PDF hosted at the Radboud Repository of the Radboud University Nijmegen

The following full text is a publisher's version.

For additional information about this publication click this link.
http://hdl.handle.net/2066/28417

Please be advised that this information was generated on 2017-11-06 and may be subject to change.
Database design by computer-aided schema transformations

by Patrick van Bommel

The focus of the paper is schema transformation during the development of an information system. A framework is described for conversion and transformation of conceptual (semantic) data models and their internal (machine-oriented) representations. This framework allows us to 'walk' through the solution space of candidate internal representations for a given conceptual data model. This walk may be randomised or performance-driven, where storage requirements and average response times are combined in a multi-objective fitness function. Furthermore, a wide variety of control parameters may be embedded, such as preferences for database table size, absence of data redundancy or absence of optional database fields. Basic experimental results produced by a prototype convertor/transformer are presented, including deviations from the standard optimal normal form for databases.

1 Introduction

1.1 Background

In this paper, we combine two different aspects of information systems development; schema transformation and performance engineering. Recent studies show that the field of performance engineering in the context of information systems development is still quite open [1]. As performance engineering aims at predicting and improving the performance of applications [1], schema transformations may be applied here.

As an example, when designing complex database structures, the selection of an appropriate implementation may be expressed in terms of structural transformations leading to performance improvement. This, however, is not fully recognised in current computer-aided software engineering tools, because usually only a standard structure is generated (e.g. optimal normal form [2, 3]). Therefore, we have designed and implemented a framework for conversion and transformation of database structures, which provides a flexible search mechanism through the solution space of all possible structures for a given database application.

The entire framework is described elsewhere [4]. The main difference to related work [2, 3, 5-7] is that transformations may be probabilistically guided, for example on the basis of a multi-objective fitness function or on the basis of other structural properties. This enables us to systematically examine properties of the solution space at hand, such as the time/space trade-off and the update/retrieval trade-off for a given application. We first provide a simple example.

1.2 Schema transformations during database design

Here we present a simple example in order to show how schema transformations can be used during the selection of efficient implementations for conceptual data models. Response time, as well as storage space, is considered.

Usually, the process of database design is initiated from a conceptual (semantic) data model resulting from the information analysis phase [2, 8]. Suppose we have a machine-oriented representation (also called internal view) of the conceptual model under consideration. When this internal representation \( x \) is used for implementation, the resulting system needs a certain amount of storage space \( s_x \), and it demonstrates a certain average response time \( f_x \). At present, not consider the computation of \( s_x \) and \( f_x \).

Next we consider the transformation of representation \( x \) into a new representation \( y \). We are especially interested in transformations presenting the correctness with respect to the conceptual data model at hand. The resulting representation \( y \) again specifies a complete and correct internal view of the same conceptual model. Obviously, representation \( y \) may require a different amount of storage space than representation \( x \). The same holds for the average response time. If representation \( y \) manifests an improvement on both resources, \( y \) is called more optimal than \( x \). This situation is shown in Fig. 1. As \( s_y < s_x \) and \( f_y < f_x \), the result of the transformation is more optimal than the input of the transformation.

During the database design process, the above-mentioned transformation \( x \rightarrow y \) is a step in the correction
direction. However, we are also interested in other kinds of transformations. Suppose, for example, that representation $y$ is transformed into representation $z$, with storage requirements $s_y$ and average response time $t_y$ (see Fig. 1). Representation $z$ is not more optimal than representation $y$ because $s_y > s_z$. However, as $t_y < t_z$, this representation is an alternative for representation $y$. Furthermore, representation $z$ still is more optimal than representation $x$. Note that representation $z'$ with $t_y < t_z$ and $s_y < s_z$ is also interesting. A preference for transformation $y \rightarrow z$ or $y \rightarrow z'$ will obviously depend on the requirements of the particular application environment.

The above example indicates how schema transformations can be used for finding 'good' internal representations of a conceptual data model. To be able to take a large number of steps through the solution space of design alternatives, it is necessary to computerise the transformation process. This has resulted in the EDO tool, allowing transformations to be randomised or probabilistically guided on the basis of multi-objective cost evaluations. In terms of average response time and storage requirements, or other control parameters such as preferences for database table size, absence of data redundancy, or absence of optional database fields.

## 2 Prototype converter/transformer

In this section we introduce the Evolutionary Database Optimizer (EDO) [9] tool, a prototype for conversion and transformation of data models. The use of this tool during performance engineering is illustrated later.

The intention of EDO is as follows. When converting a conceptual data model into an internal representation, we are confronted with the problem that the same conceptual model may have a very large number of correct internal representations [10]. Database design then deals with selecting an appropriate internal representation. Sometimes this is called implementation selection. In each step of the database design process, a preliminary internal representation is modified into a new internal representation. The process can be started from an initially generated (set of) internal representation(s). Fig. 2 illustrates this view of the database design process.

Each small circle in Fig. 2 is an internal representation for the conceptual data model under consideration. In EDO, the search for an appropriate internal representation is based on transformations (also called mutation operations). A generic description of evolutionary algorithms is given in Fig. 3. We have previously preserved an elaborate example in the context of database design [11].

In our approach, each internal representation specifies a low-level data structure for the conceptual data model. These data structures are expressed in terms of the conceptual model in an intermediate specification language. Each structure specified in this language can be interpreted in different ways, for example, as relational, hierarchical or network database schemata, with additional indices [12]. The transformations in EDO are defined for the intermediate data structures, rather than for specific interpretations. As a consequence, the transformation process illustrated in Fig. 2 is valid for any interpretation. However, choosing a specific interpretation is necessary for performance evaluations (during the transformation process), and for consistency checks and constraint mappings (after completion of the transformation process). A detailed description of the intermediate data structures in EDO is beyond the scope of this paper [10, 13].

Fig. 4 presents the architecture of EDO. First, the converter generates internal representations for the conceptual model under consideration. Next, the evolver modifies the current pool of internal representations. These modules may be activated interactively via a main menu, containing the following possibilities: files, for editing files; inspector, for obtaining 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 evolutionary algorithms 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 so-called status report. This provides a summary of the information that can be obtained via the inspector and options.

### Evolutionary algorithm for internal representations:

- Generate initial pool of internal representations;
- while Not ready do
- Modify pool by applying transformations;
- Select candidates

end.

---

Fig. 3 Generic description of evolutionary algorithm

Software Engineering Journal July 1995
3 Underlying parameters

In this Section, we discuss several underlying parameters of the transformation process. First, we address properties of the conceptual data model. We then consider the performance of alternative implementation structures, and discuss properties of the design process.

3.1 Properties of conceptual data model

We focus on data modelling techniques with an underlying object-role structure, e.g. ER [14], NAM [2, 3, 15] and the Binary Relationship Model [16, 17]. A formal definition of these models is found elsewhere [18, 19]. The basic notions are informally summarised below.

An object-role data model consists of an information structure and a set of constraints on the possible populations of the information structure. The central notion is the predicator [18], defined as a role played by an object type in a fact type. An object type corresponds to a 'thing' from the real or abstract world under consideration, whereas a fact type corresponds to an elementary statement expressing a relationship between object types [15].

As an example, in the information structure in Fig. 5, predicator \( p_1 \) is the role played by object type \( X_1 \) in fact type \( F_1 \). Further details about object-role information structures, populations and constraints can be found elsewhere [18, 19].

Next we consider the quantification of properties of information structures. It is clear that the number of atomic object types \( |\mathcal{O}| \), the number of fact types \( |\mathcal{F}| \), and the number of predicators \( |\mathcal{P}| \) provide an indication of the complexity of the conceptual model. Furthermore, these quantities can be used for several other purposes. As an example, the number of nested relational internal representations for a given conceptual model is easily shown to be exponential to the number of fact types.

As a global measure for the size of the fact types in a conceptual model, we use the average number of predicators per fact type:

\[
N_{\text{pr}} = \frac{|\mathcal{P}|}{|\mathcal{F}|}
\]

As a measure for the compactness of the information structure, we use the average number of predicators per atomic object type:

\[
N_{\text{pr}} = \frac{|\mathcal{P}|}{|\mathcal{O}|}
\]

The above measures may be used as a parameter in experiments, in order to obtain an insight into the behaviour of (randomised) transformations. In Section 4.2, we discuss an experiment concerning \( N_{\text{pr}} \).

Note that compactness of information structures is closely related to homogeneity of fact types. A fact type is called (partly) homogeneous if it contains roles played by the same object type (e.g. fact type \( F_1 \) in Fig. 5). We do not elaborate on this further in this paper. Some typical examples of homogeneous fact types are found elsewhere [18, 20].

3.2 Performance evaluation of alternative database designs

In order to characterise 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.

As a measure for the compactness of the information structure, we use the average number of predicators per atomic object type:

\[
N_{\text{pr}} = \frac{|\mathcal{P}|}{|\mathcal{O}|}
\]
Another problem is the fact that Time and Space are non-commensurable. Thus, it is not possible to compare these objectives.

In a data profile, several types of quantitative descriptors may be specified. The following possibilities are distinguished [21]:

- descriptors of central tendency such as mode, mean and median;
- descriptors of dispersion such as range (maximum and minimum), variance and standard deviation;
- descriptors of size such as the number of instances (cardinality) and the number of distinct values;
- descriptors of frequency distribution such as normality, uniformity, and value intervals and counts.

A device profile typically characterises the target DBMS. This may include the kinds of indices the DBMS supports and the query optimisation strategies that are applied.

The profiles introduced above can be used to estimate the storage requirements Space and the average response time Time for a candidate internal representation. It is well known that the properties of Time and Space express conflicting, rather than co-operative, objectives. This results in the so-called time/space trade-off. Another problem is the fact that Time and Space are non-commensurable. Thus, it is not possible to compare these objectives directly; bytes cannot be compared to seconds. This complicates the question of how to find an appropriate internal representation.

One way of treating optimisation problems with more than one objective is scalarisation, also called weighting of several objectives. This and other approaches in the context of database optimisation are discussed elsewhere [4]. Scalarisation is considered in more detail below.

In scalarisation [22] the objectives for optimisation are combined in a single fitness function. For time/space optimisation this leads to the following:

\[ \text{Fitness}(T) = \beta \times \text{Time}(T) + \gamma \times \text{Space}(T) \]

A typical approach to specifying a scalar function is to compute a weighted sum. Let \( k_1, k_2 \geq 0 \) be weight coefficients. The fitness function is then defined as follows:

\[ \text{Fitness}(T) = \frac{1}{k_1 \times \text{Time}(T) + k_2 \times \text{Space}(T)} \]

An alternative for the weighted sum given above is to multiply Time and Space. Then the coefficients \( k_1 \) and \( k_2 \) lose their importance. For the weighted sum, we consider the choice of \( k_1 \) and \( k_2 \) in more detail.

On the one hand, the coefficients \( k_1 \) and \( k_2 \) can be related to each other (e.g. \( k_1 = \beta, k_2 = 1 - \beta \) with \( \beta \in [0, 1] \)). On the other hand, (if unrelated) these coefficients can be used to compute the cost of an internal representation. If \( k_1 = \gamma_i \) is the unit cost of (response) time and \( k_2 = \gamma_s \) is the unit cost of the storage media to be used, then the total cost is defined by \( \gamma_i \times \text{Time}(T) + \gamma_s \times \text{Space}(T) \).

The parameters considered so far are presented in Table 1.

Combination of all parameters leads to the following general fitness function:

\[ \text{Fitness}(T) = \beta \times \gamma_i \times \text{Time}(T) + (1 - \beta) \times \gamma_s \times \text{Space}(T) \]

### Table 1 Parameters for scalarisation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \beta )</td>
<td>weight coefficient for time optimisation</td>
</tr>
<tr>
<td>( \gamma_i )</td>
<td>unit cost of response time</td>
</tr>
<tr>
<td>( \gamma_s )</td>
<td>unit cost of storage media</td>
</tr>
</tbody>
</table>

#### 3.3 Database design process

In this section, we view the database design process in terms of states and transitions. A state \( S = \{ r_1, \ldots, r_n \} \) in the database design process is a vector consisting of internal representations \( r_1, \ldots, r_n \) for a given conceptual data model. A state \( S \) is also called a pool of internal representations. In state \( S \) we are interested in the maximum Fitness within \( S \) and the minimum Time and Space in \( S \).

A state transition \( S \rightarrow S' \) is caused by applying transformations to the internal representations in \( S \). Then, the database design process is a family of states

\[ \{ S_t = \{ r_{1t}, \ldots, r_{nt} \} | t = 0, 1, 2, \ldots \} \]

At a given point in time \( t \), we are interested in the maximum Fitness and the minimum Time and Space found thus far.

### 4 Experiments during performance engineering

#### 4.1 Experiment set-up

Applying the transformational approach [23] to database design leads to flexible design strategies [24]. As schema transformations are in general very complex, it is desirable to computerise the transformation process. On the one hand, these computer-aided schema transformations can be used for interactive manipulation and comparison of design alternatives, database normalisation algorithms, index selection, and search strategies such as random walk, steepest ascent and probabilistic evolutionary algorithms. The tool EDO introduced above can be used for this purpose.

On the other hand, these schema transformations can be used in experiments. An experiment nondeterministically simulates database design processes. A carefully chosen experiment results in a better insight into a particular aspect of such design processes. In this Section, we discuss some basic experiments and examine their results. All experiments are performed using EDO and the results can be easily reproduced.

When setting up an experiment, a number of decisions have to be made. These decisions can be divided into two classes.
4.1.1 Decisions concerning the problem domain

- Which conceptual data model is going to be used? It may be the model resulting from the information analysis phase or it may be an adapted version. Examples of useful conceptual model transformations are found elsewhere [25]. It is also possible to use several conceptual models in a single experiment. Section 4.2 provides an example.
- What kind of profiles are going to be used, how were they obtained and how are they going to be used for fitness evaluation? Profiles may be estimated during system development or they may be extracted from a running system. Some aspects concerning workload derivation, performance analysis and performance measures are found elsewhere [1, 21]. It is also possible to use different profiles of the same type. As an example, it may be desirable to compare the effect of different access profiles on the database design process.

4.1.2 Decisions concerning scheme transformations

- What strategy (or combination of strategies) is used for scheme transformations? In EDO the following evolutionary algorithms may be used: random walk; steepest ascent, where the next transformation is deterministically chosen as the transformation leading to maximum fitness improvement; probabilistic hill climbing, where the next transformation is non-deterministically chosen, provided it does not lead to a lower fitness; normalisation strategies, where the next transformation is non-deterministically chosen on the basis of (uniqueness and total role) constraints in the conceptual model [2].
- Which setting of control parameters is used? Basic control parameters involve the pool size and the weight coefficient for storage space. Particularly non-deterministic strategies are guided by some more advanced control parameters, such as average join/split rate.

In order to indicate how transformations may be used, we describe several experiments. First we discuss basic experiments concerning ease of transformation, performance improvement and conflicting objectives, and then we describe an experiment concerning the optimal normal form (Section 5).

Software Engineering Journal July 1995

4.2 Ease of transformation

In this Section, we discuss an experiment concerning the ease of transformation. The idea is that for different conceptual data models, the difficulty in finding appropriate transformations for the corresponding internal representations may be quite different. Intuitively, the average number of predicates per atomic object type $N_{ax}$ is expected to play a significant role.

The experiment is as follows. We use conceptual data models with four different values for $N_{ax}$. In each case, transformations are generated by first choosing a predicate, and then trying to find another predicate such that they may be legally put together in an attribute (or column in a table). The process is initiated from an internal representation in which each fact type is represented in a separate relation. We count the number of steps needed to produce a universal relation, i.e. an internal representation in which all fact types are represented in a single table. The result is shown in Fig. 6.

In Fig. 6 we see that, for an information structure with a high number of predicates per atomic object type, relatively few steps are needed for producing a universal relation. This is in accordance with intuition, because in a very compact information structure (see Section 3.1) fact types can be joined relatively easily. Note that in the above experiment the fitness of the internal representations was not used as a guidance parameter.

4.3 Performance improvement

In this Section, we describe a random walk and a hill climbing walk. The process is initialised with a pool of four internal representations for the information structure shown in Fig. 5. In the initial pool, each fact type is represented in a separate relation. After each step, we examine the average and maximum fitness of the internal representations. The result is shown in Fig. 7.

In Fig. 7 we see that the random walk at first gives better results than the hill climbing walk (maximum fitness after six and ten transformations). Later the hill climber finds solutions that are not found by the random walk.

4.4 Utilisation of resources: conflicting objectives

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 ($\beta$) from 0 to 1. As a Result of this varia-
tion, a small subset of the solution space of internal representations can be placed in a graph with axes time/space. We express storage requirements in terms of abstract space units (s.u.) and average response time in terms of abstract time units (t.u.). This is further explained below.

The experiment has the following structure. We make a number of runs using the hill climbing strategy, varying $\beta$ 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. 8.

After 30 evolution steps with different values of $\beta$, five different optima are found. These optima are indicated by steps $= 30$ in Fig. 8. The optimal representation found previously is depicted as a small square. The optima found after 30 more evolution steps are indicated by steps $= 60$. In accordance with intuition, this line is closer to the origin of the graph than the line indicated by steps $= 30$.

5 Deviation from optimal normal form

In this Section, we restrict ourselves to the (flat) relational data model [2, 12]. We consider the relational optimal normal form, along with several other design alternatives obtained by transformations.

5.1 Optimal normal form

In this Section, we use a well known application concerning US presidents and elections (the presidential database [26]) as a running example. The underlying conceptual model of the presidential database is provided elsewhere [19]. This database can be implemented in the following (self-explanatory) relational tables.

Presidents: 
[Pres-Name, Birth-Year, Years-Served, Death-Year, Party, State-Born]

Pres-Hobby: 
[Pres-Name, Hobby]

Pres-Marriage: 
[Pres-Name, Spouse-Name, Marriage-Year]

Election-Candidate: 
[Elect-Years, Candidate, Votes]

Election: 
[Elect-Year, Winner]

Administration-Pres: 
[Admin-Nr, Pres-Name]

Administration-Vice-Pres: 
[Admin-Nr, Vice-Pres-Name]

Administration-Inauguration:
[Admin-Nr, Year-Inaugurated]

State: 
[State-Name, Admin-Entered, Year-Entered]

The above relational tables are in optimal normal form (ONF). For a given conceptual data model, relational tables in ONF may be obtained by examining the constraints in the conceptual model (especially uniqueness constraints) and applying a grouping procedure. In this way, several fact types may be joined in a single table. The resulting ONF tables are at least in 5th normal form, and the number of tables is minimal. The basic ONF algorithm may be found elsewhere [2, 3]. Extensions of the basic algorithm have been discussed previously [25].

5.2 Identity profiles

In case a database application is implemented in relational ONF tables, the database design process may proceed by selecting an appropriate set of indices. However, very often controlled redundancy is introduced in order to fit particular application requirements. This is a highly complex task. First, as retrieval and update patterns usually are quite complex, it is not clear which tables should be split or joined. Secondly, the number of join/split possibilities is often very large. Thirdly, because data profiles also play a role here and storage requirements must be taken into account, there is a typical trade-off. As a consequence, it is necessary to automatically derive several candidate designs of high quality, and further investigate their storage requirements and average response times. An elaborated example of a lower normal form for the presidential database is provided elsewhere [27].

When deriving candidate designs, data profiles, access profiles and device profiles are used to characterise the target environment. In this Section, we use so-called identity profiles. These profiles may be used when only a global comparison between different candidate designs is required or when it is too difficult to define non-identity profiles. Obviously, the generated candidates must then be further examined (see also Section 5.3).

We first consider identity access profiles. In simple cases, an access profile $(R, U)$ consists of a retrieval matrix $R$, where $R_{ij}$ specifies the frequency of an operation from predicator $pi$ into predicator $pj$, and an update
vector $U$, where $U_i$ specifies the frequency of update operations on predicate $p_i$. Sometimes the values in $R$ and $U$ are interpreted as probability or priority, rather than frequency. An identity access profile has the following properties.

- Unconnected predicates have no affinity:
  \[ y_{p, q} \neq \text{Fact}(p) \land \text{Fact}(q) \land \text{Base}(p) \neq \text{Base}(q) \iff R(p, q) = 0 \]
  where $\text{Fact}(p)$ denotes the fact type of predicate $p$ and $\text{Base}(p)$ denotes the object type of predicate $p$.
- All connected predicates have the same affinity:
  \[ y_{p_1, p_2} \neq 0 \land R(q_1, q_2) \neq 0 \Rightarrow R(p_1, p_2) = R(p_1, p_2) \]
- All predicates have the same update pattern:
  \[ y_{p, q} \neq \text{Fact}(p) = U(q) \]

Next we consider identity data profiles. In simple cases, a data profile $D$ is a vector where $D_i$ specifies the number of instances in the population of object type $q_i$. An identity data profile has the following properties.

- All atomic object types have the same number of instances:
  \[ y_{a_1, a_2} \neq D(a_1) = D(a_2) \]
- All composed object types (fact types) have the same number of instances:
  \[ y_{f_1, f_2} \neq D(f_1) = D(f_2) \]

The values in profiles may be absolute or relative. In our case, relative values will suffice because we are interested in comparing different design alternatives (during transformation processes or after completion). We therefore express storage requirements in terms of abstract space units (a.u.) and average response time in terms of abstract time units (a.u.). The exact computation of the properties Time and Space is beyond the scope of this paper. It is stressed here 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 this framework.

We describe an experiment which aims at introducing controlled redundancy in the ONF relational tables given above. As previously mentioned, schema transformations are used for searching the solution space of internal representations. The behaviour of such transformation processes is illustrated in Section 1 (Fig. 1). We try to find internal representations with high fitness. In order to search the complete time-space trade-off we use different values for the weight coefficient $\beta$ for storage space (see Section 3.2). The result is shown in Fig. 9.

In Fig. 9 we see several candidate internal representations for the presidential database, with an indication for their storage requirements and average response times. The number of tables in a candidate is also displayed. Note that this Figure only contains the best candidates found during the transformation process. In order to decide which of the selected candidates is most desirable, some further investigations may be necessary.

### 5.3 Further investigations

The intention of the experiment in the previous Section is as follows. If a database is implemented in relational ONF tables, it may not fit particular application requirements with respect to storage space or response time. If, however, controlled redundancy is to be introduced in order to fit specific requirements, a lower normal form must be produced. As mentioned above, this is a highly complex task. Therefore, several candidates of high quality are automatically derived (by transformations). In order to decide which of the selected candidates is most desirable, some further investigations may be necessary. There are several possibilities in EDO for further examining a candidate. We summarise four possibilities.

- **Local neighbourhood inspection**: the candidate at hand is examined, and compared with its neighbours in the solution space. This provides insight into the strong properties of the candidate and its neighbours. Different strategies can be used for walking through the neighbourhood of a candidate (see Section 4.1).
- **Profile refinement**: the data profile and/or access profile is refined, in order to characterise the behaviour of future users more exactly.
- **Index selection**: an appropriate set of indices is generated for a given candidate. Although not used here, EDO also allows the generation of indices interleaved with the schema transformation process.
- **Prototyping**: for a given candidate, a set of SQL create table statements is generated and imported into the target relational DBMS. In this way, several databases may easily be constructed for the same conceptual model.

### 6 Conclusions

In this paper, two different aspects of information systems development have been combined; schema transformation and performance engineering. We have addressed the problem of selecting an appropriate internal representation for a given conceptual data model. A prototype tool for conversion and transformation of database structures has been introduced. The main difference with related work [2, 3, 5-7] is that transformations may be probabilistically guided, for example on the basis of a multi-objective fitness function. This enables us to systematically examine properties of the solution space at hand, such as the time/space trade-off and the update/retrieval trade-off for a given application.

Our research is motivated by the problem of 'Finding a normal form is easier than getting rid of it'. This is explained as follows. A normal form can be found on the basis of constraints in the conceptual model [2, 3]. If, however, other requirements have to be met, such as storage space and/or response time, these constraints do not really help. The tool EDO offers different kinds of evolutionary algorithms for walking through the solution space of candidate internal representations for a given conceptual data model. In this way, several candidate representations of high quality can be produced. These candi-
dates can be imported directly into the target DBMS, making the construction of several (different) databases for the same conceptual model a relatively trivial task.

In order to further attune EDO to real-life database applications, a project consisting of case studies and extensive experiments within a large Dutch organisation is in its early stages. Future research concerns the interface to commercially available information analysis tools. 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. We are currently working on this topic. This theory is being based on Markov processes, as a random walk through the solution space of internal representations actually is a Markov process.

7 Acknowledgment

The author would like to thank Arthur ter Hofsteede, Theo van der Weide, Eckhard Falkenberg (KGN. University of Nijmegen, The Netherlands), Olga De Troyer (KUB, University of Tilburg, The Netherlands) and Hans-Paul Schaefer (University of Dortmund, Germany) for helpful discussions on this research; and the referees, whose comments resulted in several improvements to the paper.

8 References


The paper was first received on 30 June 1994 and in revised form on 10 April 1995.

The author is with the Department of Information Systems, Faculty of Mathematics and Computing Science, University of Nijmegen, Toernooiweg, NL-6525 ED Nijmegen, The Netherlands.