Automated Conceptual Schema Restructuring

Ulf Sundin

Knowledge Management Project
Dept. of Computer Sciences,
Chalmers University of Technology and University of Göteborg
S-412 96 Göteborg, Sweden

Abstract

The design of a conceptual model is an important and often highly complex task in the process of database design. This complexity implies a need for computer aids for the design process. Many conceptual modeling methods include phases for view integration and schema restructuring. This paper describes an approach of applying methods from Artificial Intelligence, in particular plan generation, to build a computer aid for conceptual schema restructuring in the context of view integration. We also investigate the use of heuristic search strategies and problem simplification techniques for improving the efficiency of the computer aid. Significant improvement in performance is observed when these heuristic methods are applied.

1. Introduction

Conceptual modeling is an important phase in the design of a database. As this is a highly complex task, methods and tools which can support the design process are needed. Several methods for conceptual modeling have been developed during the past decade, including languages and computerized tools to be used by the methods, e.g., [3, 11].

Conceptual modeling methods often assume the following rough phases of conceptual model design, [1, 17, 18]:

— View modeling

— View integration

This paper focus on a computer aid for view integration. Other related research areas include:

— Tools for integrating existing distributed database schemata, e.g., [7].

— Results on restructing system independent schemata into a target DBMS dependent schema, [4, 9].

In [17] a distinction is made between item level integration and object level integration. Results on item level integration are often closely related to database normalization and several automated tools have been developed, e.g., [2, 19]. However, computer aids for object level integration are sparse. In [15] a prototype of a plan generating system for automatic restructuring of conceptual schemata, intended to be used as a computer aid in the process of view integration, is presented. The computer aid produces plans, i.e., sequences of actions, for how to restructure a conceptual schema so that it satisfies certain requirements.

In this paper the the prototype is briefly presented and some problems with the implementation described in [15], are discussed. One conclusion regarding the prototype was that it is necessary, from practical considerations, both to reduce the size of the search space, and to develop appropriate heuristics. This paper also investigates the problem of finding suitable heuristics, in terms of heuristic evaluation functions, [10, 12], and techniques for reducing the search space.

In Section 2 a brief description of the process of schema restructuring in the context of conceptual modeling is given. Also, the plan generating system described in [15] is presented, including a description of the underlying data model and the set of restructuring actions. Some characteristics of the schema restructuring problem are outlined in Section 3, including a discussion of the problem of finding appropriate heuristics. Section 4 contains a discussion of some implemented heuristic methods for improving the efficiency of the system. In Section 5 the results of the paper are summarized and some remaining problems are pointed out.

2. Background

This paper presents an outline of a computer aid for conceptual schema restructuring based on plan generation. In this section the problem of schema restructuring is briefly described in the context of view integration. Furhter, a prototype of an automated tool for schema restructuring is described and discussed.

2.1 View Integration

The process of Conceptual Schema Design is often divided into several distinct phases, [1, 17]. In general these are:

This work is supported by the National Swedish Board for Technical Development (STU).

1. Specification of requirements.
2. Modeling of local views.
3. View Integration.

The objective of view integration is to merge several local view representations into an integrated structure. This integrated structure represents the community view and must satisfy the following:

— it should be internally consistent

— it should accurately reflect each of the local views

— it should support the requirements specified in the requirement analysis phase.

View integration approaches can be classified into three categories, [18]:

— Item level synthesis using frequency information,

— Item level synthesis using items and functional dependencies,

— Object level integration.

The tool presented in this paper is intended for object level integration.

Object level view integration is often described as consisting of the following three main phases, [1]:

1. **Conflict Analysis.** The aim of this phase is to detect and solve incoherencies that exist in the representation of classes of objects in the local views. We distinguish two major tasks:
   — Naming conflict analysis
   — Compatibility analysis
2. **Merging.** After the conflict analysis the part of the universe of discourse common to the two views is homogeneously represented. Thus, the two local views can be merged into a draft integrated schema.
3. **Enrichment and restructuring.** The draft integrated schema is analyzed to obtain a more reliable and clear description of the universe of discourse by means of the integrated schema. This process is often described as consisting of the following:
   a. Analysis of interschema properties, i.e., the modeling features defined between different concepts in different views, not expressed in the local views. These now have to be discovered and added to the draft integrated schema.
   b. Analysis and elimination of redundancy. The result of this task is a structure in which each fact in the schema is represented only once.
   c. Restructuring. This task is devoted to increasing the clarity and expressiveness of the schema and obtaining a semantically richer description of the universe of discourse. As 'clarity' and 'expressiveness' are vague concepts, their interpretation is often handed over to the designer. However, some attempts in defining 'goodness' of a conceptual schema have been made:
      — strong typing of object types, [16].
      — the integrity constraints should, as far as possible, be expressed by means of constructs in the model, [1].
      — normalization of the schema, [19].

There is a need for computer aids which can support all these phases of view integration, as well as conceptual modeling in general. Below, a prototype of a computer aid for schema restructuring is described.

2.2 A Computer Aid for Schema Restructuring

The computer aid for schema restructuring, described in [15], is based on a method for integrating existing databases into a conceptual 'superview', presented in [7]. The integration method is based on a set of schema transformations, which are briefly described below. It is assumed that these transformations, even if originally designed for integration of existing databases, can be used for conceptual schema restructuring.

The method of [7] is defined on a particular type of rather simple data model, described below. However, it should not be too difficult to apply the method to other similar model types, e.g., the models described in [5, 6, 13].

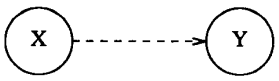
2.2.1 The Data Model

The model is defined on one type of objects, named classes, and two relationships, attribute relationships and generalization relationships. Also, domains and keys are included in the model.

— Each class X has a domain $dom(X)$ of objects.

— Attributes are binary relations between classes. Relations are not named, but uniquely identified by the classes. From this follows that only one attributive relation can hold between any two classes. In the sequel we will use the notation $att(X, Y)$ to denote an attribute relation from class X to class Y. Graphically, this will be represented as:

— Generalizations are binary relations between classes, and have the same semantics as the generalization of [14]. The notation $gen(X, Y)$ will be used to denote that Y is a generalization of X. Graphically, this is represented as:



— The **type** of a given class X is defined as:

$type(X) = \{Y \mid att(X, Y)\}$

Classes with empty types are called primitive classes.

— A key to a class X is a set K of attributes

$K \subset type(X)$

that uniquely identifies X.

2.2.2 Schema Transformations

2.2.2.1 Meet The meet operator produces a common generalization of two classes, if such a generalization can be found. The existence of a generalization is determined by the properties of keys. Assume two non-primitive classes, X and Y, not related by generalization, and that there exists

$K \subset type(X) \cap type(Y)$

such that K is a key for both X and Y. Then, the transformation $'meet(X, Y, Z)'$ can be performed, resulting in the addition of a new class Z and the relations $gen(X, Z)$, $gen(Y, Z)$ and $att(Z, R_i)$, for each $R_i \in type(X) \cap type(Y)$. The attributes $att(X, R_i)$ and $att(Y, R_i)$ are no longer explicit, but inherited from the superclass Z.

2.2.2.2 Join Meet generates a class whose type is the intersection of both types and whose domain is the union of both domains. Another class that may be generated under the same circumstances is the class whose type is the union of both types and whose domain is the intersection of both domains. The transformation $'join(X, Y, Z)'$ produces a new class Z, and adds the relations $gen(X, Z)$ and $gen(Y, Z)$.

2.2.2.3 Fold Fold removes a subtype from a generalization hierarchy. Assume two non-primitive classes, X and Y, such that $gen(X, Y)$. The transformation $'fold(X, Y)'$ is performed by removing the relationship $gen(X, Y)$ and replacing every occurrence of X by Y.

2.2.2.4 Aggregate The attribute hierarchy may be extended by aggregating a subset of the attributes of a given class into a new class, which becomes an attribute of the original class. Assume a non-primitive class X, such that

$type(X) = \{Y_1,...,Y_m,Y_{m+1},...,Y_n\}$.

The transformation $'aggregate(\{Y_1,...,Y_m\}, Y, X)'$ is performed by adding a new class Y and the relationship $att(X, Y)$. Also every relationship $att(X, Y_i)$, where $Y_i \in \{Y_1,...,Y_m\}$, is replaced by $att(Y, Y_i)$.

2.2.2.5 Telescope While aggregate extends the attribute hierarchy, telescope performs the inverse. Assume a non-primitive class Y,

$type(Y) = \{Y_1,...,Y_n\}$

and the relation $att(X, Y)$. The transformation $'telescope(Y, X)'$ is performed by removing the class Y and the relationship $att(X, Y)$, and replacing the relationships $att(Y, Y_i)$, $Y_i \in \{Y_1,...,Y_n\}$, by the relationships $att(X, Y_i)$.

2.2.2.6 Other operations The method also includes operations for adding and deleting classes. These are not considered in this paper. However, a renaming operation has to be included to name the new classes created by meet, join and aggregate. How this is included in the plan generating system is described below.

2.2.3 A Brief Description of the Prototype

The plan generating system composes a plan, consisting of a sequence of the transformations described above, for transforming one conceptual schema, i.e., the initial state, into another conceptual schema satisfying the requirements in the goal state description. The initial state is typically a schema obtained from preliminary integration of two or more local views, [1]. One of the advantages of using a plan generation system for schema restructuring is its flexibility, allowing several different uses:

1. The tool can be used for generating a set of design alternatives which satisfy a target schema. In this case, the goal state description is often partial, i.e., only some aspects of the goal state are considered.
2. It can also be used for verification of a performed restructuring. That is, the problem is to find *a* sequence of actions that produces the restructured target schema from the initial state. In this case the goal state description must be exhaustive, i.e., all objects and relations in the goal state are explicitly stated.
3. Finally, the plan generating system can be embedded into a restructuring language, i.e., commands are defined in terms of the transformations of the plan generating system.

Below, only the first of these possible usages will be discussed. In this case, we are interested in finding a set of solutions to a given problem, i.e., finding a set of sequences of actions, each resulting in a state satisfying the goal state. As an example, assume the initial state to be as in Fig.1. Assume also that 'A' and 'B' have a common key 'Z'. The goal is a schema containing a class X which is a generalization of 'B', but not a generalization of 'A'. This goal can be described as a set

{gen(B,X),not(gen(A,X))}.

First, the initial state is checked. As this state does not satisfy the goal the system generates new possible states by applying the schema transformations. An example of a possible state is the state that results from applying the action 'meet' on the classes 'A' and 'B' to generate a new class. This new class is named by a unique constant symbol,

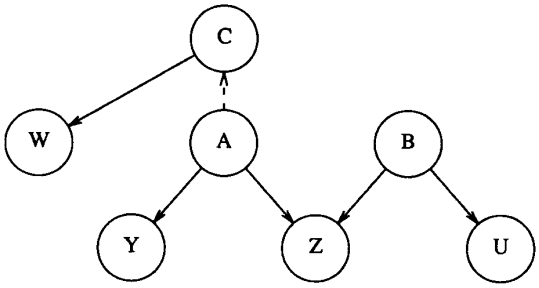


Figure 1. Initial state

which implies that when checking if the state satisfies the goal state, the possibility of renaming these new classes has to be considered.

The appearance of the search tree[1] in this example, after the second level has been partially generated, is displayed in Figure 2. In this case, the states 9), 11), and 15) are examples of states which satisfy the goal, if the name of the new class is renamed to 'X'. However, the state 14) does not satisfy the goal, as this requires that the class 'A' is renamed to 'X'. If the name of the class which should be a generalization of 'B' is irrelevant, a variable can be used in the goal description. In this case the state 14) would also satisfy the goal, by unifying this variable and 'A'. It is now possible to continue to generate states which satisfy the goal. Many of these states will, as in 8) and 12), be equivalent if the naming of new classes disregarded, i.e. the states are isomorphic. This will cause considerable redundancy in the search tree.

3. Some Problems with the Prototype

Analysis of the schema restructuring problem and the prototype reveals, among others, the following characteristics:

— The search space is very large. This is a consequence of both the low level nature of the employed actions, the duplication of states, and the possibility of introducing new objects, see below.

— The set of objects, i.e., classes, is dynamic: new objects can be added (join, meet, aggregation), objects can be removed (fold, telescope), and new objects may be renamed.

— In general, we have only partial knowledge of the goal state.

The very large search space implies a need for employing heuristic search methods, but the two other properties of the problem make this difficult. It is especially difficult to estimate the distance from a given state to the goal state.

The approach for tackling the search problem is twofold:

— Reduction of the search space.

— Use of a heuristic search strategy.

The basis of these two aspects will be discussed below.

3.1 Reduction of the search space

The approach to reducing the search space is based on the observation that a small subset of the initial state is often sufficient to solve a restructuring problem when the goal description is incomplete.

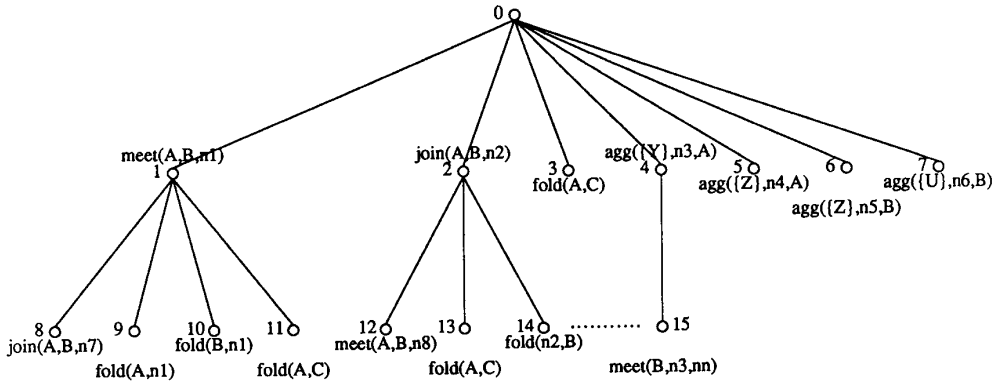


Figure 2. The search tree

1. assuming unrestricted breadth first search

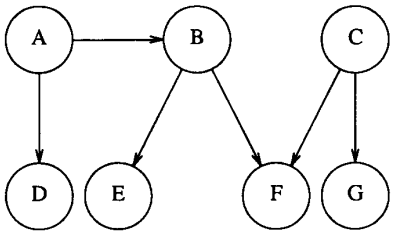


Figure 3. An initial state

For example, assume the initial state in Figure 3, where *key*(*A*)=*D*, *key*(*B*)=*F*, and *key*(*C*)=*G*. The goal state description consists of the attribute *'att*(*B*, *C*)'. It is obvious that to reach the goal state, it is not necessary to consider the class 'A' and attributes of 'A'. This piece of knowledge drastically reduces the search space. The problem is to find a generally applicable criterion for judging what can be eliminated from the initial state, without risking the destruction of possible solutions. One such criterion for problem simplification is discussed in Section 4.1.

We also note that the search space may be drastically reduced by employing more restricted preconditions for some actions, especially 'aggregate'. Consider the initial state in Figure 4,

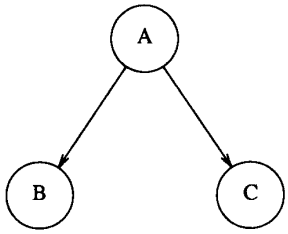


Figure 4

which has an infinite search space. First, the preconditions of *'aggregate*($\{A_1, \ldots, A_n\}$, *B*, *C*)' do not require that the aggregated attributes, i.e., $\{A_1, \ldots, A_n\}$, constitute a *true* subset of type(C). In the above example, this implies that the following is a legal sequence of transformations:

1. *aggregate*({*B*, *C*}, *new*1, *A*)
2. *aggregate*({*B*, *C*}, *new*2, *new*1)
3. *aggregate*({*B*, *C*}, *new*3, *new*2)
4. etc, etc

Secondly, even if we require that the aggregated attributes should be a true subset, it is still possible to generate a infinite sequence of actions, e.g.:

1. *aggregate*({*B*}, *new*1, *A*)
2. *aggregate*({*new*1}, *new*2, *A*)
3. *aggregate*({*new*2}, *new*1, *A*)
4. etc, etc

A possible way of avoiding such meaningless sequences is to require that at least one of the aggregated attributes isn't 'new', i.e., generated by the application of meet, join or aggregate.

3.2 Heuristic Search

A heuristic search strategy requires a function which estimates the distance (or the cost) from one state to the goal state. In the case of schema restructuring, the states are constituted by a set of binary relationships, i.e, attributes and generalizations. It would, then, seem natural to base an estimate of the distance to the goal state on the number of relationships in the goal state which do not hold in a given state, cf 'the 8-puzzle' [10, 12]. However, because of the possibility of creating new objects, and the arbitrary naming of these, and the use of partial descriptions of the goal state, problems may arise.

Concerning the possibility to rename new objects, this causes problems in determining if a state satisfies the goal state, as several alternate name substitutions may be possible.[2] Particular difficulties arise in combination with partial goal descriptions, often resulting in 'unintuitive' solutions, as demonstrated below.

Assume the initial state of Figure 5,

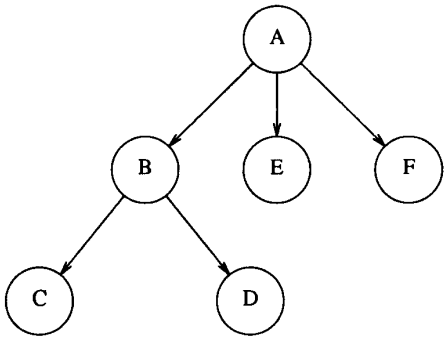


Figure 5

and the partial goal description *'att*(*B*, *E*)'. It is perhaps surprising that the state in Figure 6, obtained by the sequence

— *telescope*(*B*, *A*)

— *aggregate*({*E*}, *new*1, *A*)

satisfies the goal, as it is possible to rename '*new*1' to '*B*'.

The main problem is, however, related to the use of partial goal descriptions. In the worst case, only one relationship of the goal state is explicitly given in the goal description. If

2. Similar difficulties obviously appears when trying to estimate the distance to the goal state.

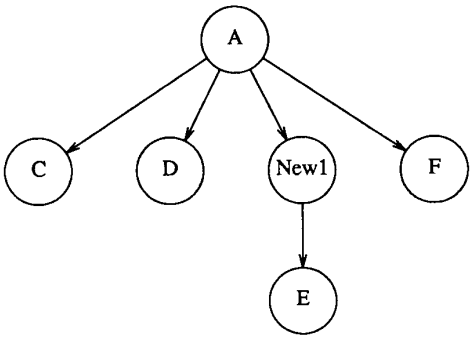


Figure 6

then if the heuristic function is defined as

$h(s)$ = number of relationships in the goal description which are not satisfied in the state s.

this function will map into the set {0,1}, which means that we can only discriminate between the states which satisfy the goal, $h(s)$=0 and states which do not satisfy the goal, $h(s)$=1. In this case, the search strategy degenerates to breadth first search, which implies that something has to be added to the heuristic function.

We note that in the search for a state that satisfies a partially described goal state, it is a good idea to concentrate on the objects that actually occur in the goal state description.

Consider the state in Figure 7,

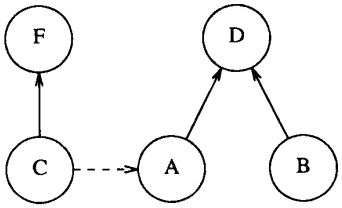


Figure 7

where $key(A) = key(B) = key(C) = D$. If the goal state is described as

$\{gen(B, A), gen(C, A)\}$

the above rule implies that it is preferable to perform the operation

$join(A, B, new)$

rather than, for example,

$aggregate(\{D\}, new, B)$

as the former transformation changes the generalization structure of two of the objects occurring in the goal state description, while the latter only manipulates one of the objects in the goal state description.

The hypothesis is that we can use the information of how a certain state has been reached, in terms of performed actions, to estimate the distance to the goal state. This is further discussed in Section 4.2.

4. Heuristic Methods

In this section the methods that have been used for reducing the search space and making the search strategy more informed are discussed.

4.1 Reducing the Search Space

As mentioned in the previous section, the approach for reducing the search space is based on elimination of irrelevant facts from the initial state. The criterion for selecting the facts to be eliminated can briefly be described as:

An attribute, $att(A, B)$, is deleted from the initial state if both of the following conditions hold:

— neither A nor B exists in the goal state description.

— B is a non-key class, i.e., it is not a part of any key.

Even if this is a quite simple criterion, the search space is drastically reduced. This is demonstrated by an example from [7]. The initial state is given in Figure 8.

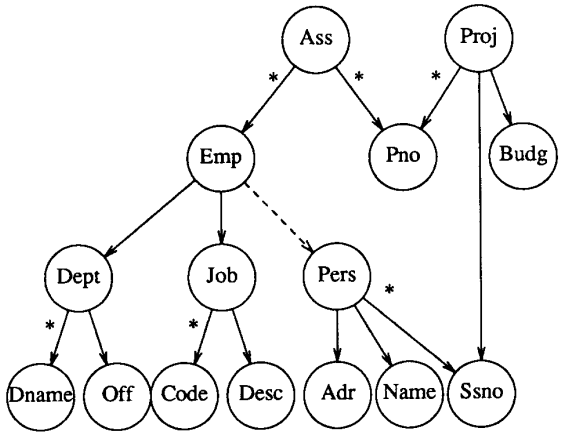


Figure 8. The initial state

Key attributes are marked with asterisks. If the goal is a state which satisfies the following:

$gen(Boss, Emp)$
$att(Proj, Boss)$

this gives the reduced initial state of Figure 9. The effect of the reduction of the initial state is that instead of generating about 11 000 states to obtain a solution, only 147 states have to be generated.

An important question is now if the reduction of the initial state prevents us from finding solutions. Obviously, it will become impossible to reach certain states if the initial state

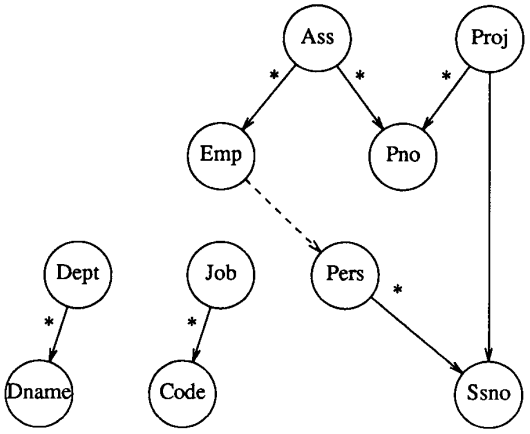


Figure 9. Reduced initial state

is reduced. The problem is then whether the states which no longer can be generated are possible goal states. For example, from the state in Figure 10

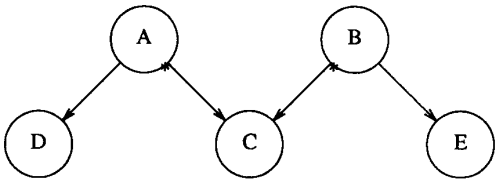


Figure 10

it is possible to reach a state in which *'att(A,B)'* holds. However, if the state is reduced according the above criterion, this is no longer possible.[3] In this case, however, it seems that a better way of expressing the relationship between A and B is by introducing a new class which is a generalization of both A and B (by the meet operation).

4.2 A Heuristic Evaluation Function

As mentioned in Sec.3, when goal descriptions are partial it is not sufficient to rely on a heuristic function based on the difference between the goal state and states. The basic principle is to select the objects which are of interest for a goal state and then find a function such that it can be used as an estimate of the distance between a state and the goal state by employing information of which objects have been manipulated in order to reach the state.

To make the discussion more comprehensible, we utilize the example described in Fig. 8. With the goal description

3. This is also due to the more restrictive preconditions of 'aggregate', see Sec. 3.1.

$att(Proj, Boss)$
$gen(Boss, Emp)$

the interesting objects are 'Emp' and 'Proj', i.e., the objects in the goal description which occur in the initial state. We now have to define, for each operation, which objects are the 'manipulated objects'. First, we note that new objects are not of interest, as the names of these objects can be substituted. Also, objects deleted by an operation are not considered. This gives the following relationships between operations and 'manipulated objects':

— $meet(A, B, New) \rightarrow \{A, B\}$

— $join(A, B, New) \rightarrow$ see 'meet'

— $fold(A, B) \rightarrow \{B\}$.

— $aggregate(\{A_1, \ldots, A_n\}, New, B) \rightarrow \{A_1, \ldots, A_n, B\}$

— $telescope(A, B) \rightarrow \{B\}$

Then the set of 'manipulated objects' related to a state reached by a sequence of operations is the union of the manipulated objects of the all operations of which the sequence consists. For example, the state reached by the sequence

1. $aggregate(\{Pno\}, New1, Ass)$
2. $join(New1, Proj, New2)$

has the set

$\{Proj, Pno, Ass\}$

as 'manipulated objects'.

We are now ready to define a skeleton heuristic function. Let

$$h(s) = w1 \times h1(s) + w2 \times h2(s) + w3 \times h3(s)$$

where

$h1(s) =$ the number of relationships in the goal description which do not hold in state s.

$h2(s) =$ the cardinality of the set difference between the set of 'interesting objects' (see above) and the set of 'manipulated objects' to reach state s. We denote this $card(IO - mo(s))$.

$h3(s) =$ $card(mo(s) - IO)$. (see above)

and $w1$, $w2$ and $w3$ are weights.

How this heuristic function influences the search is of course dependent on the values assigned to the weights. If the assignments

$w1=2$, $w2=1$, $w3=1$

are used, then the search tree displayed in Figure 11 is generated. (h-values within parenthesis) Without making too extensive conclusions from this single example, we note

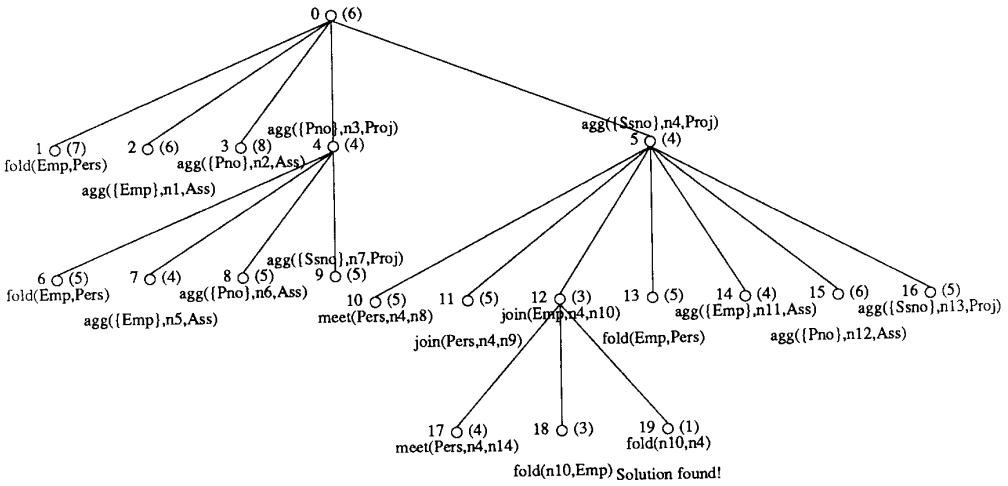


Figure 11. The search tree

that:

— Only 19 nodes are generated, which is a significant improvement compared to breadth first (147).

— Only one of the node expansions is superfluous.

We also note the relationship between $h2$ and $h3$. While $h2(s')$ never can exceed $h2(s)$, where s' is a successor of s, the reverse relationship holds for $h3$. This implies that $h2$ has greater influence on the search in the beginning of the search, i.e., before $h2(s)$=0. Thereafter, the influence of $h3$ is increased.

Another point, related to the above example, is that no g-function, [10, 12], is employed. This is motivated by the fact that we are not primarily interested in finding a minimum cost solution, but rather one of many available solutions, [12]. However, such an approach can be quite dangerous in case of infinite trees, which implies that a g-function should be used.

Finally, we note that the performance of the search strategy may be enhanced by a more detailed description of the goal. For example, if we add the relationship

$$att(Proj, Pno)$$

to the goal description in the above example, the superfluous node expansion never appears. This also requires a redefinition of 'interesting objects': As the relationship added to the goal description holds in the initial state, we are not interested in manipulating 'Pno'. This means that relationships in the goal descriptions which hold in the initial state should not be considered when extracting the set of interesting objects.

5. Concluding Remarks

In this paper an approach has been described for using plan generation as a tool for conceptual schema restructuring by using a set of schema transformations. These schema transformations were defined on a rather simple data model and have been implemented in a prototype. Further, the use of heuristic search strategies and problem simplification techniques to improve the performance of the prototype have been investigated. First, a simple criterion is proposed for reducing the search space, by simplifying the initial state. This is a way of simplifying a difficult search problem. Secondly, a simple and quite efficient heuristic function is proposed. By combining these two heuristic methods previously unsolvable problems, in terms of time and space requirements, have been made manageable.

However, these problems are still 'toy problems'. When it comes to really difficult problems the employed heuristic methods are probably not sufficient. One possible solution, which also was pointed out in [15], is to define higher level operations. This has the additional advantage that the semantics of these operations are probably easier to understand for the user of the system. In a recent paper by Motro [8] some steps in that direction is presented.

Another problem is that both the method for problem simplification and the defined heristic function requires that the description of the goal state consists of attributes and generalizations only. This implies that it is not possible to employ higher level, defined predicates in the goal descriptions, [15].

References

[1] Carlo Batini and M. Lenzerini, "A Methodology for Data Schema Integration in the Entity Relationship Model", *IEEE Trans. on Software Engineering*, Vol. 10 no. 6 (1984).

[2] P.A. Bernstein, "Synthesizing Third Normal Form Relations from Functional Dependencies", *ACM TODS*, Vol. 1 no. 4 (1976).

[3] Michael L. Brodie, John Mylopoulos, and Joachim W. Schmidt (eds.), *On Conceptual Modelling*, Springer-Verlag (1984).

[4] Roland Dahl and J. Bubenko jr., "IDBD - an interactive design tool for CODASYL-DBTG-type data bases" in *Proceedings of 8th VLDB, Mexico City* (1982).

[5] M. Hammer and D. McLeod, "Database Description with SDM: A Semantic Database Model", *ACM TODS*, Vol. 6 no. 3 (1981).

[6] B.C. Housel, V. Waddle, and S.B. Yao, "The Functional Dependency Model for Logical Data Base Design" in *Proceedings 5th Conference on Very Large Data Bases*, Rio de Janeiro (1979).

[7] A. Motro and P. Buneman, "Constructing Superviews" in *Proceedings of the ACM-SIGMOD Conference* (1981).

[8] A. Motro, "Superviews: Virtual Integration of Multiple Databases", *IEEE Transactions on Software Engineering*, Vol. 13 no. 7 (1987).

[9] S. B. Navathe, "Schema Analysis for Database Restructuring", *ACM TODS*, Vol. 5 no. 2 (1980).

[10] Nils J. Nilsson, *Principles of Artificial Intelligence*, Tioga, Palo Alto (1980).

[11] T.W. Olle, H.G. Sol, and A.A. Verjn-Stuart, (eds.), *Information Systems Design Methodologies: A Comparative Review, Proc. of IFIP TC 8 Working Conference on Comparative Review of Information Systems Design Methodologies*, North-Holland (1982).

[12] J. Pearl, *Heuristics - Intelligent search strategies for computer problem solving*, Addison-Wesley (1984).

[13] D.W. Shipman, "The Functional Data Model and the Data Language DAPLEX", *ACM TODS*, Vol. 6 no. 1 (1981).

[14] J.M. Smith and D.C.P. Smith, "Database Abstractions: Aggregation, and Generalization", *ACM TODS*, Vol. 2 no. 2 (1977).

[15] Ulf Sundin, "Using Plan Generation as a Tool for Conceptual Schema Restructring" in *Information Modelling and Database Management*, ed. H. Kangassalo, Springer-Verlag (1987), (to appear).

[16] D. Vermeir and G. M. Nijssen, "A Procedure To Define the Object Type Structure of a Conceptual Schema", *Information Systems*, Vol. 7 no. 4 (1982).

[17] S.B. Yao, S.B Navathe, and J.-L. Weldon, "An Integrated Approach to Database Design" in *Data Base Design Techniques I: Requirements and Logical Stuctures, Proceedings NUY Symp. on Data Base Design, 1978*, Springer-Verlag (1982).

[18] S.B. Yao, V. Waddle, and B. Housel, "View Modeling and Integration Using the Functional Data Model", *IEEE Trans. on Software Engineering*, Vol. 8 no. 6 (1982).

[19] C. Zaniolo and M.A. Melkanoff, "On the Design of Relational Database Schemata", *ACM TODS*, Vol. 6 no. 1 (1981).