ELSEVIER

Data & Knowledge Engineering 13 (1994) 243–263

DATA & KNOWLEDGE ENGINEERING

Experiences with EDO: An Evolutionary Database Optimizer

Patrick van Bommel*

Dept. of Information Systems, Faculty of Mathematics and Computing Science, University of Nijmegen, Toernooiveld, 6525 ED Nijmegen, The Netherlands

Received 27 July 1993; revised 22 December 1993; accepted 17 June 1994

Abstract

In this paper we introduce the database design tool EDO: an Evolutionary Database Optimizer. The term 'evolutionary' refers to a basic feature of the tool. After generating an initial pool of preliminary internal representations for a given conceptual data model, EDO allows a database designer to activate evolution strategies, modifying the preliminary internal representations into more desirable ones.

The quality of the internal representations found as yet is used to perform a guided walk through the solution space of alternative internal representations for the conceptual model under consideration. This quality (called fitness) takes into account the expected storage space and the expected average response time of a candidate internal representation.

Keywords: Data models; Database design; Time/space trade-off; Evolutionary search

1. Introduction

This paper focuses on internal representations for conceptual data models, the evolution of these internal representations and the use of evolution in finding efficient representations.

We first give a short explanation of the problem area. The aim of conceptual data models is to specify the object types to be handled by an information system, without considering the efficiency of the resulting system (see e.g. [12]). The aim of internal representations is to implement a given conceptual model in an efficient way.

Obviously, one and the same conceptual data model may have a very large number of alternative internal representations (see e.g. [7]). Each internal representation has two important properties. The first property is the average response time, the second property is the storage space needed for storing the database. Several problems are involved in these properties. As an example, it is well-known that these properties of `Time` and `Space` express

* Email: pvb@cs.kun.nl

conflicting objectives. This is the so-called time/space trade-off. Another problem involves the fact that it is not possible to compare these properties directly: bytes cannot be compared to seconds. This makes the question how to find an appropriate internal representation very hard to answer.

The naive strategy is exhaustive search. After enumerating all alternative internal representations and estimating the properties of `Time` and `Space`, some efficient candidates can be selected. However, for a large and complex conceptual data model the enumeration of all possible internal representations is far from attractive. For this reason existing tools and techniques for database design based on conceptual modelling usually avoid exhaustive search (see e.g. [1, 10, 13, 20, 22]). However, a general and flexible (time/space-driven) mechanism for walking through the solution space of alternative internal representations is not found in the literature. Moreover, no techniques are found for examining the time/space and update/retrieval trade-off for internal representations of a given conceptual data model.

Therefore we have set up a framework for database optimization in which the relevant aspects can be embedded, including logical database design, index selection and physical database design. The kernel of this framework has been worked out in detail, and has been implemented in the prototype EDO which is based on the following basic idea. For a given conceptual data model some arbitrary internal representations are generated. Then these preliminary internal representations can be automatically modified into more desirable ones. In this way a walk through the solution space of alternative internal representations is performed. For a general introduction into this kind of evolution in the context of formal (technical) systems, we refer to [21] and [11]. The underlying theoretical foundations of EDO are found in [5-7, 2, 8].

Compared to traditional database normalization (see e.g. [24]), this approach has several advantages. For example, a walk through the solution space of alternative internal representations can be guided on the basis of the expected quality (`Time` and `Space`) of the internal representations found as yet. Database normalization restricts itself to dependencies, and does not consider the properties of `Time` and `Space` explicitly. Furthermore, the result of adapting (mutating) a preliminary internal representation is guaranteed to implement the same conceptual data model. The connection with the underlying conceptual model thus remains invariant. Also, the problem of selecting an appropriate set of indexes can be embedded in the framework of mutations in a natural way (see e.g. [4]).

The organisation of the paper is as follows. In Section 2 we summarize the underlying framework for database mappings. The next sections show how basic search strategies, such as random search and hill climbing, can be considered within this framework. In Section 3 we focus on random search through the solution space of alternative internal representations for a given conceptual data model. We also present the Status Report and the Evolution Screen of EDO. Section 4 introduces the notion of hill climbing through the solution space. A comparison is made with random search and the time/space trade-off for internal representations is examined.

It will be obvious that random search and hill climbing are *not* the most powerful search strategies. It is stressed here that the aim of this paper is to show how these basic strategies can be applied to database design. The prototype EDO is currently being extended with more powerful techniques (such as Simulated Annealing [19] and Genetic Algorithms [11]). This will be discussed in Section 5, containing conclusions and directions for future research. Appendix A contains an example database which will be used throughout the paper, while Appendix B gives an example cost model.

2. Framework for database mappings

This section introduces a framework for the transformation of data models from the conceptual to the internal level. We focus on the *structure* of data models. Some related aspects, such as transformation of populations and operations, are beyond the scope of this paper. These are treated in [6].

2.1. Conceptual data models

In this section we discuss conceptual data models. We focus on data modelling techniques with an underlying object-role structure (e.g. NIAM [17, 18] and the Binary Relationship Model [16, 23]). A formal definition of these models is found in [5]. The basic notions are summarized as follows.

An object-role data model consists of an information structure $\mathcal{I}$ and a set of constraints $\mathcal{C}$ on the possible populations of the information structure. An information structure consists of the following components:

- $\mathcal{P}$ is a set of *predicators*. A predicator specifies a role played by an object type in a fact type. The associated object type is found by the operator $\mathtt{Base} : \mathcal{P} \rightarrow \mathcal{O}$.
- $\mathcal{O}$ is a set of *object types*.
- $\mathcal{F}$ is a partition of the set $\mathcal{P}$ of predicators. The elements of $\mathcal{F}$ are called *fact types*. Fact types are also objects types: $\mathcal{F} \subseteq \mathcal{O}$. They are also referred to as *composed object types*. The object types in $\mathcal{A} = \mathcal{O} - \mathcal{F}$ are called *atomic object types*. There are two different sorts of atomic object types: entity types ($\mathcal{E}$) and label types ($\mathcal{L}$).
- Sub is a partial order for atomic object types, with the convention that a Sub b is interpreted as: a is a subtype of b. Each element of $\mathcal{A}$ has associated a (unique) top element, its pater familias. It is found by the function $\sqcap : \mathcal{A} \rightarrow \mathcal{A}$.

We call predicators p and q *attached* to each other ($p \sim q$), when $\sqcap(\mathtt{Base}(p)) = \sqcap(\mathtt{Base}(q))$. The fact type that corresponds with a predicator is given by the operator $\mathtt{Fact} : \mathcal{P} \rightarrow \mathcal{F}$, defined by: $\mathtt{Fact}(p) = f \Leftrightarrow p \in f$. The set $\mathcal{H}$ contains all objectifications: $\mathcal{H} = \{p \in \mathcal{P} | \mathtt{Base}(p) \in \mathcal{F}\}$.

As an example, in the information structure in Fig. 12 (see Appendix A) predicator p_1 expresses the role played by object type X_1 in fact type f_1. In this case $\mathtt{Base}(p_1) = X_1$ and $\mathtt{Fact}(p_1) = f_1$.

Using constraints on the possible populations of the information structure, it should be guaranteed that each entity type can be identified by a (combination of) label type(s). Furthermore, constraints are used for specifying more advanced integrity rules. In section 2.5 two basic kinds of constraints are discussed in more detail. For other constraints we refer to [5]. Advanced modelling constructs, such as set types, sequence types, schema types and generalization hierarchies are treated in [15].

2.2. *Profiles*

In order to characterize the environment in which the database under development will operate, the following profiles are used:

- The *access profile* contains information about the (retrieval and update) operations to be performed. Fig. 13 in Appendix A shows an example (read-only) access profile, specifying paths through the information structure from Fig. 12, along with their relative frequency. Each path characterizes a class of conceptual retrieval operations in terms of object types and roles from the information structure. More details about path-based operations for conceptual information structures are found in [9] and [14].
- The *data profile* contains a quantification of the database contents. Fig. 13 in Appendix A shows an example data profile, specifying the relative number of instances for each object type in the information structure from Fig. 12. Usually, the data profile also specifies the size of the lexical objects used for the identification of non-lexical objects.
- The *device profile* contains information about device-oriented aspects, such as the average time needed for processing a single storage unit, the average cost of the storage media to be used, and the average cost of response time.

2.3. *The evolutionary approach to database design*

A conceptual data model may have a very large number of alternative internal representations (see e.g. [7, 6]). As a consequence, the enumeration of all possible internal representations is far from attractive. Therefore, we adopt the transformational approach and apply data structure transformations in so-called evolutionary design strategies (see e.g. [19, 21]). In this section we describe how such strategies can be used for database design.

In each step of the database design process, a preliminary internal representation is modified into a (more desirable) internal representation. The process can be started from an initially generated (set of) internal representation(s). Fig. 1 illustrates this view of the database design process. The initial generation may be guided by techniques discussed in [7].

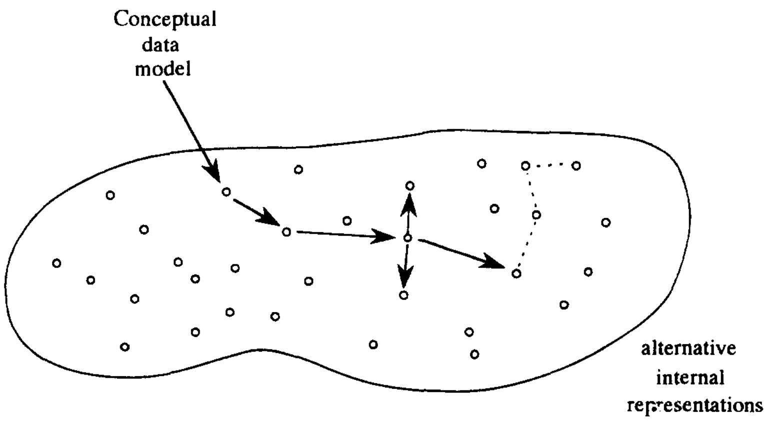


Fig. 1. Walking through the solution space.

In Fig. 1 each small circle represents an internal representation for the conceptual data model under consideration. The search process for an appropriate internal representation is based on evolution operators. Obviously, these operators depend on the specification language used for the internal representations. As an example, in the relational model we may have `Split` and `Join`, for splitting and joining relations respectively.

A generic description of the underlying evolutionary search algorithm is given in Fig. 2. In the next sections this generic description will be made more concrete and it will be illustrated with examples.

In our approach each internal representation specifies a low level data structure for the conceptual data model, using an intermediate specification language. These data structures can, but do not necessarily have to, be interpreted as (nested) relational database schemata with additional indexes. Our internal representations are expressed *in terms of* the conceptual model at hand. Details about these internal representations are found in [7] and [6]. The basic notions are summarized in Section 2.4.

2.4. *Internal representations of object-role models*

In this section we discuss a mechanism for representing object-role information structures internally. Intuitively, such a representation is the result of lifting up certain atomic object types of the information structure and cutting the connections between these object types. Formally, a forest (set of trees) $T = \langle \mathcal{N}, E, \ell \rangle$ is called an internal representation of information structure $\mathcal{I} = \langle \mathcal{P}, \mathcal{O}, \mathcal{F}, \mathtt{Base}, \mathtt{Sub}, \sqcap \rangle$ if it is a labelled graph, satisfying the following wellformedness conditions $t_1, \ldots, t_6$ discussed below [7, 6].

2.4.1. *Nodes of predicators*

t_1 : A node is intended to model a field (or attribute) in the database. Therefore, the set of nodes $\mathcal{N}$ is a partition of $\mathcal{P}$, s.t. predicators in the same node belong to the same subtype hierarchy. In the sequel the following notational convention is used. As all predicators in the same node are type related, the common pater familias in node n may be denoted as $\mathtt{Base}(n)$.

t_2 : Loss of information should be excluded. Therefore all predicators sharing the same node belong to different fact types:

$$\forall_{n \in \mathcal{N}, f \in \mathcal{F}}[|n \cap f| \leq 1]$$

t_3 : Objectifications (elements of $\mathcal{H}$) are treated separately: $\forall_{n \in \mathcal{N}}[|n \cap \mathcal{H}| \leq 1]$

```
Evolutionary search for internal representations:
    Generate initial pool of internal representations;
    while Not ready do
        Modify pool
    end .
```

Fig. 2. Generic description of evolutionary search.

2.4.2. *Labelled edges*

E is a set of edges. The pair $\langle m, n\rangle$ represents an edge from node m to node n.

t_4 : Edges are labelled by fact types in the obvious way, with some care to handle objectifications. The function $\ell : E \rightarrow \mathcal{F}$ assigns $\ell(\langle m, n\rangle) = f$, iff both the source node and the destination node contain a predicator from f:

(1) $n \cap f \neq \emptyset$

(2) $m \cap f = \emptyset \Rightarrow \texttt{Base}(m) = f$

t_5 : Fact types are located around a single parent: $\ell(\langle m_1, n_1\rangle) = \ell(\langle m_2, n_2\rangle) \Rightarrow n_1 = n_2$.

2.4.3. *Completeness and transformations*

t_6 : In order to guarantee that the complete information structure is represented, each predicator must be involved in some edge:

$$\forall_{p \in \mathcal{P}} \exists_{\langle m,n\rangle \in E} [p \in m \cup n]$$

We call a tree representation incomplete if it does not necessarily fulfil the last requirement. Fig. 14 in Appendix A gives an example internal representation for the information structure from Fig. 12.

In [7] it was shown that the following property can be derived from the wellformedness conditions discussed above, in combination with the operator Fact:

$$\text{Each edge } e = \langle m, n\rangle \text{ with } \ell(e) = f \text{ and } \texttt{Base}(m) \in \mathcal{A} \text{ has } |m \cap f| = |n \cap f| = 1 .$$

The corresponding predicator in source node m is unique for node m and is denoted as Anchor(m). The corresponding predicator in destination node n is unique for fact type f and is denoted as Hook(f).

These notions *anchor* and *hook* facilitate the transformation of internal representations into new internal representations. Two basic transformations are used [8, 2]:

(1) Promote(p), promoting the anchor p in node m by swapping edge $\langle m, n\rangle$ to $\langle n, m\rangle$.

(2) Move(p, q), moving Fact(p) with all its descendants to Node(q). Predicators p and q should belong to the same subtype hierarchy. If this is not the case, $\{p\}$ becomes the root of a new tree.

2.5. *Constraints*

2.5.1. *Constraint transformation framework*

Constraints on the possible populations of an information structure are used for specifying integrity rules. Obviously, when considering internal representations of an information structure, these constraints must be translated. Dynamic constraints, used for the exclusion of unwanted transitions, are not considered in this paper.

We describe how constraint translations are embedded in our framework for database mappings. This may happen during four different phases: (1) preprocessing, (2) initial generation, (3) evolution and (4) postprocessing. We consider constraint translations during initial generation and evolution in more detail.

The process of initial generation and evolution can be influenced on the basis of guidance conditions. These guidance conditions may be expressed in terms of structural properties of the internal representations at hand (e.g. size and depth), or in terms of constraint translations (e.g. involving null values and redundancy). Examples are given in [7], where candidates with undesirable properties are excluded from the generation process. Another example involves the evolution of simple internal representations, where each fact type is taken separately, into more complex internal representations where fact types are joined together. In this way specific Normal Forms may be constructed stepwise (e.g. the Optimal Normal Form [17, 18]), such that partly normalized representations are yielded as intermediate solutions. Finally, guidance conditions may also be expressed in terms of storage requirements and average response time of the internal representations, for instance in hill climbing strategies. This will be discussed in later sections.

We discuss two kinds of constraints in more detail: uniqueness constraints and total role constraints. These constraints are of vital importance for the conceptual information structure, since they are used for identification purposes [5, 18]. Furthermore, these constraints are important for internal representations, because several basic properties of these representations can be derived from them (see e.g. [7]).

2.5.2. *Total role constraints*

First we consider total role constraints. A total role constraint for predicator p expresses that each instance of $\mathtt{Base}(p)$ must occur in at least one instance of $\mathtt{Fact}(p)$. For an internal representation containing predicator p this has the following consequences:

(1) If in this representation predicator p is the anchor of a non-leaf node, then predicator p must have a total role constraint. In case this predicator is not total, the internal representation is illegal because it may result in loss of information. Consider for example the situation where predicator p_1 in Fig. 12 is not total. Then there can be instances of fact type f_2 that cannot be represented in the structure given in Fig. 14.

(2) If in this representation predicator p is the hook of a fact type and p does not have a total role constraint, then the subtable corresponding to $\mathtt{Fact}(p)$ is *optional.*

2.5.3. *Uniqueness constraints*

Next we consider uniqueness constraints. The meaning of a uniqueness constraint is similar to the notion of a key (see e.g. [5, 24]). As a consequence, presence or absence of a uniqueness constraint will not cause an internal representation to be illegal. However, a special situation occurs if the hook of a fact type is unique. Then the subtable corresponding to that fact type is flat rather than nested. More details and several examples are found in [7].

2.6. *Fitness of internal representations*

The profiles discussed in Section 2.2 can be used to compute the expected average response time `Time` and the expected storage space requirements `Space` for a given internal representation. In order to interrelate the values of the conflicting criteria `Space` and `Time`, we use the parameters shown in Fig. 3.

Using the parameters from Fig. 3, the fitness (quality) of a given internal representation is defined by:

parameter	explanation
$\beta \in [0,1]$	weight coefficient for storage space
γ_s	the average cost of storage media
γ_t	the average cost of response time

Fig. 3. Parameters for conflicting objectives.

$$\texttt{Fitness} = \frac{1}{\beta \times \gamma_s \times \texttt{Space} + (1-\beta) \times \gamma_t \times \texttt{Time}}$$

The weight coefficient β is used for specifying the relative importance of Time and Space optimization, rather than for representing the trade-off between Time and Space. This trade-off, and the update/retrieval trade-off, is taken care of by the underlying cost model for Time and Space.

A basic cost model for the computation of the properties Time and Space is found in Appendix B. It is stressed that the evolution mechanism for walking through the solution space of internal representations is independent of the underlying cost model for those representations. As a consequence, different cost models may be embedded in the same framework. In order to focus on the functionality of the prototype EDO (see Section 2.7) in this paper, the values of Time, Space and Fitness produced by this tool are expressed in an abstract manner in terms of time units (t.u.), space units (s.u.) and fitness units (f.u.).

Note that the cost model in Appendix B is a *basic* cost model, in which other aspects can be embedded. As an example, consider the enforcement of constraints after updates. In general, constraint enforcement causes update overhead. This can be embedded within the basic cost model by refining the cost d_i of updates (see Appendix B). Other examples are the effect of null values on storage, and indexes invoking trade-offs between retrieval and update times.

2.7. EDO

The framework discussed so far has been implemented in the prototype database design tool *Evolutionary Database Optimizer* (EDO). We shortly summarize the main menu of this interactive tool, allowing a database designer to activate procedures via the following submenus: *Files*, for editing files; *Inspector*, for getting detailed information about the conceptual data model, the current pool of internal representations or the search process until the current moment; *Generator*, for generating an initial pool; *Evolver*, for activating evolution strategies modifying the current pool; *Lister*, for producing output; *Options*, for setting control parameters of the tool.

General information about current parameter settings and pool properties is presented below the main menu in the Status Report. This gives a summary of the information that can be obtained via the submenus *Inspector* and *Options*. In the next sections the Status Report will be discussed in more detail.

3. Random search

This section shows how the basic search strategy, called random search, can be embedded within the framework discussed in Section 2.

3.1. Random search strategy

We describe a simple random search strategy through the solution space of alternative internal representations. The intention of this strategy is as follows. First an initial pool of internal representations is generated. Then this pool is iteratively modified. In a pool modification each element x of the current pool is replaced by a randomly generated mutation y. The algorithm is given in Fig. 4.

Let `PoolSize` be the number of internal representations in a single pool and let `Steps` be the number of evolution steps to be made. Then the complexity of the random search strategy shown in Fig. 4 is given by:

```
PoolSize×(g+f+Steps×(m+f))
```

where `g` is the cost of generating an initial internal representation, `f` is the cost of a single fitness evaluation, and `m` is the cost of a mutation. For generation and mutation, the number of (re)construction steps is linear in the number of predicators. In each step a simple and rather cheap assignation is made (see [2]). On the other hand, fitness evaluation requires a number of multiplications which is at least quadratic in the number of predicators (see the basic cost model in Appendix B). As a result, the formula given above is in accordance with the fact that usually the bottle-neck in the performance of evolutionary search algorithms is fitness evaluation, rather than the actual evolution mechanism (see e.g. [11,21]).

3.2. Initial generation

Before a random search strategy is activated we first generate an initial pool of internal representations, using the *Generator* introduced in Section 2.7. After completing the generation process, the system gives the Status Report shown in Fig. 5.

From the first column of the Status Report in Fig. 5 we conclude that the conceptual model

```
Random search strategy:
    Generate initial pool and evaluate fitness;
    while Not ready do
        for each $x$ in pool do
            Generate $y$ := Mutate ($x$) and evaluate fitness;
            Replace ($x, y$)
        end
    end .
```

Fig. 4. Random search strategy.

Schema:	2	Current fitness:	0.00317	Space weight:	5
Data profile:	1	Max fitness:	0.00317	Pool size:	4
Access profile:	1			Randseed:	1
		Current space:	311		
Object types:	10	Min space:	311	Steps:	30
Fact types:	5	Current time:	320	Display mode:	0
Predicators:	13	Min time:	320	Step mode:	0

Fig. 5. Status report after initial generation.

under consideration is called 'Schema 2'. The profiles used for optimization are 'Data profile 1' and 'Access profile 1'. As the Status Report shows, the conceptual model consists of 10 object types, where 5 object types are composed object types (fact types), with 13 predicators in total.

Next we consider the second column of the Status Report in Fig. 5. In the current pool the minimal space is 311 s.u., the minimal time is 320 t.u., and the resulting highest fitness is 0.00317 f.u. Since this is the initial pool, the best properties in the current pool (e.g. Current fitness) are equal to the best properties found in the evolution process until now (e.g. Max fitness).

In the third column of the Status Report in Fig. 5 we find some system parameters. The importance of storage optimization (Space weight) is 0.5. The number of representations in a single pool (Pool size) is 4, while the number of evolution steps to be made by evolution strategies (Steps) is 30. Also, we find some other parameters (Randseed, Display mode, Step mode) which are outside the scope of this paper.

3.3. *An example random walk*

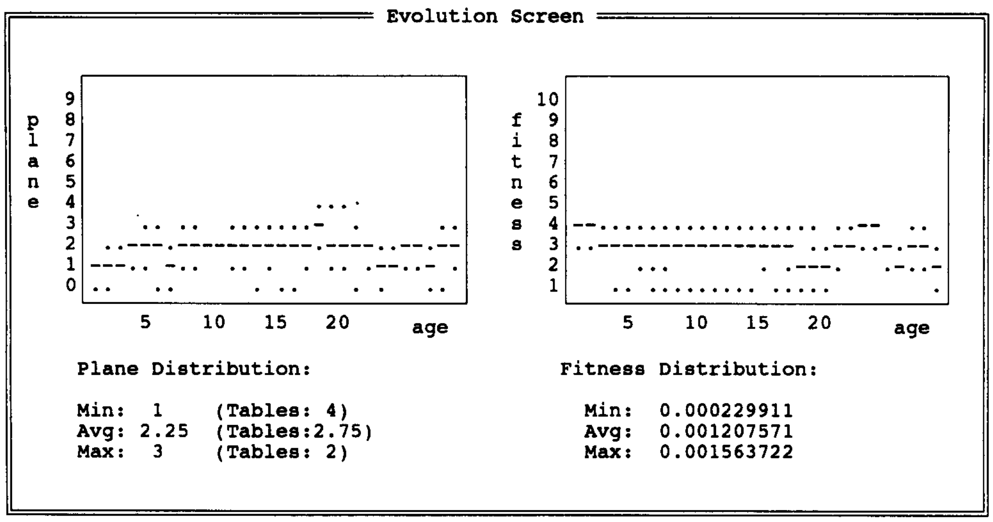
Next we will activate a random walk strategy, using the *Evolver* introduced in Section 2.7. During the search process the Evolution Screen shown in Fig. 6 is constructed.

In the left part of the Evolution Screen in Fig. 6 a graph is given with horizontal axis *age* and vertical axis *plane*. In the right part a graph is given with horizontal axis *age* and vertical axis *fitness*. We first explain the age on the horizontal axes.

Let $\text{Gen}(i)$ be the result after evolution step i. The initial pool of internal representations, denoted as $\text{Gen}(0)$, is produced by the generator. Each next generation $\text{Gen}(i+1)$ is produced by the evolver, and is the result of mutating generation $\text{Gen}(i)$ where $i \geqslant 0$. The age on the horizontal axes corresponds to i.

Next we explain the plane-axis. The solution space $\mathbf{S}$ of alternative internal representations for a given information structure $\mathcal{I}$ with fact types $\mathcal{F}$ consists of $|\mathcal{F}|$ hyperplanes, where a hyperplane $\nu_j \subseteq \mathbf{S}$ is the subset of $\mathbf{S}$ containing the representations with j joins $(0 \leqslant j \leqslant |\mathcal{F}| - 1)$. The numbers on the plane-axis correspond to values of j. More details about these planes are found in [3].

As a consequence, the left figure in the Evolution Screen shows in which parts of the solution space $\mathbf{S}$ the search process is performed. For instance, the pool resulting from the first iteration of the evolver, denoted as $\text{Gen}(1)$, is shown in the first column of graph plane/age. We see that the elements of this pool are in hyperplanes ν_0 and ν_1. If pool $\text{Gen}(i)$ contains an



```
Plane Distribution:                 Fitness Distribution:

Min:  1    (Tables: 4)               Min:  0.000229911
Avg: 2.25  (Tables:2.75)             Avg:  0.001207571
Max:  3    (Tables: 2)               Max:  0.001563722
```

Fig. 6. Evolution screen during random walk.

element from plane v_j, the graph shows a dot '.' in position (i, j). The average plane for a given pool is given by a hyphen '-'.

Next we explain the *fitness* on the second vertical axis. As was mentioned in Section 2.6, each internal representation in S has a certain fitness. The graph fitness/age is expressed as a percentage of an upperbound for the conceptual model under consideration. As an example, pool Gen(1) contains internal representations with a fitness of 30 and 40% of the upperbound, while the average fitness in that pool is 40%.

3.4. *Inspecting the results*

When the evolution process has been completed, there are several ways to examine the results. Obviously, the Evolution Screen itself has given information about the search process. After returning to the main menu further investigation is possible as follows.

Firstly, the Status Report contains new values with respect to the fitness. In the excerpt of the Status Report shown in Fig. 7, we see that the maximal fitness in the current pool is 0.00156 (Current fitness), while the maximal fitness found during the evolution process is 0.00323 (Max fitness). Compared to the original values in Fig. 5 the former property has

Current fitness:	0.00156
Max fitness:	0.00323
Current space:	969
Min space:	309
Current time:	280
Min time:	260

Fig. 7. Excerpt of status report after evolution.

worsened, while the latter property has improved. This is a typical feature of random search. Similar properties with respect to Space and Time are given. Note that an improvement is indicated by an increasing fitness and by a decreasing Space and Time.

As was mentioned before, the Status Report only gives global information about current pool properties. Therefore, a second possibility for further investigation is provided by the *Inspector* (see Section 2.7). The Pool Inspector gives more detailed information about the current pool, while the Evolution Inspector gives more information about the evolution process which resulted in the current pool.

4. Hill climbing

4.1. Hill climbing strategy

In this section we consider the possibility of hill climbing (see e.g. [21]) through the solution space of alternative internal representations. We describe a simple hill climbing strategy and formulate some basic properties.

The intention of the climbing algorithm is as follows. First an initial pool of internal representations is generated. Then this pool is iteratively modified. In a pool modification each element x of the current pool may be replaced by a randomly generated mutation y of x. Whether x will actually be replaced by y depends on their fitness. In this way the hill climbing nature is guaranteed. The algorithm is given in Fig. 8.

Next we formulate some basic properties of the algorithm in Fig. 8, involving the fitness of the internal representations found during a search process:

(1) In each stage of the search process the current pool contains the best internal representation found as yet:

$$\forall_{i \geq 0}[\texttt{MaxFitness}(i) = \texttt{CurFitness}(i)]$$

where CurFitness(i) and MaxFitness(i) express the properties introduced in Section 3.2 with respect to pool Gen(i).

```
Hill climbing strategy:
  Generate initial pool and evaluate fitness;
  while Not ready do
    for each $x$ in pool do
      Generate $y$ := Mutate ($x$) and evaluate fitness;
      if Fitness ($x$) $\leq$ Fitness ($y$)
      then Replace ($x, y$)
      end
    end
  end .
```

Fig. 8. Hill climbing strategy.

(2) Therefore, the best representation in the current pool cannot be worse than the best representation in previous pools:

$$\forall_{j \geq 0} \forall_{0 \leq i \leq j} [\texttt{CurFitness}(i) \leq \texttt{CurFitness}(j)]$$

(3) As a direct consequence, this property also holds for the best representation found as yet:

$$\forall_{j \geq 0} \forall_{0 \leq i \leq j} [\texttt{MaxFitness}(i) \leq \texttt{MaxFitness}(j)]$$

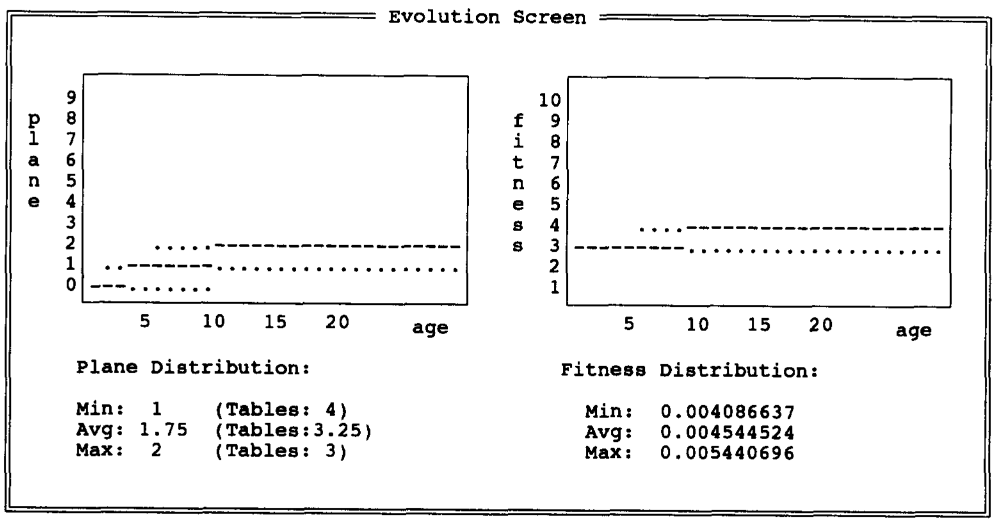
The computational complexity of the hill climbing algorithm shown in Fig. 8 is basically the same as the complexity of the random search strategy from Section 3. If however a 'steepest ascent' hill climber is used, each step evaluates all possible mutations, rather than a single one. Then the complexity $\texttt{Steps} \times \texttt{(m+f)}$ becomes $\texttt{Steps} \times N \times \texttt{(m+f)}$, where N is the maximum number of neighbours within the solution space. In this paper we do not require a hill climber to find the steepest gradient.

4.2. An example hill climbing walk

In this section we will make an example hill climbing walk through the solution space of alternative internal representations. We use the information structure from Fig. 12 and the access profile and data profile from Fig. 13 (see Appendix A). In this example run, the weight coefficient for storage space is set to 0.3.

After generating an initial pool of internal representations using the *Generator*, a hill climbing strategy is activated using the *Evolver*. During the search process the Evolution Screen shown in Fig. 9 is constructed.

The graph fitness/age in Fig. 9 clearly shows the climbing nature. From the fitness distribution it can be concluded that the maximal fitness that was found in the search process is



```
Plane Distribution:                   Fitness Distribution:

Min:  1     (Tables: 4)                Min:  0.004086637
Avg: 1.75   (Tables:3.25)              Avg:  0.004544524
Max:  2     (Tables: 3)                Max:  0.005440696
```

Fig. 9. Evolution screen during hill climbing walk.

0.00544 f.u. We shortly examine the corresponding internal representation. In the optimal internal representation that was found, fact type f_3 is represented in a separate nested table with key p_8:

X_2 of p_8	f_3	
	X_1 of p_6	X_2 of p_7

Fact types f_1 and f_2 are represented in a common table with key p_3 (obtained from fact type f_1). Fact type f_2 is nested within the subtable for fact type f_1.

X_3 of p_3	f_1		
	X_2 of p_2	X_1 of p_1, p_4	f_2
			X_2 of p_5

Fact types f_5 and f_4 are represented in a common table with key p_{12} (obtained from fact type f_5). Fact type f_4 is nested within the subtable for fact type f_5.

X_2 of p_{12}	f_5		
	X_1 of p_{11}	X_3 of p_{10}, p_{13}	f_4
			X_2 of p_9

Note that these nested tables are expressed in terms of the information structure from Fig. 12. Note furthermore that these tables correspond to the internal representation in Appendix A (see also Section 2.4).

As was mentioned at the beginning of this section, in this example hill climbing walk the weight coefficient for storage space was 0.3. Therefore, the highest priority is given to the access profile. It is easily checked that the internal representation discussed above supports the access profile from Fig. 13 appropriately, e.g. in terms of the number of inter-table accesses needed for that profile, without going to the extreme of representing the complete information structure in a single table.

The exact values of the properties of `Space` and `Time` for this internal representation are given in Section 4.4. Also, a comparison with other values of the weight coefficient for storage space will be given. We now first make a comparison with a random walk.

4.3. Comparison with random search

In this section we make a random walk for the same information structure under the same conditions. The resulting Evolution Screen is shown in Fig. 10.

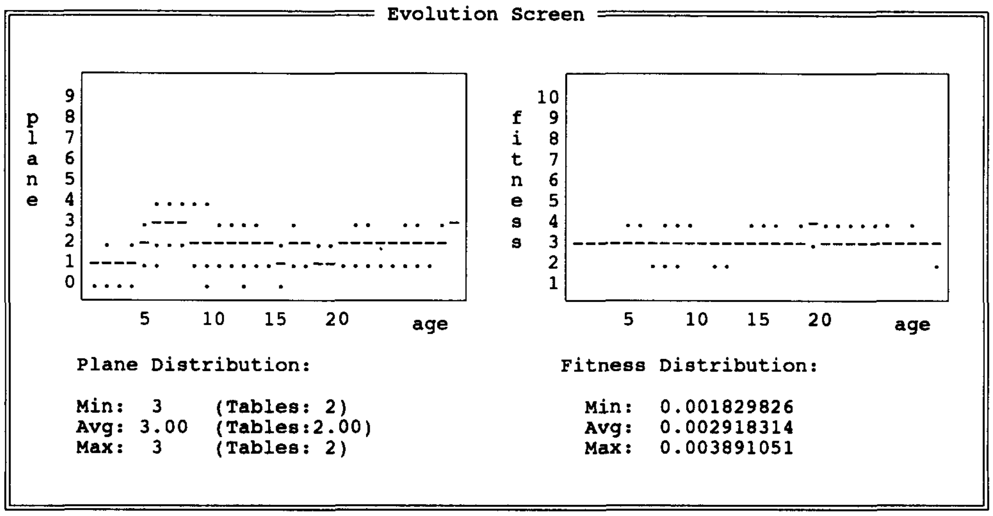
From the graph plane/age in Fig. 10 it can be concluded that this random walk has searched in a larger part of the solution space (compared to the graph plane/age in Fig. 9). This is one of the advantages of random search strategies. For this reason these strategies may blindly find rather good solutions.

However, the graph fitness/age shows that the maximal fitness is found at a later stage (compared to the graph fitness/age in Fig. 9). Furthermore, the Status Report which is shown after returning to the main menu indicates that the maximal fitness found during the search process is lower than the maximal fitness that was found by the hill climbing strategy in the previous section.

4.4. Time/space trade-off

In this section we examine the time/space trade-off for internal representations. In EDO this trade-off can be experimentally examined by varying the weight coefficient for storage space (β) from 0 to 1. As a result of this variation, a small subset of the solution space of internal representations can be placed in a graph with axes time/space (see below).

The experiment has the following setup. We use the information structure and the profiles given in Appendix A. We make a number of runs using the hill climbing strategy, varying β from 0 to 1. In each case the optimal internal representation is recorded after 30 and 60 evolution steps. Each run is initiated from the same pool of internal representations. The optimal internal representation in this initial pool is indicated by `Steps=0` in Fig. 11.



```
Plane Distribution:                 Fitness Distribution:

Min:  3    (Tables: 2)               Min:  0.001829826
Avg: 3.00  (Tables:2.00)             Avg:  0.002918314
Max:  3    (Tables: 2)               Max:  0.003891051
```

Fig. 10. Evolution screen during random walk.

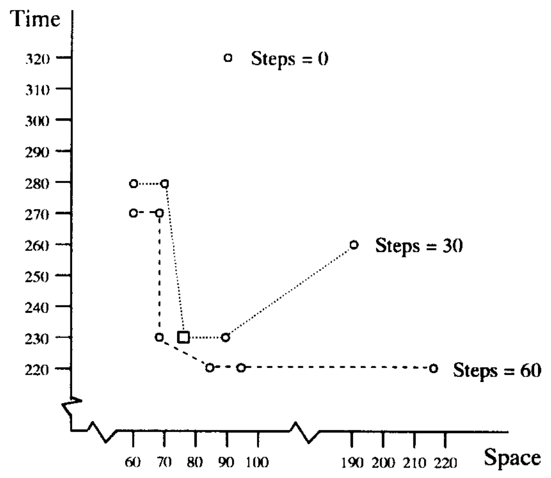


Fig. 11. Time/space trade-off.

After 30 evolution steps with different values of β, five different optima are found. These optima are indicated by the line `Steps=30` in Fig. 11. The optimal representation discussed in Section 4.2 is depicted as a small square.

The optima that are found after 30 more evolution steps are indicated by the line `Steps=60` in Fig. 11. In accordance with intuition, this line is closer to the origin of the graph than the line indicated by `Steps=30`.

We conclude this section with the following remark, concerning the optimum (after 30 steps) which ignores space optimization ($\beta = 0$). In Fig. 11 this so-called egoistic optimum with respect to time is given in position (190,260). This optimum is dominated by other optima with better properties for `Time` as well as `Space`, which is contrary to expectation because the access profile does not contain updates and indexes are not considered. However, after 30 more evolution steps this egoistic optimum has a value for `Time` which is not dominated by any of the other solutions. As a consequence, the line indicated by `Steps=60` reflects a proper convex Pareto-Optimal front as one would expect in a multicriteria optimization problem with conflicting objectives (see also [11]).

5. Conclusions and future research

In this paper we introduced the prototype for evolutionary database design EDO. For a given conceptual data model, this interactive tool allows a database designer to generate preliminary internal representations and to make these representations evolve into more desirable ones. The fitness of internal representations is estimated on the basis of profiles, characterizing the environment in which the database under development will operate.

Compared to existing tools and techniques for database design based on conceptual modelling (see e.g. [1, 10, 13, 20, 22]), our tool has several interesting features. The internal representations manipulated by EDO are expressed *in terms of* the conceptual model under consideration. As a consequence, it is easily checked that the result of mutating a preliminary internal representation is still implementing the same conceptual model. While using this flexible mechanism for walking through the solution space of alternative internal representations, a designer may ask for guidance based on the properties of `Time` and `Space` of the representations found as yet. A multicriteria fitness function enables the designer to examine the time/space trade-off for the given database application.

In the current version of EDO two basic kinds of search strategies have been implemented. Firstly, by means of random search it is possible to walk through many different parts of the solution space. Secondly, by means of hill climbing search it is possible to walk through the solution space under the condition that a next step is required to be an improvement. This can be considered a simulation of a database design process. However, hill climbing strategies have the disadvantage that they do *not* accept worsening situations. As a consequence, these strategies may be trapped in local optima.

There are several possibilities to overcome the problem of local optima. The first possibility is to start searching from different parts of the solution space. The *Generator* mentioned in Section 2.7 is EDO's facility that can be used for this purpose. Techniques discussed in [7] may be used here. The second possibility is to perform a few random steps between two sequences of hill climbing steps. The *Evolver* (Section 2.7) is EDO's facility that can be used for this purpose.

A third possibility to avoid local optima is relaxing the requirement that a hill climber excludes worsening fitness. The deterministic replacement mechanism in Fig. 8 then becomes a probabilistic mechanism: if a mutation results in a representation with a certain fitness, this mutation is performed with a probability which is proportionate to that fitness. Such probabilistic replacement mechanisms are applied in e.g. Simulated Annealing and Genetic Algorithms ([11, 19] and [21]).

The latest version of EDO also supports the so-called Steepest Ascent strategy (see also Section 4.1), while the Simulated Annealing approach is currently being implemented.

Another future extension involves the notion of distance between internal representations, along with an underlying theory for convergence of search strategies for our solution space. This will be based on Markov processes. Moreover, in order to further attune EDO to real life database applications, a project consisting of case studies and extensive experiments in cooperation with a large organization is in its early stages.

Acknowledgement

The author wishes to thank Theo van der Weide, Eckhard Falkenberg, Arthur ter Hofstede and Carlos Lucasius for helpful discussions and constructive comments on this research. Furthermore, he gratefully acknowledges the advice of Hans-Paul Schwefel (University of Dortmund, Germany). The remarks of the anonymous referees resulted in many improvements.

Appendix A: Example

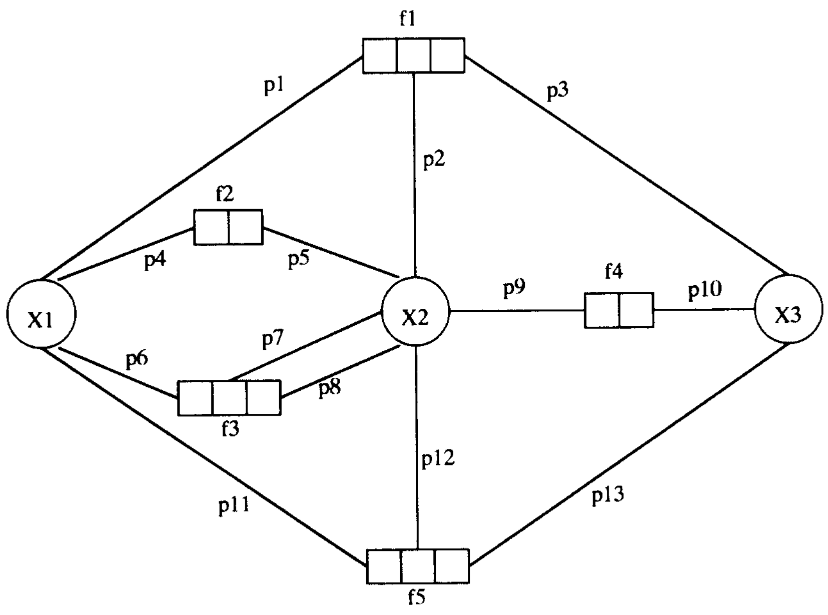


Fig. 12. Information structure.

Relative frequency	Path expression
0.10	$p_4, p_5, p_8, p_6, p_{11}, p_{12}$
0.50	$p_9, p_{10}, p_{13}, p_{12}$
0.40	p_5, p_4, p_1, p_2

Type:	x_1	x_2	x_3	f_1	f_2	f_3	f_4	f_5
Relative nr. of instances:	3	2	1	1	2	3	1	2

Fig. 13. Example access profile and data profile.

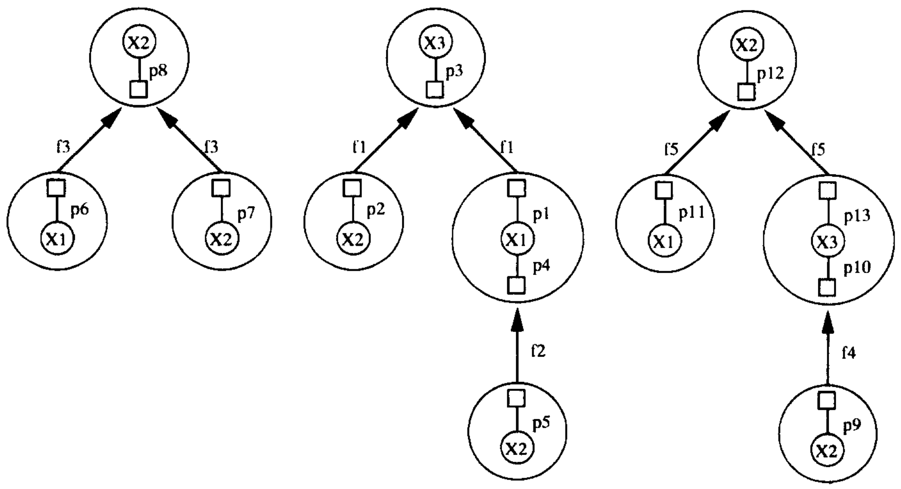


Fig. 14. Candidate internal representation.

Appendix B: Cost model

This appendix summarizes the basic cost model. Let $\mathcal{I}$ be an information structure and let D be a data profile, specifying the number of instances of entity types and fact types in $\mathcal{I}$. Furthermore, D specifies the size of the standard name for each entity type (standard names are *lexical* label types, in contrast with *non-lexical* entity types). Finally, let A be an access profile, specifying the expected pattern of retrieval and update requests to be performed on $\mathcal{I}$.

We first consider the expected storage requirements $\mathtt{Space}(\mathcal{T}, D)$ for candidate internal representation $\mathcal{T}$:

$$\mathtt{Space}(\mathcal{T}, D) = \sum_{n \in \mathcal{R}} \mathtt{DataSpace}(n, D) + \sum_{n \in \mathbf{I}} \mathtt{IndexSpace}(n, D) \tag{1}$$

where $\mathcal{R}$ and **I** contain all root nodes and indexed nodes, respectively. The space needed for an index on node n is restricted by the number of instances in the population of that node:

$$\mathtt{IndexSpace}(n, D) = |\mathtt{Pop}(\mathtt{Base}(n))| \times \mathtt{Size}(\mathtt{Base}(n)) \tag{2}$$

where $\mathtt{Base}(n)$ is the entity type in node n and $\mathtt{Size}(x)$ gives the size of the standard name for entity type x. Next we consider the computation of $\mathtt{DataSpace}(n, D)$. Let n be a node with incoming edges labelled by $f_1, \ldots, f_k$ (see Fig. 15). If $k = 0$, then node n is a leaf node. Each fact type f_i is represented using the nodes $m_{i1}, \ldots, m_{il_i}$ being a child of node n. Then the expected storage requirements for node n is recursively defined by:

$$\mathtt{DataSpace}(n, D) = |\mathtt{Pop}(\mathtt{Base}(n))| \times \left[\mathtt{Size}(\mathtt{Base}(n)) + \sum_{\substack{1 \le i \le k \\ 1 \le j \le l_i}} |\mathtt{Pop}(f_i)| \times \mathtt{DataSpace}(m_{ij}, D) \right] \tag{3}$$

In this formula $|\mathtt{Pop}(\mathtt{Base}(n))|$ is the maximum number of tuples in node n. Each tuple t has exactly one root value in n and at most $|\mathtt{Pop}(f_i)|$ sub-tuples in child m_{ij}, nested within t (where $1 \le i \le k, 1 \le j \le l_i$). The size of the root value of entity type x is given by $\mathtt{Size}(x)$.

Next we discuss the processing of access profiles. An access profile $A = \langle R, U \rangle$ consists of a $p \times p$ matrix R, modelling the expected pattern of retrieval requests, and a p-vector U,

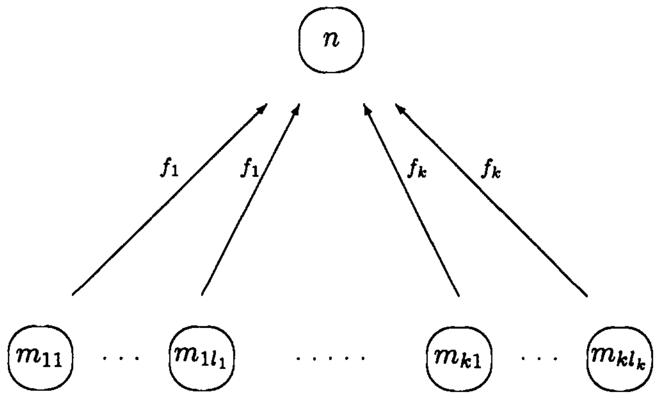


Fig. 15. General structure of tree.

modelling the expected pattern of update requests (p is the number of predicators contained in $\mathcal{I}$). Entry U_i gives the expected number of updates for predicator p_i per time unit (e.g. per day). Entry R_{ij} gives the expected number of accesses to be performed from predicator p_i into predicator p_j. Then, the average response time of representation $\mathcal{T}$ based on profiles D and A is defined as:

$$\mathtt{Time}(\mathcal{T}, D, A) = \sum_{i,j \leq p} c_{ij} \times R_{ij} + \sum_{i \leq p} d_i \times U_i \tag{4}$$

where c_{ij} is the cost for an access from $\mathtt{Node}(p_i)$ into $\mathtt{Node}(p_j)$, and d_i is the cost for updating $\mathtt{Node}(p_i)$.

For the computation of c_{ij} and d_i several approaches can be used. Usually a detailed estimation is given in terms of memory blocks, pages, number of I/O's, buffer size, etc. In this paper a global cost estimation in terms of nodes of the internal representation will suffice. Let ξ be the average time needed for processing a single storage unit (e.g. a byte). Furthermore, let $S(y)$ be a shorthand for $\mathtt{Size}(\mathtt{Base}(p_y)) \times |\mathtt{Pop}(\mathtt{Base}(p_y))|$. Then, the cost c_{ij} for access from $\mathtt{Node}(p_i)$ into $\mathtt{Node}(p_j)$ is estimated as follows:

$$c_{ij} = \begin{cases} \xi \times S(i) & (\mathtt{Node}(p_j) \in \mathbf{I}) \\ \xi \times S(i) \times S(j) & (\text{otherwise}) \end{cases} \tag{5}$$

Next we consider the cost d_i for updating predicator p_i. In the absence of an index on $\mathtt{Node}(p_i)$, the number of changes is restricted by the number of instances in $\mathtt{Base}(p_i)$. An index on $\mathtt{Node}(p_i)$ will double this, as a result of index maintenance:

$$d_i = \begin{cases} \xi \times S(i) & (\mathtt{Node}(p_i) \not\in \mathbf{I}) \\ 2 \times \xi \times S(i) & (\text{otherwise}) \end{cases} \tag{6}$$

References

[1] A. Amikam, On the automatic generation of optimal internal schemata, *Informat. Syst.* 10(1) (1985) 37–45.

[2] P. van Bommel, A randomised schema mutator for evolutionary database optimisation, *Australian Comput. J.* 25(2) (May 1993) 61–69.

[3] P. van Bommel, Database design modifications based on conceptual modelling, in H. Kangassalo and H. Jaakkola, eds, *Information Modelling and Knowledge Bases V*, Amsterdam, The Netherlands, 1993 (IOS Press, in press).

[4] P. van Bommel, Implementation selection for object-role models, in: R. Meersman and T.A. Halpin, eds, *Int. Conf. on Object-Role Modelling* (*ORM-1*), Townsville, Australia (July 1994).

[5] P. van Bommel, A.H.M. ter Hofstede and Th.P. van der Weide, Semantics and verification of object-role models, *Informat. Syst.* 16(5) (Oct. 1991) 471–495.

[6] P. van Bommel, Gy. Kovács and A. Micsik, Transformation of database populations and operations from the conceptual to the internal level, *Informat. Syst.* 19(2) (1994) 175–191.

[7] P. van Bommel and Th.P. van der Weide, Reducing the search space for conceptual schema transformation, *Data & Knowledge Eng.* 8 (1992) 269–292.

[8] P. van Bommel and Th.P. van der Weide, Towards database optimization by evolution, in: A.K. Majumdar and N. Prakash, eds, *Proc. Int. Conf. on Information Systems and Management of Data* (*CISMOD 92*), Bangalore, India (July 1992) 273–287.

[9] O.M.F. De Troyer, R. Meersman and F. Ponsaert, RIDL User Guide. Research report, International Centre for Information Analysis Services, Control Data Belgium, Inc., Brussels, Belgium, 1984.
[10] O.M.F. De Troyer, R. Meersman and P. Verlinden, RIDL* on the CRIS case: A workbench for NIAM, in: T.W. Olle, A.A. Verrijn-Stuart and L. Bhabuta, eds, *Computerized Assistance during the Information Systems Life Cycle* (Amsterdam, The Netherlands, 1988. North-Holland/IFIP) 375–459.
[11] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning* (Addison-Wesley, Reading, MA, 1989).
[12] J.J. van Griethuysen, ed., *Concepts and Terminology for the Conceptual Schema and the Information Base*, Publ. nr. ISO/TC97/SC5-N695 (1982).
[13] T.A. Halpin, WISE: a Workbench for Information System Engineering, in: V.-P. Tahvanainen and K. Lyytinen, eds, *Next Generation CASE Tools*, vol. 3 of *Studies in Computer and Communication Systems* (IOS Press, 1992) 38–49.
[14] A.H.M. ter Hofstede, H.A. Proper and Th.P. van der Weide, Formal definition of a conceptual language for the description and manipulation of information models, *Informat. Syst.* 18(7) (1993) 489–523.
[15] A.H.M. ter Hofstede and Th.P. van der Weide, Expressiveness in conceptual data modelling, *Data & Knowledge Eng.* 10(1) (Feb. 1993) 65–100.
[16] I. Kobayashi, Classification and transformations of binary relationship relation schemata, *Informat. Syst.* 11(2) (1986) 109–122.
[17] C.M.R. Leung and G.M. Nijssen, Relational database design using the NIAM conceptual schema, *Informat. Syst.* 13(2) (1988) 219–227.
[18] G.M. Nijssen and T.A. Halpin, *Conceptual Schema and Relational Database Design: A Fact Oriented Approach* (Prentice-Hall, Sydney, Australia, 1989).
[19] C.R. Reeves, ed. *Modern Heuristic Techniques for Combinatorial Problems* (Blackwell Scientific Publications, Oxford, UK, 1993).
[20] N. Rishe, A file structure for semantic databases, *Informat. Syst.* 16(4) (1991) 375–385.
[21] H.P. Schwefel, *Numerical Optimization of Computer Models* (John Wiley and Sons, New York, 1981).
[22] P. Shoval and M. Even-Chaime, ADDS: A system for automatic database schema design based on the binary-relationship model, *Data & Knowledge Eng.* 3(2) (1987) 123–144.
[23] P. Shoval and S. Zohn, Binary-Relationship integration methodology, *Data & Knowledge Eng.* 6(3) (1991) 225–250.
[24] J.D. Ullman, *Principles of Database and Knowledge-base Systems*, vol. I (Computer Science Press, Rockville, Maryland, 1989).

P. van Bommel received his masters degree in Computer Science from the University of Nijmegen, Netherlands, in 1990. He is now a junior researcher at the Department of Information Systems at the University of Nijmegen. His main research interests include equivalence and mapping of data models, the transformation of conceptual data models into efficient internal representations, and the corresponding transformation of database populations, operations and constraints.