EFFICIENT ROAD-MAP MANAGEMENT FOR A CAR NAVIGATION SYSTEM

by P.D.M. LAHAIJ and R.H.H. WESTER**

**Nederlandse Philips Bedrijven, P.D. Consumer Electronics, P.O. Box 218, 5600 MD Eindhoven, The Netherlands

**Philips Research Laboratories, P.O. Box 80.000, 5600 JA Eindhoven, The Netherlands

Abstract
A new heuristic algorithm, called 'history search', for solving assignment problems is presented. The new algorithm is applied to the problem of finding a cost-efficient arrangement of road-map data on a Compact Disc. A comparison is presented with some well known algorithms, in particular with Simulated Annealing. From the numerical results it is concluded that our heuristic algorithm compares favourably both with respect to the quality of the final result and to the running time.

Keywords: assignment problem, combinatorial optimization, heuristic algorithm, simulated annealing

1. Introduction

CARIN, a CAR Information and Navigation system¹, uses a Compact Disc (CD) for the storage of a detailed digitized road-map of a large area. Unfortunately, a CD player is, compared to other storage media, a relatively slow device regarding retrieval times. However, an intelligent storage arrangement of the road-map data on the CD can improve the retrieval times.

In this paper we present and compare some approximation algorithms solving this storage ordering problem²,³). The approximation algorithms are presented in a general form such that they can be applied to a general class of assignment problems. Our comparisons are based on an application to a specific road-map data storage problem on Compact Disc.

In sec. 2 the problem is formulated. In sec. 3 approximation algorithms are discussed including a new heuristic algorithm and the Simulated Annealing algorithm. Section 4 contains experimental results. In the last section we summarize the results and draw some conclusions.
2. Problem formulation

To store a road-map, it is partitioned into small sub-maps called buckets, which are sequentially stored on the sectors of the spiral track of a CD. A CD is a relatively slow device with respect to retrieval times, compared to hard disc drives. Jumps of the laser head over many tracks cost relatively a large amount of time. This is because a Constant Linear Velocity is used, which implies that after a large jump of the head the rotation velocity must be adjusted. Furthermore, a large jump has an inaccuracy up to 10%. Small jumps take a small amount of time and can be performed accurately.

To improve upon retrieval times, we propose an intelligent arrangement of the buckets on the CD. The rationale behind this is understood by considering the possible road-map requests in more detail.

To keep track of the position of the car at any moment, it is required to store the road-map data of an area around the current position of the car in main memory. Therefore, the buckets that cover a rectangular area, for instance the shaded buckets in fig. 1, are kept in main memory. When the car moves, a number of new neighbouring buckets will be retrieved from the CD and others can be released, as displayed in fig. 1. The requested buckets should be close to each other on the CD, so that only small jumps of the laser head are needed to read them. In other words, buckets that are 'close' to each other on the map have to be 'close' to each other on the CD.

This problem can be formulated as a Quadratic Assignment Problem.

Fig. 1. Rectangular area around the current position of the car (heavy dot). The shaded buckets are in main memory when the car is positioned in bucket 11. Buckets 5, 6, 8 and 13 have to be retrieved when the car is moving towards bucket 7.
Efficient road-map management for a car navigation system

(QAP). It can be seen as the assignment of map-areas to CD-addresses. A QAP is an NP-complete problem. The size of such problems is usually too large to obtain an exact solution within a reasonable amount of computer time. Therefore, there is a need for heuristic approximation algorithms that obtain, in an acceptable time, near-optimal results.

In our case the QAP can be formulated as a pair \( (R, C) \) where \( R \) denotes the configuration space, i.e. the set of possible arrangements of the buckets and \( C \) the cost function which assigns to each configuration a real number denoting a quantitative measure of goodness of that configuration.

A cost function can be given by using the following definitions:

\[
\begin{align*}
B & \quad \text{the set of all buckets}, \\
N(i) & \quad \text{the set of all neighbouring buckets of bucket } i \text{ on the map}, \\
s_i & \quad \text{the start address of bucket } i \text{ on the CD}, \\
e_i & \quad \text{the end address of bucket } i \text{ on the CD}, \\
c_{ij} & \quad \text{the connection weight between neighbouring buckets } i \text{ and } j \text{ and} \\
t_{e,s} & \quad \text{the time needed to jump from CD-address } e \text{ to CD-address } s.
\end{align*}
\]

The cost function is now defined as

\[
\sum_{i \in B} \sum_{j \in N(i)} c_{ij} t_{e_{i,j}}.
\tag{2.1}
\]

The problem is: find an assignment of buckets to CD-addresses such that eq. 2.1 is minimal.

The connection weight \( c_{ij} \) is a measure of the desirability that buckets \( i \) and \( j \) have to be stored close to each other on the CD. These connection weights depend on the road-pattern of the map and on the size of the rectangular area, as shown in the following definitions:

\[
\begin{align*}
R (k, l) & \quad \text{the set of buckets that have to be retrieved from the CD for a} \\
& \quad \text{chosen rectangular area if the car moves from bucket } k \text{ to bucket} \\
& \quad l, (R(k, l) = \emptyset \text{ if } k \text{ and } l \text{ are not neighbours}) \\
M & \quad \text{a set of road-classes, (we chose a categorization of four classes)} \\
r_m & \quad \text{a number representing the importance of road class } m \in M, \\
n_{mk} & \quad \text{the number of roads of class } m \in M \text{ crossing the mutual boundary of buckets } k \text{ and } l (n_{mk} = 0 \text{ if } k \text{ and } l \text{ are not neighbours}). \\
X_{klij} & \quad 1 \text{ if buckets } i, j \in R (k, l) \text{ and } j \in N(i), \\
& \quad 0 \text{ otherwise.}
\end{align*}
\]

The connection weights \( c_{ij} \) are now defined as
P.D.M.E. Lahaije and R.H.H. Wester

The jumping time \( t \) between the end of sector 1 and the beginning of sector \( s \).

\[
0.75 \\
0.5 \\
0.25 \\
0.25 \\
0 \\
\]

\( t (s) \)

2 4e4 8e4 sector

Fig. 2. The jumping time \( t \) between the end of sector 1 and the beginning of sector \( s \).

\[
c_{ij} = \sum_{k, l \in B} \chi_{klij} \sum_{m \in M} r_m n_{mkj}.
\]  

(2.2)

The times \( t_{e,s} \) are determined by the CD-player's access mechanism. It is a discontinuous function due to the differences in times needed for small and large jumps. The function is plotted in fig. 2, where the first argument is kept constant and the second argument is variable. An expression for the access time is given in the appendix, eq. A4.

3. Approximation algorithms

In this section we present some approximation algorithms. A new heuristic algorithm is presented that is applicable to many assignment problems. The underlying technique is called 'history search'. A second algorithm is the Simulated Annealing algorithm. For both heuristic algorithms, first a general approach is presented and then the specific application to the storage ordering problem.

3.1. History search assignment

This heuristic procedure creates gradually a feasible solution. The basis is the use of information of the part of the assignment that has been made. This information is problem-dependent and can be obtained intuitively or
Efficient road-map management for a car navigation system

experimentally. We will present the general approach, followed by an application to the bucket ordering problem.

3.1.1. Theory

Consider a general assignment problem, where elements of a set $W$ have to be assigned to elements of a set $V$. Let $F \subset (W \rightarrow V)$ represent the set of feasible solutions. A general scheme for gradually constructing an $f \in F$ is defined by the induction on number of assigned elements. For any application, a number of problem dependent definitions have to be made. Later we will give examples of such definitions.

We presuppose an ordering on the set $V$.


$a$ neighbourhood for each $w \in W$, the set of assigned elements
represents a partial assignment that is extendible to at least one feasible solution,
a set of candidates for the next element to assign,
a relation between elements from $W$ and the set of elements that have been assigned $S_n$,
a relation between elements from $W$ and the set of elements that still have to be assigned $W \setminus S_n$,

\begin{align*}
N(w) & \quad \text{a neighbourhood for each } w \in W,
S_n & = \{w_1, w_2, \ldots, w_n\}, \text{the set of assigned elements}
f_n : S_n \rightarrow V & \quad \text{represents a partial assignment that is extendible to at least one feasible solution,}
T_n & = \bigcup_{w \in S_n} N(w) \setminus S_n, \text{a set of candidates for the next element to assign,}
h_n : W \rightarrow R, & \quad \text{a relation between elements from } W \text{ and the set of elements that have been assigned } S_n,
g_n : W \rightarrow R, & \quad \text{a relation between elements from } W \text{ and the set of elements that still have to be assigned } W \setminus S_n,

\text{initialization:}
S_0 = \emptyset, \quad T_0 = \emptyset,
\forall w \in W : h_0(w) = \alpha(w) \text{ and } g_0(w) = \beta(w)
\alpha \text{ and } \beta \text{ are initial values for the functions } h \text{ and } g.

\text{induction on } n:
1. if $T = \emptyset$
   then pick a $w_{n+1} \in W \setminus S_n$
   else pick a $w_{n+1} \in T_n$ such that
   \[ \forall \nu \in T_n : h_n(\nu) < h_n(w_{n+1}) \lor [h_n(\nu) = h_n(w_{n+1}) \land g_n(w_{n+1}) \leq g_n(\nu)] \]
2. $S_{n+1} = S_n \cup \{w_{n+1}\}$,
3. $\forall w \in S_n : f_{n+1}(w) = f_n(w)$ and
   \[ f_{n+1}(w_{n+1}) = \min_{f \in F} \{f'(w_{n+1}) | \forall w \in S_n : f'(w) = f_n(w)\} \]
4. $T_{n+1} = T_n \cup N(w_{n+1}) \setminus S_{n+1}$,
5. $\forall \nu \in W : h_{n+1}(\nu) = \Psi_{n+1}(h_n(\nu), \nu, w_{n+1})$
   $g_{n+1}(\nu) = \Phi_{n+1}(g_n(\nu), \nu, w_{n+1})$
The functions $\Psi$ and $\Phi$ are problem-dependent. We will deal with them later.

**finally:**

$S_n = W$ and $f_{|W|}$ is the assignment.

Remarks:

The following definitions are required for application to a given problem instance: firstly the ordering on the set $V$ and the set of feasible solutions $F$, next the neighbourhoods $N(w)$ and the functions $\alpha(w)$, $\beta(w)$, $\Psi_n(x, v, w)$, $\Phi_n(x, v, w)$. These functions are automatically defined by the definition of the function $h_n$ and $g_n$.

In the following subsection we give two examples of these definitions.

### 3.1.2. Application

For the application to the bucket ordering problem we can make the following specifications:

- $W$ equal to the set of all buckets $B$,
- $V$ the (ordered) set of CD-addresses,
- $F$ the set of all possible assignments of the buckets, such that for each $f \in F$, $f$ represents a consecutive ordering of the buckets on the CD,
- $N(w)$ defined as in sec. 2.
- $h_n(w)$ is a 'heuristic connection value' of bucket $w$ with all the ordered buckets (we will give an intuitive explanation hereafter),
- $\gamma_w \in W : \alpha(w) = 0,$
- $\Psi_n(x, v, w) = \text{integer } [a \cdot (x + c_{vw})]$, for $0 < a < 1$ and $c_{vw}$ as in eq. 2.2,
- $\Phi_n(x, v, w) = \begin{cases} x & \text{if } v \notin N(w) \\ x - 1 & \text{if } v \in N(w). \end{cases}$

In induction step 1, a bucket with the highest heuristic connection value is selected or, if there are two buckets with maximum heuristic connection value, an arbitrary bucket with the lowest number of not-ordered neighbours is chosen. In induction step 5 the new heuristic connection values are computed. Intuitively the expression supports the idea that buckets which have a high connection value with all the ordered buckets should get a high
heuristic connection value such that they will be placed close to these or-
ordered buckets. The multiplication factor \( a \) in the function \( \Psi \) is a devaluation
of the history. This is deduced from the CD-access mechanism, see the appen-
dix. For the access mechanism it is not important whether two neigh-
bouring buckets have 10 tracks or 20 tracks in between them. Of importance
is the fact that two neighbouring buckets lie sequentially or have more than
a certain number of tracks in between them, see also fig. 2. Hence the last
ordered bucket should have more influence on the choice of a new candi-
date than the buckets ordered before. A good value for this factor \( a \) is hard
to obtain; a small variation of \( a \) very often leads to a different bucket or-
dering.

We end this section with some remarks. In practice the history search is
done \( |W| \) times, each time with a different initial element \( w_i \). The best con-
figuration is kept. Sometimes an iterative improvement (see sec. 3.2.1)
starting off with the best configuration can improve the final result.

We successfully applied the same history search procedure on a Graph
Colouring Problem with the following definitions:

\( W \) contains all the nodes of a graph,
\( V \) a set of colour numbers.
\( N(w) \) are all the neighbours of node \( w \).
\( h_n(w) \) denotes the number of forbidden colours for node \( w \), so

\[
\forall w \in W : \alpha(w) = 0 \text{ and } \\
\Psi_n(x, \nu, w) = \begin{cases} 
  x + 1 & \text{if } \nu \in N(w) \text{ and } \\
  x & |\{z \in S_n \cap N(\nu) \mid f_n(w) = f_n(z)\}| = 1 \\
\end{cases}
\]

\( g_n(w) \) is minus the number of not coloured neighbours of node \( w \), thus

\[
\forall w \in W : \beta(w) = -|N(w)| \text{ and } \Phi_n(x, \nu, w) = \begin{cases} 
  x + 1 & \text{if } \nu \in N(w) \\
  x & \text{if } \nu \notin N(w). \\
\end{cases}
\]

This history search for a graph colouring problem turned out to be very ef-
fective. For a random graph with 50 nodes and half density (each node had
about 25 vertices), this heuristic of quadratic order found the absolute min-
imum number of colours in only a few seconds.
3.2. Simulated Annealing

Detailed information about the theoretical and implementational aspects of the Simulated Annealing algorithm is found in Laarhoven and Aarts\(^5\); here we merely restrict ourselves to a summary of the relevant aspects.

3.2.1. Theory

Simulated Annealing is a generalized version of iterative improvement in the sense that in addition to improvement the algorithm also accepts deteriorations.

Consider a Combinatorial Optimization Problem (COP), and define for each configuration a set of neighbouring configurations. Iterative improvement tries, starting off from an initial configuration, to improve the value of the cost function by repeatedly generating transitions from the current configuration to a neighbouring one. This is done by selecting a configuration from the neighbourhood. When this configuration is better it becomes the current configuration. Iterative improvement stops when a configuration is reached that has no neighbouring configuration with better value of the cost function.

Generally the final configuration has a value of the cost function that deviates considerably from the global optimum. To prevent this, one could use an algorithm that accepts, in a limited way, configurations with a deteriorated value of the cost function. An algorithm based on this idea was in 1982 introduced by Kirkpatrick, Gelatti and Vecchi\(^6\) and independently by Černy\(^7\). It is now generally known as the Simulated Annealing algorithm due to the analogy with the simulation of the annealing of solids.

With the Simulated Annealing algorithm, solutions can be obtained that are independent of the initial configuration and have a cost function value usually close to a global minimum. Another major advantage is that it is generally applicable. The major drawback is the large computational effort which is required for many applications.

Simulated Annealing starts off with a certain initial configuration. It generates a proposal for a transition from the current configuration to a neighbouring one according to a generation probability. This transition is accepted according to a criterion that allows deteriorated values of the cost function. In the beginning of the optimization, the acceptance criterion accepts almost every proposed transition. As the optimization proceeds, the probability to accept deteriorated values of the cost function decreases and finally approaches zero. The control of the acceptance probability is done by a so-called cooling parameter.
Efficient road-map management for a car navigation system

The convergence of the Simulated Annealing algorithm is determined by a set of parameters combined in this cooling parameter. These parameters are the starting value of the cooling parameter, the number of proposed transitions for a constant value of the cooling parameter, a decrement rule for the cooling parameter and a stop criterion to terminate the algorithm.

In summary, the necessary ingredients for the Simulated Annealing algorithm are:
- the definition of the configuration space and a cost function,
- a transition mechanism, that is, the neighbourhoods of the configurations, the generation and acceptance probabilities,
- a cooling schedule.

3.2.2. Application

We will now discuss our application of Simulated Annealing to the bucket ordering problem. A configuration space and a cost function have been defined in sec. 2. We used the standard exponential form of the acceptance probabilities defined by Kirkpatrick et al.

We investigated different transition mechanisms. First, we used the transition mechanism applied by Burkard and Rendl\(^8\) to small Quadratic Assignment Problems, the largest being for 36 objects. This transition mechanism is called the swap mechanism. Applied to our problem it randomly selects two buckets from a given ordering and swaps their positions. This has only a local effect on the cost function. For the two selected buckets, the jumping times with all their neighbours on the map have to be recomputed as the CD-addresses of the selected buckets have changed.

The second transition mechanism we applied is a well-known transition mechanism for a Travelling Salesman problem. This transition mechanism is called the \(k\)-opt transition\(^9\), first described by Lin and Kernighan. It can be applied to our problem by considering an ordering as a tour that visits all the buckets. The simplest form is the 2-opt transition, it selects two buckets and reverses the part of the ordering in between the two buckets. A change for the bucket ordering is not local now since the CD-addresses of all reversed buckets have changed. As the buckets have connections with their neighbours on the map, all jumping times between buckets of the reversed part of the ordering and their neighbours on the map have to be recomputed.

We obtained better solutions with the 2-opt transition mechanism than with the swap mechanism in the same amount of computer time. The transition mechanisms are compared in table I for a map containing 100 buckets.

We remark that the discontinuity of the jumping times in the cost function
Results with different transition mechanisms for a map containing 100 buckets. The number of proposed transitions for a constant value of the cooling parameter, called the ‘chain length’, is taken to be the size of the neighbourhood \(\frac{1}{2}N(N-1)\) elements.

<table>
<thead>
<tr>
<th>transition mechanism</th>
<th>value of cost function 2.1</th>
<th># chains of 4950 proposals</th>
<th>total CPU time in hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>swap</td>
<td>740.3</td>
<td>1314</td>
<td>4.47</td>
</tr>
<tr>
<td>swap</td>
<td>723.3</td>
<td>666</td>
<td>2.25</td>
</tr>
<tr>
<td>swap</td>
<td>778.6</td>
<td>186</td>
<td>0.63</td>
</tr>
<tr>
<td>2-opt</td>
<td>669.5</td>
<td>371</td>
<td>7.01</td>
</tr>
<tr>
<td>2-opt</td>
<td>684.9</td>
<td>109</td>
<td>2.06</td>
</tr>
<tr>
<td>2-opt</td>
<td>699.9</td>
<td>43</td>
<td>0.86</td>
</tr>
</tbody>
</table>

had a large influence on these results. A linear jumping function on a small map improved the performance of the swap mechanism. However, the 2-opt transition mechanism was still about 2% better.

The cooling schedule for our experiments was taken from Aarts and van Laarhoven\(^{10}\). In their article they conclude that this cooling schedule is very powerful. Aarts and van Laarhoven applied this cooling schedule to a Travelling Salesman Problem and obtained a two percent deviation from the optimum, independent of the problem size, using a cubic time restriction.

We mention three implementational aspects that generally improve the performance of Simulated Annealing algorithm.

Simulated Annealing first generates a global arrangement of the buckets, after which only small rearrangements will be accepted. Therefore we implemented a range limiter for the transition mechanism. This means that as the optimization proceeds, small rearrangements are proposed with larger probability than large rearrangements. Especially for the 2-opt transition mechanism this is an important aspect as small rearrangements are more easily to compute than larger rearrangements.

A second aspect is that the final configuration of the Simulated Annealing algorithm is not always the best configuration that is investigated. Therefore we keep the best configuration found during the optimization process.

Finally we apply an iterative improvement on the best configuration found by Simulated Annealing, such that the ending configuration is a local minimum.
4. Experimental results

4.1. Set-up

All programs were written in PASCAL and run on a VAX/780 under the VMS operating system.

We used three detailed road-maps, each containing about 100 buckets. We chose four road-classes 1, 2, 3 and 4. They were given weighting factors \( r_i = i \).

We assumed that all buckets occupy 9 sectors on the CD. Since the available road-map data only covers a small part of the CD, the CD-access mechanism was scaled in order to simulate a full CD; without this scaling large jumps would never occur. The maximum number of tracks that can be jumped quickly was reduced to 5, 12 and 24 tracks.

We made a random ordering, denoted by ‘random’ in table II, to give an indication of possible improvements. An ordering technique called ‘Peano arrangement’ \(^{11}\) was also tested. This technique does not use the explicit form of the road-pattern but is based on geometry. A Peano arrangement is an arrangement on the centre coordinates of the buckets such that bucket close

### TABLE II

The experimental results for the three maps.

<table>
<thead>
<tr>
<th>ordering method</th>
<th>max tracks for quick jumps</th>
<th>map # 1</th>
<th>map # 2</th>
<th>map # 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>value of cost function 2.1</td>
<td>total CPU time in s.</td>
<td>value of cost function 2.1</td>
</tr>
<tr>
<td>random</td>
<td></td>
<td>2431.2</td>
<td>1.5</td>
<td>2536.6</td>
</tr>
<tr>
<td>Peano</td>
<td></td>
<td>1178.6</td>
<td>1.7</td>
<td>1188.6</td>
</tr>
<tr>
<td>heuristic 5</td>
<td></td>
<td>879.6</td>
<td>131.1</td>
<td>973.6</td>
</tr>
<tr>
<td>heuristic 8</td>
<td></td>
<td>819.7</td>
<td>594.5</td>
<td>900.2</td>
</tr>
<tr>
<td>Sim. Ann.</td>
<td></td>
<td>870.6</td>
<td>8963.0</td>
<td>887.6</td>
</tr>
<tr>
<td>random</td>
<td></td>
<td>2160.4</td>
<td>1.4</td>
<td>2252.0</td>
</tr>
<tr>
<td>Peano</td>
<td></td>
<td>1020.2</td>
<td>1.6</td>
<td>1097.9</td>
</tr>
<tr>
<td>heuristic 12</td>
<td></td>
<td>754.2</td>
<td>129.3</td>
<td>821.7</td>
</tr>
<tr>
<td>heuristic 24</td>
<td></td>
<td>739.5</td>
<td>509.1</td>
<td>777.6</td>
</tr>
<tr>
<td>Sim. Ann.</td>
<td></td>
<td>725.6</td>
<td>9985.0</td>
<td>827.9</td>
</tr>
<tr>
<td>random</td>
<td></td>
<td>1811.6</td>
<td>1.6</td>
<td>1868.5</td>
</tr>
<tr>
<td>Peano</td>
<td></td>
<td>962.6</td>
<td>1.6</td>
<td>1003.4</td>
</tr>
<tr>
<td>heuristic 24</td>
<td></td>
<td>681.2</td>
<td>129.1</td>
<td>733.6</td>
</tr>
<tr>
<td>heuristic 24</td>
<td></td>
<td>675.5</td>
<td>403.2</td>
<td>716.4</td>
</tr>
<tr>
<td>Sim. Ann.</td>
<td></td>
<td>684.9</td>
<td>7654.0</td>
<td>711.0</td>
</tr>
</tbody>
</table>
4.2. Discussion of the results

The experimental results are given in table II. We will first discuss the results, followed then by a summary of the advantages and disadvantages of each of the methods.

The Peano ordering method yields reasonable results compared to the random ordering but the result is far from optimal. Much better solutions are obtained by explicitly using the road-pattern.

History search performs very well for all maps. We mentioned before that it is hard to obtain a good range for the multiplication factor \( a \), see sec. 3.1.2. Therefore we took different values for \( a \), namely \( a = 0.03, 0.035, \ldots, 0.055 \). Tests showed that the heuristic algorithm showed less fluctuations for a linear expression for the jumping time than for the discontinuous function presented in the appendix.

History search with an additional iterative improvement performed very well, sometimes even more than 5% better than the History search. A disadvantage of this additional improvement is that no polynomial upper bound can be given for the computer time necessary to perform the search. In theory it is possible that this search will cost an exponentially increasing computer time although this is very improbable to occur in practice.

Table II shows Simulated Annealing to perform well. It is a robust algorithm and finds good solutions. The solutions can be made better at the cost of more computer time. The main disadvantage is that Simulated Annealing needs a lot of computer time, while results are of the same quality as a good problem-tailored heuristic algorithms which may be much faster.

5. Conclusions

History search is an effective and efficient heuristic algorithm. Even for the 'hard' Quadratic Assignment Problem presented in this article it yielded
Efficient road-map management for a car navigation system

good results. We state that many other assignment problems can be solved
with the general scheme. One of the major points is the double itemization
in step 1 of the induction. The functions $h_n$ and $g_n$ have to be well chosen.
This can be done experimentally or intuitively.

Simulated Annealing is an effective heuristic algorithm when solutions can
be obtained without computer time restrictions or when there is no insight
in the problem. Good solutions can be obtained at the cost of large com-
puter times. The quality of the technique is strongly depending on the im-
plementation and the problem definition. The most direct implementation
should not always be chosen as we have shown that a more powerful tran-
sition mechanism can yield better results than a simpler one using the same
amount of computer time.

REFERENCES

1) M. L.G. Thoone, L. H. M. E. Driessen, C. A. C. M. Hermus and K. van der Valk,
2) P. D. M. E. Lahaije, Master's Thesis, Eindhoven Technical University, The Netherlands,
1986.
5) P. J. M. van Laarhoven and E. H. L. Aarts, Simulated Annealing: Theory and Appli-
11) R. Laurini, Eurographics, 327 (1985)

Appendix: Model of the access mechanism of a CD-player

Definitions:

$db$ the radial distance from the central point to the first sector on
the disc (in meters).
$de$ the radial distance from the central point to the last sector on
the disc (in meters).
$td$ the distance between the harts of two tracks next to each other
(in meters).
$rv$ the scanning velocity (in meters per second).
$sl$ the average length of one sector (in meters).
$totnrt$ the total number of tracks on the CD-ROM.
$totnrs$ the total number of sectors on the CD-ROM.
the maximal number of tracks that can be overcome by a small jump.  
the time necessary to move the laser over the entire disc (in seconds).
the reading time for the track on which sector s lies (in seconds).
the number of tracks that lie in between the beginning of sector s1 and the beginning of sector s2.

A few values for the parameters for a 60 minute disc are roughly known

db = 0.0250 m.  
de = 0.055 m and max. 0.058 m.  
td = 1.6 \times 10^{-6} m.  
rv = 1.20 to 1.40 m/s, i.e. about 75 sectors per second.  
sl = 0.017333 m. 

totnrt ≈ 18750.  
totnrs ≈ 270000.  
trsj ≈ 256.  
lmed ≈ 1 sec. 

Define the set of sector numbers as \{1, \ldots, totnrs\}. Let s be a sector number, the function \( h(s) \) is defined as

\[
h(s) = \sqrt{(db)^2 + \frac{(s - 1) \cdot sl \cdot td}{\pi}}.
\]  

(A.1)

Let s1 and s2 be sector numbers, then the function \( tinb(s1, s2) \) for the number of tracks in between these two sectors numbers is

\[
tinb(s1, s2) = \frac{1}{td} \cdot |h(s1) - h(s2)|.
\]  

(A.2)

The reading time \( rt(s) \) for the track on which sector s lies, is

\[
rt(s) = \frac{\pi}{rv} (td + 2h(s)).
\]  

(A.3)

The function for the jumping time is a two-argument function and is called \( t_{s1, s2} \) with the dimension of seconds. The jumping time for small jumps, i.e. if \( tinb(s1, s2) \leq trsj \), is estimated at the time needed to scan half a revolution:
Efficient road-map management for a car navigation system

\[ t_{s1, s2} = \frac{rt(s2)}{2}. \quad (A.4a) \]

The jumping time for larger jumps, i.e. if \( t_i b(s1, s2) > trsj \), is

\[ t_{s1, s2} = \frac{t_{ib}(s1, s2)}{totnr} \cdot lmed + X(t_{ib}(s1, s2)) + \frac{rt(s2)}{2}, \quad (A.4b) \]

where \( X \) is a discrete function of the number of tracks that has to be jumped. \( X \) is the sum of the time to decelerate or accelerate the rotation velocity and the matching-time on the subcode and is estimated at

\[ X(d) = \begin{cases} 
0.100 \text{ sec.} & \text{if } 256 < d \leq 2560 \text{ for } 1 \text{ jump} \\
0.180 \text{ sec.} & \text{if } 2560 < d \leq 25600 \text{ for } 2 \text{ jumps}. 
\end{cases} \quad (A.5) \]

The first term in expression A.4b represents the time for the laser to move over a part of the disc. The last term of the sum is the average time needed to place the laser at the beginning of the wanted sector by scanning the intervening date or by a small jump.

Paul D.M.E. Lahaije: Ir. degree (Computer Science). University of Eindhoven. The Netherlands. 1986; Philips Consumer Electronics, Eindhoven. 1986-. His main interests are in the field of databases. He is presently involved in the development of a car information and navigation system.