Which Are My Preferred Items?

Riccardo Torlone[1] and Paolo Ciaccia[2]

[1] Dip. di Informatica e Automazione
Università Roma Tre
Via della Vasca Navale, 79
00146 Roma, Italy
torlone@dia.uniroma3.it
[2] DEIS — CSITE–CNR
Università di Bologna
Viale Risorgimento, 2
40136 Bologna, Italy
pciaccia@deis.unibo.it

Abstract. Dealing with user preferences is becoming a widespread issue in novel data-intensive application domains, such as electronic catalogs, e-commerce, multimedia databases, and real estates. Given a set of preferences, an important problem is to efficiently determine which are the "best" objects, according to such preferences. In this paper we assume that preferences are expressed in a *qualitative* way over the tuples of a relation schema (e.g., *I prefer product A to product B*), which is quite natural from the user point of view and also includes, as a proper subcase, *quantitative* preferences defined by means of a scoring function. Starting from an analysis of basic properties of (qualitative) preferences, we consider the *Best* operator, which can be used to smoothly embed preferences in queries of relational algebra. We study general properties of this operator and present a practical algorithm for its computation. We show how the algorithm improves the simple nested-loops approach and can lead to faster response times.

1 Introduction

Several modern applications nowadays are faced with the problems of managing user preferences. A remarkable example arises in e-commerce Internet sites (marketplaces, electronic shops, and others), which provide information on thousands, even millions, of products. Effectively supporting user search and browsing over such large repositories entails the problem of properly understanding user needs, filtering out unrelevant items, helping the user to formulate the most appropriate queries, and presenting results ranked according to their presumed relevance. Similar issues arise in multimedia databases, when the user looks for objects similar to a given one. In this case, the problem is that the notion of similarity can be a subjective one, thus the system has to "learn" it and then to exploit the acquired knowledge about user preferences to retrieve the most relevant objects [3].

In the database field, the problem of expressing and managing user preferences has received growing attention in the last few years [1, 4, 6, 8, 9, 11–15]. In many approaches [1, 7–9, 12] preferences are expressed *quantitatively* by defining a *scoring function* that is a weighted linear combination of attributes' values (which have therefore to be numeric). Since the scoring function associates each tuple with a numeric score, tuple t_1 is preferred to t_2 if the score of t_1 is higher than the score of t_2. However, it has been observed that quantitative preferences have a limited expressive power, since they can only define linear orders over tuples, and as such they cannot be used to model more complex patterns of preferences [10]. For this reason *qualitative* preferences have also recently been considered from database researchers [4, 6]. Rather than relying on the existence of a scoring function, with qualitative preferences one just assumes that tuples can be compared using some computable function that, in the typical case, defines a partial order over the tuples, but in the general case it is just a binary relation between the objects of a database.

In this paper we consider qualitative preferences and, for the sake of generality, disregard issues related to *how* (that is, in which language) preferences are expressed. This means that we do not rely on any linear order on "dimensions" or similar properties. With this in mind, we first argument about some basic properties of qualitative preferences and clarify some issues which are relevant for query evaluation. To this purpose, we introduce the *Best* operator, discuss its properties, and then turn to consider how Best queries can be evaluated. In particular, unlike previous works, we consider the case where not only the "best" (i.e., highest ranked) tuples are to be computed [4, 6], but possibly also tuples with a lower rank are to be returned. This leads us to develop a basic algorithm whose major aim is to limit as much as possible the number of comparisons between tuples, which in the worst case are $O(n^2)$ for a relation with n tuples. The algorithm can also be adapted to work with *cyclic* preferences.

The paper is organized as follows. In Section 2 we introduce the basic notions and investigate some general issues about preferences. In Section 3 we present the Best operator and study its general properties. Then, in Section 4, we propose an algorithm for computing the Best operator and discuss possible improvements of this algorithm. Finally, in Section 5, we list a set of practical and theoretical open issues.

Because of space limitation, the proofs of the various results are omitted and can be found in the full paper [5].

2 Qualitative Preferences

In this section we introduce the basic notion of user (qualitative) preferences. In order to simplify the notation, we will refer to the relational model [2] but the various notions and results can be extended to more involved data models, like an XML-based or an object oriented one.

We denote by $R(X)$ a *relation scheme*, where R is the *name* of the relation and $X = A_1, \ldots, A_k$ is a set of *attributes*. We also assume that each attribute

$A \in X$ has associated a set of values D called the *domain* of A. A *tuple* t over a scheme $R(X)$ associates with each $A \in X$ a value taken from its domain. A *relation* r over a scheme $R(X)$ (also called an *instance* of $R(X)$) is a finite set of tuples over $R(X)$.

Preferences over a relational scheme $R(X)$ can be naturally expressed by a collection of pairs of tuples over $R(X)$: each pair specifies the preference of one tuple over another one.

Definition 1 (Preference relation). *A* (qualitative) preference relation $\succ$ *over a relational scheme* $R(X)$ *is a binary relation over* $\prod_{i=1}^{k} D_i$. *We denote a single preference* $(t_1, t_2) \in \succ$ *by* $t_1 \succ t_2$ *and say that* t_1 is preferable *to* t_2.

Example 1. Let us consider the following relation.

	Make	Model	Color	Price
t_1	BMW	330	Black	30K
t_2	Ford	Escort	White	20K
t_3	Toyota	Corolla	Silver	15K
t_4	Ferrari	360	Red	100K

A set of qualitative preferences over it, possibly expressed by a potential customer, can be the following: $t_1 \succ t_2$, $t_3 \succ t_2$, $t_4 \succ t_1$ and $t_3 \succ t_1$.

A preference relation can be naturally represented by means of a directed graph $G_\succ$ that we call *preference graph*. In this graph, nodes correspond to tuples and there is an edge from a node t_1 to a node t_2 if $t_1 \succ t_2$. For instance, the preference graph for Example 1 is reported in Figure 1. It is remarkable that several properties of preference relations can be expressed and studied in terms of properties of the corresponding graph.

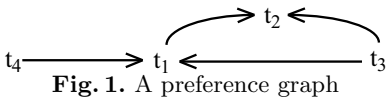

Fig. 1. A preference graph

Note that Definition 1 does not put any a priori hypothesis on the binary relation $\succ$. Usually, in many applications, $\succ$ is at least a (strict) partial order, i.e. $\succ$ is irreflexive (we never have $t \not\succ t$), asymmetric (we never have both $t_1 \succ t_2$ and $t_2 \succ t_1$), and transitive (if $t_1 \succ t_2$ and $t_2 \succ t_3$ then also $t_1 \succ t_3$). Intuitively, while reflexivity does not make sense in this context, asymmetry prevents "inconsistencies" among preferences and transitivity yields a simple way to infer preferences.

In some practical cases, however, both asymmetry and transitivity may fail to hold. For instance, when preferences are obtained by taking into account single properties of tuples. Consider as a simple example the following symmetric preferences: *I like Ferrari more than Toyota, but Toyota is preferable to Ferrari for its price*. Similarly, as an example of nontransitive preferences, consider the

following: *I prefer the color of car t_1 to the color of car t_2, and the model of car t_2 to the model of car t_3, but I have no reason to prefer car t_1 to car t_3.* Since however symmetric and, more in general, *cyclic* preferences ($t_1 \succ t_2 \succ \ldots t_n \succ t_1$ for some finite n) lead to ambiguous situations, we introduce a "stronger" notion of preference under which a tuple t_1 is preferable to a tuple t_2 only if the converse does not hold.

Definition 2 (Strong preference relation). *Given a preference relation $\succ$ over $R(X)$, we say that t_1 is* strongly preferable *to t_2, denoted by $t_1 \succ\!\!\succ t_2$, if and only if $t_1 \succ t_2$ and $t_2 \not\succ t_1$.*

Example 2. Consider the relation of Example 1 and assume that $t_1 \succ t_2$, $t_3 \succ t_2$, $t_2 \succ t_4$ and $t_2 \succ t_1$. Then we have $t_3 \succ\!\!\succ t_2$ and $t_2 \succ\!\!\succ t_4$, but $t_1 \not\succ\!\!\succ t_2$ and $t_2 \not\succ\!\!\succ t_1$.

A strong preference relation is a preference relation itself for which we can show that if $\succ$ is transitive then $\succ\!\!\succ$ is a partial order.

Two other natural notions among tuples can be derived from the above definitions. The first one is called *indifference* since it holds between tuples for which an explicit preference does not exist. More precisely, we say that t_1 *is indifferent to* t_2 (written $t_1 \| t_2$) iff neither $t_1 \succ t_2$ nor $t_2 \succ t_1$ hold.

The notion of *likeness* concerns those tuples for which the given preferences do not allow us to establish that one is better than the other. In particular, we say that t_1 *is like* t_2 (written $t_1 \approx t_2$) if both $t_1 \succ t_2$ and $t_2 \succ t_1$ hold.

Example 3. Consider again the relation of Example 1 and assume that now the set of preferences is: $t_1 \succ t_2$, $t_4 \succ t_1$, $t_3 \succ t_2$, $t_4 \succ t_2$ and $t_2 \succ t_1$. Then we have $t_1 \| t_3$, $t_3 \| t_4$ and $t_1 \approx t_2$.

It is interesting to note that even if $\succ$ is a partial order, the indifference relation is not necessarily transitive. Consider for instance the case in which we simply have: $t_1 \succ t_2$ and $t_3 \succ t_4$ which is trivially a partial order. Then $t_1 \| t_3$, $t_3 \| t_2$ but t_1 and t_2 are not indifferent. In this case, it is probably inappropriate to call it indifference since it would actually represent incomparability among tuples. Conversely, it is easy to show if $\succ$ is transitive then $\approx$ is an equivalence relation, that is, it is a reflexive, symmetric and transitive relation. For its practical relevance it is also useful to consider the transitive closure, $\succ^*$, of a preference relation $\succ$.

3 Querying with Preferences

In this section we present and investigate an operator that, combined with the standard operators of relational algebra [2], can be used to specify queries over a database with preferences. This operator is called *Best* (denoted by $\beta^m_\succ$) its behavior is quite natural: $\beta^1_\succ(r)$ returns all the tuples t of a relation r for which there is no tuple in r better than t according to $\succ$. If the user is not satisfied from the basic result, this operation can be applied iteratively: at each step it returns the best tuples of r, excluding the tuples retrieved in previous steps.

Definition 3 (Best operator). *Let r be a relation over a scheme $R(X)$ and let $\succ$ be a qualitative preference relation over $R(X)$. The* Best operator $\beta^m_\succ$ of rank $m > 0$ *is defined as follows:*

- $\beta^1_\succ(r) = \{t \in r \mid \nexists t' \in r, t' \succ t\}$
- $\beta^{m+1}_\succ = \beta^1_\succ(r - \bigcup_{i=1}^{m} \beta^i_\succ(r))$

Example 4. Consider the preferences of Example 1: $t_1 \succ t_2$, $t_3 \succ t_2$, $t_4 \succ t_1$ and $t_3 \succ t_1$. Then we have $\beta^1_\succ(r) = \{t_4, t_3\}$, $\beta^2_\succ(r) = \{t_1\}$ and $\beta^3_\succ(r) = \{t_2\}$.

Used together with the other operators of relational algebra, the Best operator can be profitably used in queries where preferences are taken into account.

Example 5. Let us consider again Example 1. The Make and the Model of the best cars, according to the given preferences, whose price is less than $50K$ can be obtained by the following expression: $\pi_{Make,Model}(\beta^1_\succ(\sigma_{Price<50K}(r)))$.

We now consider some interesting properties of the Best operator. The first one establishes that the Best operator is somehow insensitive on the transitivity of $\succ$.

Theorem 1. *Let $\succ$ be a preference relation over a scheme $R(X)$, then for each instance r of $R(X)$ and each $i > 0$, $\beta^i_\succ(r) = \beta^i_{\succ^*}(r)$.*

Other interesting properties of the Best operator can be defined and checked by considering the directed graph $G_\succ$ associated with the preference relation $\succ$. We recall that a source of a graph is a node with no incoming edges, a path is a sequence of nodes connected by edges, the length of a path composed by n nodes is $n - 1$. We have the following results.

Lemma 1. *Let $\succ$ be a preference relation and $G_\succ$ be its preference graph.*

1. *$\beta^1_\succ(r)$ returns the sources of $G_\succ$.*
2. *If $G_\succ$ is acyclic then, for every $t \in r$, $t \in \beta^{k+1}_\succ(r)$ where k is the length of the longest path from any source of $G_\succ$ to t.*
3. *The fixpoint of $G_\succ$ is equal to the length of the longest path from any source of $G_\succ$ to a node of $G_\succ$ involved in a cycle.*
4. *If $G_\succ$ is acyclic then the fixpoint of $\beta^m_\succ(r)$ is equal to the length of the longest path in $G_\succ$.*

4 Evaluating the Best Operator

In this section we propose an algorithm for computing the Best operator, demonstrate its correctness, and discuss possible improvements of this algorithm. Initially, we assume that the preference relation is a partial order. By Theorem 1, the assumption of transitivity does not imply a loss of generality, whereas the hypothesis of asymmetry (as well as irreflexivity) will be later relaxed.

Let us start with a basic result that fixes a lower bound for the computation of the Best operator. The basic operation in this computation consists in verifying whether a tuple is preferable to another: we assume that this task requires constant time. Thus, the time computational complexity will be measured in terms of the number of such comparisons.

Theorem 2. *The computation of the Best operator requires $O(n^2)$ time in the worst case, where n is the cardinality of r.*

We point out that the above result is valid not only for the first application of the Best operator but for the whole computation, which may require several iterations, until a fixpoint is reached.

Now, although in some cases $O(n^2)$ comparisons are necessarily needed to completely determine the result of the Best operator, it is also true that the actual computational effort might strongly depend on (the structure of) the preference relation and on how an algorithm is able to exploit it. Although a nested-loops algorithm can be easily applied here [4], it is evident that it will perform poorly in the general case, since it will execute $O(n^2)$ comparisons regardless of the underlying preference relation.

We now present an algorithm that, unlike nested-loops, can compare tuples multiple times, but it can achieve much faster response times because of the way it organizes its search space.

Let us consider a relation r and a preference relation $\succ$ that is a partial order. The algorithm for the evaluation of $\beta^i_\succ(r)$ is composed by a number of *phases*, one for each iteration of the Best operator. In turn, each phase consists of a set of one or more *scans* over a set C_i of *candidate tuples* which might belong to the output Out_i of the $i-$th phase. At the end of each scan only one tuple is selected and inserted into Out_i.

First phase In the first phase we set $C_1 = r$. Each tuple of C_1 is extracted and processed, in any order but one at time. Let t be first tuple extracted from C_1 in the first scan, which temporarily plays the role of *selected* tuple. We compare t with another tuple t' extracted from C_1. Three cases are possible:

1. $t \parallel t'$: in this case we just put t' in a set U_1 of *unresolved* tuples and t remains the selected tuple;
2. $t \succ t'$: in this case we put t' in a set $D^t_\succ$, which contains the tuples dominated by t according to $\succ$, and t remains the selected tuple; if t' belongs to another set $D^{t''}_\succ$ it is deleted from it;
3. $t' \succ t$: in this case t' becomes the selected tuple and t is added to the set $D^{t'}_\succ$, which contains the tuples dominated by t' according to $\succ$; if t belongs to another set $D^{t''}_\succ$, it is deleted from it.

The algorithm proceeds by processing in this way all the tuples in C_1.

Let t be the selected tuple we get at the end (that is, when $C_1 = \emptyset$). We can show that there is no tuple among all the processed ones that dominates t and so $t \in \beta^1_\succ(r)$. However, there might be some tuple t' in U_1 dominated by t that

has not been compared with t. For this reason, we also compare t with all the tuples in U_1 for which this comparison has no been done yet: if a tuple t'' in U_1 is actually dominated by t, we put t'' in the set $D^t_\succ$ and delete it from U_1. At the end of this job, we finally put t in the set Out_1, in which we collect the tuples to be returned as output of the first phase. This completes the first scan.

At this point, if U_1 is not empty, we copy in C_1 all the tuples in U_1, we empty U_1, we select a tuple in C_1 and repeat the whole procedure, i.e. execute another scan at the end of which another tuple will be selected and inserted in Out_1. When, at the end of a scan, we obtain $U_1 = \emptyset$ the first phase is concluded.

Following phases In the second phase, we put in C_2 only the tuples in the sets $D^t_\succ$, for each $t \in Out_1$. Note that this strongly reduces the search space. We then select a tuple from C_2 and proceed as in the first phase. As before, the second phase terminates when the set U_2 is empty.

The subsequent phases proceed similarly. Each phase starts by inserting in C_i only the tuples in the sets $D^t_\succ$, for each $t \in Out_{i-1}$. We can show the following result.

Theorem 3. *The basic algorithm correctly computes the result of $\beta_\succ$ over a relation r of cardinality n and requires $O(n^2)$ time in the worst case.*

Example 6. Let $r = \{t_1, t_2, t_3, t_4, t_5\}$ and assume we have the following preferences: $t_1 \succ t_2$, $t_4 \succ t_1$, $t_4 \succ t_2$, $t_4 \succ t_3$, $t_5 \succ t_2$ and $t_5 \succ t_3$. Initially, we set $C_1 = r$. Let t_1 be the first tuple extracted from C_1 and so the initial selected tuple. If we now extract t_2, t_1 remains the selected tuple as $t_1 \succ t_2$ and we set $D^{t_1}_\succ = \{t_2\}$. Assume we now extract t_3: since t_1 and t_3 are indifferent, t_1 remains again the selected tuple and we set $U_1 = \{t_3\}$. If we then extract t_4, this tuple becomes the selected tuple as $t_4 \succ t_1$ and we set $D^{t_4}_\succ = \{t_1\}$. Finally, we extract t_5 from C_1 that is indifferent to t_4 and so we set $U_1 = \{t_3, t_5\}$. We now need to compare t_4 with t_3, which is in U_1 and has not been compared with it. Since $t_4 \succ t_3$, we add t_3 to $D^{t_4}_\succ$, we delete it from U_1, and set $Out_1 = \{t_4\}$. We now repeat the procedure by setting $C_1 = U_1 = \{t_5\}$. Since now C_1 is a singleton, we just set $Out_1 = \{t_4, t_5\}$ and since $U_1 = \emptyset$, this conclude the first phase. In the second phase we put in C_2 only the tuples in $D^{t_4}_\succ$, that is: t_1 and t_3, and proceed similarly. Since t_1 and t_3 are indifferent, we get $Out_2 = \{t_1, t_3\}$ and also the second phase is concluded. We then have $C_3 = \{t_2\}$ and so we immediately obtain $Out_3 = \{t_2\}$. The algorithm terminates here as $D^{t_2}_\succ = \emptyset$.

Several variants of the basic algorithm can be defined for the purpose of taking into account more general cases and of improving its efficiency.

Let us first consider the case in which $\succ$ is neither asymmetric nor irreflexive. The most effective way to deal with this case is to resort to the $\succ\!\!\succ$ relation, under which two tuples t_1 and t_2 such that $t_1 \approx t_2$ are returned in the same iteration of the Best operator.

The algorithm can be modified accordingly: at each iteration of the i-th phase, we keep a *set* of selected tuples S_i instead of just one tuple. When a tuple t' is extracted from the set C_i, we compare t' with one tuple t in S_i. Then, besides

the three cases considered in the basic algorithm, we can also have $t \approx t'$. In this case, we just add t' to S_i. In the other cases, we proceed as in the basic version of the algorithm, with the only difference that if $t' \succ t$, we set $D^{t'}_{\succ} = D^{t'}_{\succ} \cup S_i$ and $S_i = \{t'\}$. Note that, as before, only one comparison is needed at each iteration with a representative in S_i. This is because $\approx$ is an equivalence relation.

With respect to the efficiency of the algorithm, we note that the crucial operation of the algorithm consists in the extraction of a tuple from the set C_i of candidates. Then, performance can be improved with an appropriate choice of such a tuple based on the preference relation itself. Differently from what it might appear at first sight, it turns out that is convenient to extract tuples form C_i in reversed order of preference. In fact, in this way, the selected tuple changes frequently and, as a consequence, dominated tuples are distributed among several sets $D^t_{\succ}$. It follows that the cardinality of such sets is minimized. Since the set of candidate tuples is obtained at each phase as union of the sets $D^t_{\succ}$, for each t in the output of the previous phase, this in turn implies a reduction of the search space.

Another improvement can be obtained by employing special properties of relation $\|$. For instance, when $\|$ is an equivalence relation. In this case, if at a certain iteration of the i-th phase t is the selected tuple and we find $t' \succ t$, t' becomes the selected tuple and we can set $D^{t'}_{\succ} = D^{t'}_{\succ} \cup \{t\} \cup U_i$ and empty U_i. We can show that with this change the correctness of the algorithm is preserved and the number of comparisons is generally reduced.

5 Final Discussion

In this paper we have considered the problem of querying database relations when qualitative preferences are defined over them. Starting from an analysis of basic properties of (qualitative) preferences, we have introduced and studied the *Best* operator, which can be used to embody preferences in queries. Although the computation of this operator has an inherent quadratic complexity, much better results can be obtained depending on the structure of preferences. To this end, since the only available algorithm for qualitative preferences was based on a nested-loops evaluation style, we have introduced an algorithm that tries to limit the number of comparisons among tuples by reducing the "search space" at each iteration.

Starting from this algorithm, a number of open problems remain to be solved. We list only a few of them.

- Devising a disk-based algorithm for computing the Best operator on large relations appears to be a challenging problem. For the efficient management of the $D^t_{\succ}$ sets of dominated tuples a paged index could be considered.
- From a theoretical point of view it would be useful to identify specific patterns of preferences and characterize the algorithm complexity accordingly.
- Devising a specific strategy for performing an effective scan over the set of candidate tuples is also a relevant issue. This seems to be tightly related to

the theoretical problem of the minimum number of comparisons needed to infer a partial order.

References

1. R. Agrawal and E. L. Wimmers. A Framework for Expressing and Combining Preferences. In *Proc. of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, USA*, pp. 297–306, 2000.
2. P. Atzeni and V. De Antonellis. Relational Database Theory. *Benjamin/Cummings*, 1995.
3. I. Bartolini, P. Ciaccia, and F. Waas. FeedbackBypass: A New Approach to Interactive Similarity Query Processing. In *Proc. of 27th Intl. Conf. on Very Large Data Bases, Roma, Italy*, pp. 201–210, 2001.
4. S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *Proc. 17th Intl. Conf. on Data Engineering, Heidelberg, Germany*, pp. 421–430, 2001.
5. P. Ciaccia and R. Torlone. Finding the Best when it's a Matter of Preference. *Technical report available at: http://www.dia.uniroma3.it/~torlone/pubs/pub.htm*, 2002.
6. J. Chomicki. Querying with Intrinsic Preferences. In *Proc. 8th International Conf. on Extending Database Technology (EDBT'02), Prague, Czech Republic.*, 2002.
7. P. Ciaccia, D. Montesi, W. Penzo, and A. Trombetta. Imprecision and User Preferences in Multimedia Queries: A Generic Algebraic Approach. In *Proc. 1st Int. Symp. on Foundations of Information and Knowledge Systems (FoIKS 2000), Burg (Spreewald), Germany*, pp. 50–71, 2000.
8. R. Fagin. Combining Fuzzy Information from Multiple Systems. In *Proc. 15th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS'96), Montreal, Canada*, pp. 216–226, 1996.
9. R. Fagin and E.L. Wimmers. Incorporating User Preferences in Multimedia Queries. In Proc. 6th Intl. Conf. on Database Theory (ICDT'97), Delphi, Greece, pp. 247–261, 1997.
10. P. C. Fishburn. Preference Structures and Their Numerical Representations. *Theoretical Computer Science*, 217(2): 359–383, 1999.
11. K. Govindarajan, B. Jayaraman, and S. Mantha. Preference Queries in Deductive Databases. *New Generation Computing*, 19(1): 57–86, 2001.
12. V. Hristidis, N. Koudas, and Y. Papakonstantinou. PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries. In *Proc. of the 2001 ACM SIGMOD International Conference on Management of Data, Santa Barbara, USA*, pages 259–269, 2001.
13. M. Lacroix, P. Lavency. Preferences: Putting More Knowledge into Queries. In *Proc. of 13th Intl. Conf. on Very Large Data Bases, Roma, Italy*, pp. 217–225, 1987.
14. W. Ng. An Extension of the Relational Data Model to Incorporate Ordered Domains. *ACM Transactions on Database Systems*, 26(3), 2001.
15. K. Tan, P. Eng, B. C. Ooi. Efficient Progressive Skyline Computation. In *Proc. 17th Intl. Conf. on Data Engineering, Heidelberg, Germany*, pages 301–310, 2001.