallel (3). Execution of a route begins at

$$max(\mathtt{w_r}(c_{1,1}), \mathtt{w_r}(c_{2,1}))$$

or later. Consecutive trips can start at

$$max(\mathtt{w_r}(c_{1,j}), \mathtt{w_r}(c_{2,j}))$$

or

$$\mathtt{w_d}(c_{1,j-1}) + t_{\mathtt{o}(c_{1,j-1}),\mathtt{d}(c_{1,j-1})}$$

whichever is later. Waiting times occur whenever a vehicle has arrived before the next containers are ready. While waiting a driver is on duty.

An instance of a CTP is completely described by a network and a set L of trips. Once these data are given a set E of trips can be constructed containing the necessary transportation requests for empty containers. Together with a finite and possibly empty set N of unproductive trips we have $C = L \cup E \cup N$. Note that neither E nor N are needed to describe an instance of the CTP as they are not pre-defined.

Let C be a set of trips. A set of routes is called a *schedule* for C iff every trip in C belongs to one and only one route. In real life, each route will be carried out by a vehicle. So the number of routes in a schedule equals the number of vehicles used.

2.3 Optimization Goals and Cost Function

The optimization task can be stated as finding a least cost schedule for C. We have three classes of costs:

1. **Fixed costs**
   for vehicles and containers reflect acquisition costs, taxes, etc.

2. **Variable costs**
   arise for using the ressources. Normally, they are proportionate to the distance travelled and the working time of the drivers.

3. **Penalty costs**
   are needed for the purpose of guiding the optimization process. For example, if return trips are favoured penalty costs mirror the costs for a virtual return trip.

In cooperation with a logistics company we have designed the following cost function. Executing a route ϱ costs

$$X(\varrho) := F + V_{km} + V_t + P$$

The fixed costs are represented by a constant amount F of money. Variable costs arise from driving

$$V_{km} = \sum_{i=1}^{n} k_{\mathtt{o}(c_{1,i}),\mathtt{d}(c_{1,i})} \cdot d_{\mathtt{o}(c_{1,i}),\mathtt{d}(c_{1,i})}$$

and from the time the driver is on duty

$$V_t = \lambda \cdot (\mathtt{w_a}(c_{1,n}) - \mathtt{w_d}(c_{1,1}))$$

The aforementioned factor λ (cf. 2.2) is a payment by the hour.

Penalty costs arise only if the vehicle does not return to its home base. So if $\mathtt{o}(c_{1,1}) \neq \mathtt{d}(c_{1,n})$ we have

$$P(\varrho) = F + k_{\mathtt{o}(c_{1,1}),\mathtt{d}(c_{1,n})} \cdot d_{\mathtt{o}(c_{1,i}),\mathtt{d}(c_{1,i})}$$

The term F reflects the fact that the vehicle is not available for daytime usage and another one has to be hired.

The costs for a complete schedule are computed as the sum over all routes in the schedule.

2.4 Combinatorial Complexity

To vaguely assess the combinatorial complexity we put aside time windows. Assume we are given a certain set C of trips. Let

$$\mathtt{E}(a,b) := \|\{c \in C | \mathtt{o}(c) = a \wedge \mathtt{d}(c) = b\}\|$$

be the number of trips to be executed from a depot a to a depot b. The upper bound for the number of possible concatenations of two trips then is

$$\sum_{(a,b)\in A} \mathtt{E}(a,b) \cdot \sum_{x \in V} \mathtt{E}(b,x) \tag{4}$$

The combinatorial complexity depends on the number of edges in V, i.e. $(|V|)^2$, and the assignment of trips to depots. For a rough estimation we replace the second sum by the average number of trips leaving a depot

$$\bar{E} = \frac{|C|}{|V|}$$

and get

$$\bar{E} \sum_{(a,b)\in A} \mathtt{E}(a,b) = \frac{(|C|)^2}{|V|}$$

When taking trailers into account we first have to find the number of possible combinations of two parallel trips which is

$$\binom{\mathtt{E}(a,b)}{2} \approx \frac{1}{2} \cdot \mathtt{E}(a,b)^2$$

From Formula 4 we derive

$$\frac{1}{4} \sum_{(a,b)\in A} \mathtt{E}(a,b)^2 \cdot \sum_{x \in V} \mathtt{E}(b,x)^2$$

As an upper bound for the possible number of routes of length 2 for a truck with a trailer we get

$$\frac{1}{4} \cdot \bar{E}^2 \cdot (|C|)^2 \approx \frac{(|C|)^4}{|V|}$$

In general, for routes of length n the complexity is proportionate to $(|C|)^{2n}/|V|$.

One has to bear in mind that C changes during the course of the optimization. Balance tours are created and/or deleted and, more important, trips can be split up. On the other hand, side constraints reduce the number of valid combinations. In practice, we found the average length of the routes to be between 2 and 3.

$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \end{pmatrix}$
$\begin{pmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,m} \\ b_{2,1} & b_{2,2} & \dots & b_{2,m} \end{pmatrix}$
$\dots$

Table 1: Encoding

3 Genetic Approach

3.1 Encoding

An individual represents a complete solution to the CTP. Thus, a route is coded in a gene and the set of genes makes up a complete chromosome. By definition, the chromosome represents a schedule. In contrast to standard encodings for genetic algorithms the chromosome for the CTP is of variable length. First, the length of a route (gene) is limited by the maximum time a driver can be on duty. In rare cases like shuttle traffic there can be up to 10 trips in a route. Second, the number of routes is to be reduced. So the number of genes in the chromosome will be shrinking and may vary from one individual to another.

3.2 Standard Initialization

The balance tours are built by a random process connecting sources and sinks. So for each individual I we have a set E_I containing a specific set of balance tours. The initial schedule for I consists of *elementary routes*

$$\begin{pmatrix} c_{1,1} \in L \cup E_I \\ c_{2,1} \in N \end{pmatrix}$$

If we assume the network and the transportation requests to be consistent this is a valid solution to the CTP. It is also the worst conceivable solution one can find without adding further unproductive trips. At every stage of the optimization process the individuals represent valid solutions iff the genetic operators transform one valid solution into another.

3.3 Genetic Operators

In [1] detailled descriptions of mutation, crossover, and local search are given. We briefly present the elementary optimization strategies implemented in mutation and local search.

(H1) **Attach a trailer**

The task is to find a sequence of trips which can be executed inparallel. Note that it is possible to split up trips if necessary. This increases the combinatorial complexity considerably. We can, for instance, look for two trucks without trailer starting in the same depot and ending in the same depot:

$$\begin{array}{rcl} \varrho & := & (r_{1,1} \quad \dots \quad r_{1,n}) \\ \tau & := & (t_{1,1} \quad \dots \quad t_{1,m}) \end{array}$$

with

$$\begin{array}{rcl} \mathtt{o}(r_{1,1}) & = & \mathtt{o}(t_{1,1}) \\ \mathtt{d}(r_{1,n}) & = & \mathtt{d}(t_{1,m}) \end{array}$$

Splitting trips where necessary we can try to construct a route for a truck with a trailer:

$$\sigma := \begin{pmatrix} r_{1,1} & \dots & r_{1,n} \\ t_{2,1} & \dots & t_{2,n} \end{pmatrix}$$

Without loss of generality we have assumed $n \geq m$.

Likewise, we can look for two trucks without trailer whose trips are "almost" parallel in the sense that they start (end) in the same depot but do not end (start) in the same depot. After appending an unproductive trip we can apply the method just mentioned.

(H2) **Concatenating routes**
Given two routes where the first ends in the same depot where the second starts we can concatenate these routes. Like in the previous case it might be useful to append unproductive trips to fulfill this condition. With

$$\varrho := \begin{pmatrix} r_{1,1} & \dots & r_{1,n} \\ r_{2,1} & \dots & r_{2,n} \end{pmatrix}$$

$$\tau := \begin{pmatrix} t_{1,1} & \dots & t_{1,m} \\ t_{2,1} & \dots & t_{2,m} \end{pmatrix}$$

where

$$\mathtt{d}(r_{1,n}) = \mathtt{o}(t_{1,1})$$

we build

$$\sigma := \begin{pmatrix} r_{1,1} & \dots & r_{1,n} & t_{1,1} & \dots & t_{1,m} \\ r_{2,1} & \dots & r_{2,n} & t_{2,1} & \dots & t_{2,m} \end{pmatrix}$$

(H3) **Replacing unproductive trips**
As shown in the examples above unproductive trips may occur within routes. If there exists a single truck carrying out a trip c which is parallel to a given unproductive trip u we can try to replace u by c.

All the strategies are implemented in both the Local Search and the mutation operator. The main difference is that the Local Search will perform a heuristic only iff a local improvement is guaranteed whereas the mutation performs its action regardless of the consequences (cf. [1]).

In the mutation operator there is also a swap mechanism. Let

$$\varrho := \begin{pmatrix} r_{1,1} & \dots & r_{1,n} \\ r_{2,1} & \dots & r_{2,n} \end{pmatrix}$$

and

$$\tau := \begin{pmatrix} t_{1,1} & \dots & t_{1,m} \\ t_{2,1} & \dots & t_{2,m} \end{pmatrix}$$

be two routes with $n > 0, m > 0$. Assume there is a depot $x \in V$ with $\mathtt{d}(r_{k,i}) = x$, $1 \leq i \leq n$, $k = 1, 2$ and $\mathtt{o}(t_{k,j}) = x$, $1 \leq j \leq m$, $k = 1, 2$. If

$$\bar{\varrho} := \begin{pmatrix} r_{1,1} & \dots & r_{1,i} & t_{1,j+1} & \dots & t_{1,m} \\ r_{2,1} & \dots & r_{2,i} & t_{2,j+1} & \dots & t_{2,m} \end{pmatrix}$$

and

$$\bar{\tau} := \begin{pmatrix} t_{1,1} & \dots & t_{1,j} & r_{1,i+1} & \dots & r_{1,n} \\ t_{2,1} & \dots & t_{2,j} & r_{2,i+1} & \dots & r_{2,n)} \end{pmatrix}$$

are both feasible they replace ϱ and τ in S. In a way, this swap operation shows some similarity to a 1-point-crossover between routes. The crossover-point is a depot where both routes meet.

4 Adaptive Strategy Selection

There are two major optimization goals. One is to minimize the fleet size, the other to minimize the driving distance. Variations to the set of balance tours are made in order to achieve these goals. In fact, for either optimization goal there is a class of strategies. PAR contains all heuristics from H1 and H3 which enhance parallel execution of routes. SEQ contains heuristics from H2 which concentrate on sequential execution of routes. Strictly speaking, there also exists a class PEN dealing with penalty costs. A member of this class of strategies is the one constructing return trips. In the end, it is a special case of concatenation. Even the swap operator can be put into either class. In case the crossover-point x is the termination point of the first route and the starting point of the second one, that is

$$\mathtt{d}(r_{k,n}) = \mathtt{o}(t_{k,0}) = x, k = 1, 2$$

the swap turns into a concatenation. In principle, we have two swap operators. One allows the crossover-point to be at the beginning and will therefore belong to SEQ. The other one requires the crossover-point to be within the route and belongs to PEN. With an appropriate choice of the crossover-point it can created return trips or will at least reduce the distance to the home base.

Up to a point, these optimization goals are mirrored by the cost function (cf. 2.3). While $\sum F$ corresponds to the fleet size the variable costs mainly depend on the driving distance.

At every generation we can compute the ratio

$$Q_i(t) = \frac{total\ driving\ distance}{No\ of\ vehicles}$$

for every individual (schedule) i. With $Q(t)$ we denote the function Q computed for the best individual at time t. The growth of $Q(t)$ during the last n generations can be estimated during of the optimization process as

$$\begin{aligned} \bar{Q}(t) &= \tfrac{1}{n} \sum_{i=1}^{n} Q(t-i) \\ \Delta\bar{Q}(t) &= \bar{Q}(t) - \bar{Q}(t-1) \end{aligned}$$

With $\Delta\bar{Q}(t)$ we have a means to assess the type of progress we observe. If $Q(t)$ is growing very fast it is safe to say the progress is mostly due to the reduction of the fleet size. On the other hand, if $Q(t)$ is growing slowly the driving distance is reduced in the first place with the fleet size shrinking only moderately.

Based on these considerations we build the set of applicable strategies depending on the growth of $Q(t)$. First, we compute the probability

$$p = \begin{cases} exp(-\alpha \cdot \Delta\bar{Q}(t)), & \Delta\bar{Q}(t) \geq 0, \alpha \geq 0 \\ 0 & otherwise \end{cases} \tag{5}$$

We then compute the probabilities to draw from either set of strategies as

$$\begin{aligned} Prob(\text{PAR}) &= \tfrac{1+p}{2} \\ Prob(\text{SEQ}) &= 1 - Prob(\text{PAR}) \end{aligned}$$

If we observe a fast growing $Q(t)$ (reduction of the fleet size) the probability p will be close to 1. So $Prob(\text{PAR})$ will also be close to 1. Thus, we focus on strategies to reduce the driving distance and neglect strategies for the reduction of the fleet size.
On the other hand, if we observe no progress at all then $\Delta\bar{Q}(t)$ will be zero. So $Prob(\text{PAR}) = Prob(\text{SEQ}) = 0.5$ and all strategies can be applied with an equal probality. This makes perfectly

sense because in case of stagnation any strategy available should be applied.
The Adaptive Strategy Selection can be switched off by simply setting $\alpha = 0$. The optimizer then draws always from the set of all strategies with an equal probability.

A slightly different approach to this problem can be found in [9] where the concept of competition between sub-populations was introduced. Individuals belonging to a sub-population all perform the same strategy, e.g. a certain mutation operator. The more successful a strategy is the more individuals it attracts, in other words, the larger the size of that sub-population becomes. The overall population size is kept constant. Even the least successfull sub-population can never die out so every strategy will be applied throughout the whole search.
We have learned from experiments with our container transportation problem that at times it is more successful *not* to apply some of the strategies *although* they would improve the fitness. The progress, however, would only be of short duration. In the long run, it pays to adapt the application of strategies to the kind of progress we observe.

5 Incremental Optimization

5.1 Motivation

A carrier expects a tool like our optimizer to cover a wide range of tasks. In the first place, it is the generation of master schedules. In addition, the tool should also provide answers to other problems.

- **Joining networks**
  Let us assume a logistics company operates a network when a potential customer would like to join that network. What will be the cost of the joint network?

- **Establishing a hub**
  What will be the effect if a certain depot is operated as a hub? Similarly, what will happen if a new hub is established?

- **Placement of depots**
  Would it make sense to close down a poorly utilized depot? Would it make sense to open a new depot?

In all these cases there exist already one or more schedules. Often, the corresponding networks and freight volumes differ only slightly from the problem instance in question. When a new master schedule is needed we are facing exactly the same situation. Be it that a few depots send out more (less) containers than last time, be it that a depot has been removed completely from the network. One would expect the new master schedule to be similar to its predecessor if the problem instance has changed only slightly. However, when using a random process such as a GA one cannot hope to get similar schedules even in multiple runs for one problem instance.

When creating a schedule for a new problem instance there are two major requirements to the optimizer. First, results are needed quickly. Second, the new schedule should be as similar to the old one as possible. As it is often the case, these goals cannot easily be reached at the same time.

In the field of mobile telecommunication we find an analoguos situation. When the need to extend the network arises one cannot come up with a totally different placement scheme for the antennas. Rather, the optimization has to start with the given set of antennas. Changes, if ever, may concern the antenna type only.

5.2 Definitions

First, we introduce two notions of similarity. We call two trips c_1 and c_2 M_1-*similar* if

$$\mathtt{o}(c_1) = \mathtt{o}(c_2) \wedge \mathtt{d}(c_1) = \mathtt{d}(c_2)$$

The trips c_1 and c_2 are M_2-*similar* if they are M_1-*similar* and

$$\mathtt{s}(c_1) = \mathtt{s}(c_2) \wedge \mathtt{id}(c_1) = \mathtt{id}(c_2)$$

holds. Obviously, only trips belonging to different schedules can be M_2-similar. Likewise, two routes

$$\alpha := \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \end{pmatrix}$$

and

$$\beta := \begin{pmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,m} \\ b_{2,1} & b_{2,2} & \dots & b_{2,m} \end{pmatrix}$$

are M_1-*similar* if $a_{i,j}$ is M_1-similar to $b_{i,j}, i = 1, 2$ and $j = 1, \dots n$. M_2-similarity is defined in the same way for routes from different schedules.

The logistics company we are co-operating with is interested in knowing how many routes from a schedule S_1 are also in a schedule S_2. More formally, how many pairs

$$(\alpha, \beta), \alpha \in S_1, \beta \in S_2, \quad \alpha \ M_1\text{-}similar \ \beta$$

exist if each route α, β belongs to one and only one pair. In other words, we are counting the number of vehicles visiting the same depots in the same order. Arrival and departure times may be different and also the load status. Obviously, the number of M_2-similar routes will be smaller because not only the load status but also the identification number must match.

5.3 Adapting a schedule to a new problem instance

Suppose we are given a schedule S_{old} for a certain problem instance. For reasons of simplicity we assume no transportation request identification numbers to be given and all container types to be equal. Let $\acute{L}$ be the new set of trips for loaded containers. Therefore, sources and sinks for empty containers may also have changed.

The process of adapting S_{old} to a new environment takes place when initializing an individual I. So a set E_I of balance tours has to be created, too. The new schedule is called S_I contained in the chromosome of I. Accordingly, L_I is the new set of trips for loaded containers.

(S1) Set S_I to S_{old} and L_I to L.

(S2) The difference sets $\Delta_1 = L \setminus \acute{L}$ and $\Delta_2 = \acute{L} \setminus L$ are built. The execution of trips in Δ_1 is no longer requested. Trips in Δ_2 have to integrated into S_I.

(S3) For every route R in S_I we built the set

$$P_R := \{(\varrho_1, \varrho_2) | \varrho_1 \in \Delta_1, \varrho_2 \in \Delta_2, \varrho_1 \text{ in R}, \varrho_1 \ M_1\text{-similar } \varrho_2\}$$

If P_R is not empty we try to find as many pairs in P_R as possible where we can replace ϱ_1 by ϱ_2 in S_I and R still remains valid. Replaced trips are removed from Δ_1 and Δ_2, respectively. This step is repeated until no further replacement is possible.

(S4) If there are still pairs left we proceed like in the mutation operator. ϱ_1 will be removed from its route R which possibly causes R to be split up. ϱ_2 is added to L_I and as an elementary route to S_I.

(S5) The new sources and sinks for empty containers are determined. Balance tours going from a sink to a source are removed. If necessary, new balance tours are created.

In S3 and S5 random processes are involved. Thus, in different individuals we will have slightly different schedules because we applied minimal changes only. On the one hand, this is a necessary precondition to achieve a high degree of similarity. On the other hand, a population based search algorithm like a GA needs a certain level of genetic variance in the population. Otherwise, the search might terminate too quickly. The trade off between similarity and quality of schedules shows up.

If M_2-similarity is required there is not much of a choice for the random processes. Routes will be destroyed more often and the resulting elementary routes may be connected in a different way than in the initial schedule.

6 Computational Results

First, results for runs starting from scratch are reported. We then present the outcome of the incremental optimization. All experiments were run on PCs with Pentium Pro 200Mhz and 128 MB RAM under LINUX.

Data for these experiments were provided by a logistics company. Their network consisted of 131 depots and 462 loaded containers.

6.1 Producing schedules from scratch

For complex problems a parallel genetic algorithm [5] yields better results than one single population. We set up a parallel BGA on our cluster of 15 PCs with 90 islands (sub-populations). The islands were connected as a 10x9 torus. Each island runs a BGA with 128 individuals and 0.125% selection rate. A copy of one of the selected parents migrates in every generation. Termination criterion was 1000 generation without any improvement.

The averge results of three series of three runs each is shown in Figure 1. In the non-adaptive case the control mechanism for selecting the basic optimization strategies was disabled. The weight factor α (cf. Equation 5) is zet to zero. The search process terminates quickly but fails to produce high quality schedules. With the Adaptive Strategy Selection enabled ($\alpha = 3$) the search takes longer and results in slightly better schedules. By far the best result is produced when each island has its own weight factor. Neighboring islands run the Adaptive Strategy Selection with different yet similar values of α. We subdivided the 10x9 torus into 8 segments. Each segment with 11 islands overlaps a row in the torus. Because $8 \cdot 11 = 88$ the last segment consists of 13 islands. In each segment the weight factor on island i is computed as

$$\alpha_i = \begin{cases} i & i \leq 6 \\ 11 - i & otherwise \end{cases}$$

So neighboring islands have different weight factors.

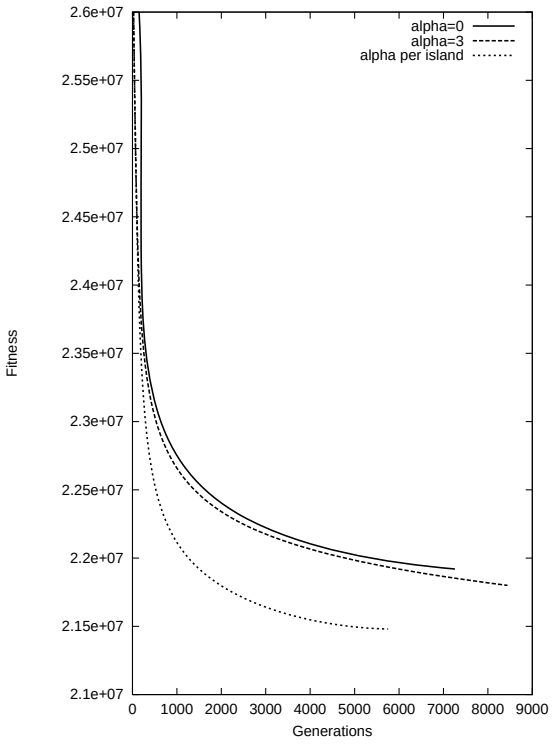

Figure 1: Performance

6.2 M_2-Similarity

The main objective here is similarity of schedules. The two data sets are for the same network on two different days, namely May, 15, and June, 2. At theses days they changed their master schedule. In order to compare the results they also provided some information about their schedules.
Δ_1 contains 8 trips, Δ_2 16 trips.

In preparation of the benchmark we first produced a series of 10 schedules for each day using a panmictic population of 128 individuals. The termination criterion was 1000 generations without improvement. As a reference, we made one run with a Parallel BGA using 120 islands on 15 PCs. This schedule will be referred to as *PAR15*. With *B15* and *B6* we denote the best schedule out of the series of ten.

When starting from scratch the ten schedules differ strongly from one another (Table 2. Routes for long distance traffic only are common to almost all schedules because of the tight time windows further routes cannot be appended. Other elementary routes are combined in a completely different way.

Three series of ten runs each in the Incremental Search mode were performed. The Incremental Search started from the best schedule out of the ten panmictic runs and from the reference schedule. Our intention was to investigate whether a highly optimized schedule is more sensitive to changes than other schedules. As can be seen from Table 3 the incrementally created schedules are more similar to one another than those created from scratch.
Schedules derived from the *PAR15* are particularly similar to one another.

Date	Avg No of Similar Routes M_1	M_2	Avg Number Routes	Generations
May,15	83.6 = 38.5%	67.2 = 30.98%	216.9	11,670
June,6	91.4 = 41.2%	73.3 = 33.05%	221.8	11,520

Table 2: Multiple runs from scratch

Transition	Avg No of Similar Routes M_1	M_2	Avg Number Routes	Generations
B6 $\rightarrow$ May,15	122.3 = 56.08%	105.9 = 48.53%	218.5	9386
B15 $\rightarrow$ June,6	150.6 = 70.03%	133.5 = 62.04%	215.1	7536
PAR15 $\rightarrow$ June,6	192.4 = 91.88%	178.5 = 85.25%	209.4	2074

Table 3: Multiple runs with Incremental Search

Transition	Similarity to Initial Schedule M_1	M_2	Routes
B6 $\rightarrow$ May,15	128.5 = 60.90%	116.9 = 55.40%	211
B15 $\rightarrow$ June,6	151.9 = 71.90%	141.5 = 67.06%	211
PAR15 $\rightarrow$ June,6	180.9 = 90.45%	177.7 = 88.85%	200
man sched		134 = 77.46%	173

Table 4: Comparison with the initial schedule

The manually generated schedule consists of less routes than the machine-generated. This is due to the fact that a few routes can be generated covering a 24 hour time period. Approximately every eight hours a new driver takes over the vehicle. At the time these experiments were executed our optimizer was not able produce those routes. Instead, it generates two or three separate routes.

The most remarkable result is that the machine generated schedules based on the *PAR15* show on average a higher degree of similarity than the hand-written schedule. So it shows that the highly optimized schedule is not sensitive to changes. In the contrary, the less optimized schedules leave room to further optimizations taking place in the incremental phase. Obviously, it pays to start from the best schedule available.

The Incremental Search also reduces the run time considerably if the reference schedule is used as a start. The search is speeded up by a factor of 5.

We like to point out that these results were achieved without implementing any strategy whatsoever enforcing similarity. Conceiveably, one could protect certain routes from being affected by mutation. If the need arises this can still be done.

We also investigated the impact of the Incremental Search on the costs.

Master Schedule	Costs	Incremental Step May, 15	June, 6
May, 15	225131		233200
June, 6	233323	234600	

The incremental step from May, 15, to June, 6, yields surprisingly goods results. On average, the costs for the incrementally generated schedules are almost equal to the costs of the master schedule. However, this is not true if we perform the step in the other direction. The costs are

now about 4.2% higher than for the master schedule. This step seems to be harder which may be due to the fact that more trips (16) are to be removed from the master schedule. More data for consecutive problem instances would be needed to make more profound statements.

The incremental step from May, 15, to June, 6, performed by human planners resulted in a schedule with about 3% higher costs.

Note that costs mentioned in this chapter include penalty costs and are computed according to an artificially designed cost function. They are nowhere close to real life costs.

6.3 Outlook

Using a tool like Cargo/CT results in a growing database of schedules together with their corresponding network data. If the main objective is the fast generation of a schedule the Incremental Search starts with searching the database for candidate instances. The corresponding schedules are adapted to the new task. Thus, the variance in the initial population is very high and one cannot expect the resulting schedule to be similar to any one of the candidate schedules. The search process, however, will be faster due to the higher variance.

How often the incremental step can be repeated remains as an open question. With respect to the results from the previous section one can proceed as follows. Assume there is a master schedule S_0 created from scratch. In an incremental step a similar schedule S_1 is generated. Then for the next incremental step both S_0 and S_1 may serve as a starting point. In the end, we have a sequence $S_0, S_1 \ldots$ of similar schedules we can draw from. Normally, a new master schedule is produced over the weekend. So there will be time enough to generate a lot of schedules from various combinations of starting points. A human planner will select the schedule he thinks is the best one. To assess the trade-off between similarity and quality another schedule can be created from scratch. A decision has to be made which schedule will be the new master schedule.

7 CONCLUSION

We have tested our optimizer based on the Breeder Genetic Algorithm on a number of benchmarks supplied by major carriers in Germany. All the tests were successfully completed. If hand-written schedules existed for the benchmark our optimizer was able to produce better schedules without being tuned to the special requirements.

The key feature of our optimizer is first and foremost the Adaptive Strategy Selection. It not only improves the quality of the solutions it also reduces the run time. In numerous experiments we have found that the above given results hold for different networks and different fitness functions.

In every day use the fast generation of high quality solutions is required. Moreover, an additional objective is to produce similar solutions to similar problem instances An equally efficient and elegant way to deal with this kind of problems is the method of Incremental Search. Previously created solutions which seem appropriate are selected and adapted to the new requirements. The search process starts with pre-optimized solutions instead of randomly created ones. Special care has to be taken in designing the adaptation process. It must not destroy difficult to built internal structures of a solution yet provide the necessary variance. According to our experience, the adaptation process should act like a mutation.

The benchmark for the Incremental Search was completed successfully. As regards the objective of similarity the machine-generated schedules maintained a higher degree of similarity and at the same time showed an acceptable increase of the costs.

References

[1] Ulrich Bartling, H. Mühlenbein *Optimization of large scale parcel distribution systems by the Breeder Genetic Algorithm (BGA)*. Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA '97), Morgan Kaufman, pp. 473-480. ISBN 1-55860-487-1

[2] Ulrich Bartling, H. Mühlenbein *A Breeder Genetic Algorithm with Adaptive Strategy Selection for Optimizating a Container Transport System* submitted for the Genetic and Evolutionary Computation Conference (GECCO), 1999.

[3] M. Gendreau, G. Laporte, J.-Y. Potvin *Vehicle routing: modern heuristics*. Local Search in Combinatorial Optimization eds.:E. Aarts and J.K. Lenstra 311-336 Wiley:Chichester 1997

[4] E. Hadjiconstantinou R. Baldacci *A multi-depot period vehicle routing problem arising in the utilities sector*. Journ. ORS 49 1239-1248 (1998)

[5] H. Mühlenbein *Evolution in Time and Space - The Parallel Genetic Algorithm*. Foundations of Genetic Algorithms, Morgan-Kaufman, pp. 316-337 (1991).

[6] H. N. Psaraftis *Dynamic Vehicle Routing: Status and Prospects*. Annals of Operations Research 61, 143-164 (1995)

[7] M.W.P. Savelsbergh *Local search for routing problems with time windows*. Annals of Operations Research 4, pp. 285-305 (1986)

[8] M.W.P. Savelsbergh *An efficient implementation of local search algorithms for constrained routing problems*. European Journal of Operations Research 47, pp. 75-85 (1990)

[9] D.Schlierkamp-Voosen, H. Mühlenbein *Adaptation of Population Sizes by Competing Subpopulations* Proceedings of the International Conference on Evolutionary Computation (ICEC'96), Nagoya, Japan, pp. 330-335 (1996).