2014 47th Hawaii International Conference on System Science

Soft Retrieval and Uncertain Databases

Ronald R. Yager
Iona College
yager@panix.com

Rachel L. Yager
Metropolitan College of New York
ryager@mcny.edu

Abstract

We investigate tools that can enrich the process of querying databases by enabling the inclusion of more human focused considerations. We show how to include soft conditions with the use of fuzzy sets. We describe some more sophisticated techniques for aggregating the satisfactions of the individual conditions based on the inclusion of importances and the use of the OWA operator. We discuss a method for aggregating the individual satisfactions that can model a lexicographic relation between the individual requirements. We look at querying databases in which the information in the database can have some probabilistic uncertainty.

1. INTRODUCTION

Database structures play a pervasive role in underlying many of our most prominent websites, as a result the intelligent retrieval of information from databases is an important task. The use of soft concepts often allows us to model human cognitive concepts. Here we focus on providing a framework for retrieving information from database structures via soft querying [1-4]. A query can be seen as a collection of requirements and an imperative for combining an object's satisfaction to the individual requirements to get the object's overall satisfaction to the query. We describe some fuzzy set based methods that enable the inclusion of soft human focused concepts in the construction of a query [5-8]. Here we also describe sophisticated techniques for aggregating the satisfactions of the individual conditions to obtain an object's overall satisfaction to a query. The inclusion of importances and the use of the OWA operator are some tools that provide this facility. Here we also discuss a method for aggregating the individual satisfaction's which can model a lexicographic relation between the individual requirements. In addition we look at databases in which the information can have some probabilistic uncertainty

2. STANDARD DATABASE QUERYING

A database consists of a collection of attributes, V_j, $j = 1$ to r and a domain X_j for each attribute. Associated with a database is a collection of objects D_i, the objects in the database. Each D_i is an r tuple, $(d_{i1}, d_{i2}, ..., d_{ir})$. Here $d_{ij} \in X_j$ is the value of attribute V_j for object D_i.

A query Q is of a collection of q pairs. Each pair $P_k = (V_{Q(k)}, F_{Q(k)})$ consists of an attribute $V_{Q(k)}$ and an associated property for that attribute $F_{Q(k)}$. Here Q(k) indicates the index of the attribute in the k^{th} pair. Thus if Q(k) = i then the attribute in the kth pair is V_i. In addition to the collection of pairs a query contains information regarding the relationship between the pairs. We denote this information as Agg.

The querying process consists of testing each object D_i to see if it satisfies the query. The process of testing an object consists of two steps. The first is the determination of the satisfaction of each of the pairs by the object. We denote this as $T_r(P_k|D_i)$ or more simple as t_{ik}. The second step is to calculate the overall satisfaction of D_i to the query Q, $Sat(Q/D_i)$, by combining the individual t_{ik}, guided by the instructions contained in Agg. Here then $Sat(Q/D_i) = Agg(t_{i1}, ..., t_{iq}) = T_i$

The calculation of t_{ik}, $T_r[P_k|D_i]$ is based on determining whether the condition $F_{Q(k)}$ is satisfied based on $d_{iQ(k)}$, the value of $V_{Q(k)}$ for the i^{th} object D_i. In the standard environment $F_{Q(k)}$ is represented as a crisp subset of $X_{Q(k)}$ indicating the desired values of $V_{Q(k)}$. The determination of satisfaction, $T_r[P_k|D_i]$, is based on whether $d_{iQ(k)} \in F_{Q(k)}$. The system returns the value $t_{ik} = 1$ if $d_{iQ(k)} \in F_{Q(k)}$ and $t_{ik} = 0$ if $d_{iQ(k)} \notin F_{Q(k)}$. Using the notation $F_{Q(k)}(d_{iQ(k)})$ to indicate membership of $d_{iQ(k)}$ in $F_{Q(k)}$ we get $t_{ik} = F_{Q(k)}(d_{iQ(k)})$. Thus in this case each of the values $t_{ik} \in \{0, 1\}$ where 0 indicates false, not satisfied, and 1 indicates true, satisfied.

In this binary environment the aggregation process is constructed as a well formed logical statement consisting of conjunctions, disjunctions , negations and implications. The overall satisfaction $Sat(Q|D_i)$ is equal to the truth-value of this well-formed formula. The calculation of this truth-value involves the use of Min, Max and Negation. For example the requirement that all pairs in Q are satisfied is expressed as $T_i = Min_{k=1 \text{ to } r}[t_{ik}]$. The requirement that any of the conditions need be satisfied is expressed as $T_i = Max_{k=1 \text{ to } r}[t_{ik}]$. We note this is a binary environment, $T_i \in \{0, 1\}$, and hence we have two categories of objects, those which satisfy the query and those that do not.

978-1-4799-2504-9/14 $31.00 © 2014 IEEE
DOI 10.1109/HICSS.2014.123

IEEE computer society

3. Soft database querying

An extension of the querying process involves the use of soft or flexible queries [6, 8]. This idea extends the process of querying standard databases in a number of ways. The most fundamental is to allow the search criteria, the $F_{Q(k)}$, to be expressed as fuzzy subsets of the domain $X_{Q(k)}$. This allows one to represent imprecise non-crisply bounded concepts. In this case the determination of the satisfaction of a criteria $P_k = (V_{Q(k)}, F_{Q(k)})$ by an object D_i, t_{ik}, again equals the membership grade of $d_{iQ(k)}$ in $F_{Q(k)}$, $t_{ik} = F_{Q(k)}(d_{iQ(k)})$. However in this case $t_{ik} \in [0, 1]$, it takes a value in the unit interval rather than simply in the binary set {0, 1}. An important implication of this is that the overall satisfaction of the element D_i will also lie in the unit interval. This will allow for a richer and more discriminating ordering of the elements in regard to their satisfaction to the query then simply satisfied or not, here we get a degree of satisfaction. We note the evaluation of the overall satisfaction to a query Q whose Agg operator is based on a well formed logical formula can be implemented in this framework. In particular conjuncting the satisfactions to two criteria pairs P_{k_1} and P_{k_2} is implemented by $\text{Min}(t_{ik_1}, t_{ik_2})$. Disjuncting the satisfactions to two criteria pairs P_{k_1} and P_{k_2} is implemented by $\text{Max}(t_{ik_1}, t_{ik_2})$. The negation of the satisfaction to P_{k_i} is implemented as $1 - t_{i_{k_1}}$.

A second benefit obtained by using flexible queries is the allowance for more sophisticated formulations for the operator Agg used to determine the overall satisfaction to the query [6]. Let us look at some of these extensions.

A first class of extensions is a generalization of the operations used for implementing the "anding" and "oring" operator. Assume P_1 and P_2 are two conditions whose satisfaction for D_i are t_{i1} and t_{i2}. Classically the "anding" of the satisfactions to these two criteria has been implemented using the Min operator: P_1 and $P_2 \Rightarrow \text{Min}(t_{i1}, t_{i2})$. The "oring" of the satisfactions to these two criteria has been implemented using the Max operator: P_1 or $P_2 \Rightarrow \text{Max}(t_{i1}, t_{i2})$. The t-norm operator is a generalization of the "anding" operator [9], it provides a class of cointensive operators that can be used to implement the conjunction. We recall a t-norm is a binary operator T: $[0, 1] \times [0, 1] \rightarrow [0, 1]$ having the properties: symmetry, associativity, monotonicity and has one as the identity. Notable examples of t-norm in addition to the Min are $T_P(a, b) = a\ b$ and $T_L(a, b) = \text{Max}(0, a + b - 1)$. Since for any t-norm T we have $T(a, b) \leq \text{Min}(a, b)$ then using other t-norms increases the penalty for not completely satisfying the conditions.

The t-conorm operator provides a generalization of the *oring* operator. A t-conorm is a mapping S: $[0, 1] \times [0, 1] \rightarrow [0, 1]$ which has the same first three properties of the t-norm binary, condition four is replaced by S(0, a) = a, zero as Identity. In addition to the Max other notable examples of t-co-norm are $S_P(a, b) = a + b - ab$ and $T_L(a, b) = \text{Min}(a + b, 1)$. We note for any t–norm S we have $S(a, b) \geq \text{Max}(a, b)$.

Another extension to the standard situation is the inclusion of importances associated with the conditions [10, 11]. Here we assume each of the q conditions P_k has an associated importance weight $w_k \in [0, 1]$. The methodology for including importances depends on the aggregation relationship between the criteria, it is different for "anding" and "oring".

Assume for object D_i we have that $t_{ik} \in [0, 1]$ as its degree of satisfaction to P_k. Consider first the case in which we desire an "anding" of the P_k. That is our aggregation imperative is expressed as

$$(P_1, w_1) \text{ and } (P_2, w_2) \text{ ..... and } (P_q, w_q).$$

In this case the weighted aggregation results in an overall the satisfaction $\text{Min}(h_1, \ldots, h_q)$ where $h_k = \text{Max}((1 - w_k), t_{ik})$ [12].

It is interesting to observe the standard situation is obtained when all $w_k = 1$. In this case we get $\text{Min}_k[t_{ik}]$. We can also observe that since $\text{Max}((1 - w_k), t_{ik}) \geq t_{ik}$ then $\text{Min}_k[\text{Max}((1 - w_k), t_{ik})] \geq \text{Min}_k[t_{ik}]$. Thus the use of importances is effectively to reduce the requirements, we are easing the difficulty. We also observe that more generally we can replace Min with any t-norm T and Max with any t–conorm S thus we can use $\mathop{\mathrm{T}}_{k=1}^{q} [S((1 - w_k), t_{ik})]$ [10].

Consider now the case where we have an "oring" of the criteria. Thus here our aggregation imperative is (P_1, w_1) or (P_2, w_2) or (P_3, w_3) or, ..., (P_q, w_q). In this case the weighted aggregation and the satisfaction is $\text{Max}(g_1, g_2, \ldots, g_q)$ where $g_k = \text{Min}(w_k, t_{ik})$ [10, 13].

We observe the standard situation is obtained when all $w_k = 1$. In this case we get $\text{Max}_k[t_{ik}]$. We can also observe that since $\text{Min}(w_k, t_{ik}) \leq t_{ik}$ then $\text{Max}_k[\text{Min}(w_k, t_{ik})] \leq \text{Max}_k[t_{ik}]$. Thus the use of importances reduces the influence of individual criteria.. We also observe that more generally we can replace Min with any t-norm T and Max with any t–conorm S thus we can use $\mathop{\mathrm{S}}_{k=1}^{q} [T(w_k, t_{ik})]$.

An interesting observation can be made with respect to the weighted "anding" and "oring" observations. In the case of the weighted *oring*, increasing the weights associated with the criteria can only result in an increase in the overall satisfaction while in the case of the "anding" increasing the weights can only result in a decrease in the satisfaction. More formally if we let w_k and $\tilde{w}_k$ be two pairs of weights such that $\tilde{w}_k \geq w_k$ and let $g_k = T(w_k, t_{ik})$ and let $\tilde{g}_k = T(\tilde{w}_k, t_{ik})$ then for any t–conorm S, we have $S(g_1, \ldots, g_q) \leq S(\tilde{g}_1, \ldots, \tilde{g}_q)$. On the other hand if we let $h_k = S((1 - w_k), t_{ik})$ and $\tilde{h}_k = S((1-$

$\tilde{w}_k$), t_{ik}) then for any t-norm T, T(h_1, ..., h_q) ≥ T($\tilde{h}_1$, ..., $\tilde{h}_q$).

We can make the following observation about the weighted "oring" aggregation in the case of the using S = Max. In this case our overall aggregation is Max(g_1, ..., g_q) where g_k = T(w_k, t_{ik}) for a t-norm T. We further observe that if w_k < 1 then for any T and t_{ik} we have g_k < 1. From this we can conclude that in the case of the weighted "or" aggregation using S = Max we can never get an overall satisfaction of one unless at least one at the criteria conditions has importance of one. We note this is a necessary but not sufficient condition for getting an overall satisfaction of one.

A corresponding result can be obtained in the case of the weighted "anding" aggregation in the case of using T = Min. Recall the definition h_k = S($\tilde{w}_k$, t_{ik}) for any t-conorm S, from this we can observe that if w_k < 1, then $\tilde{w}_k$ > 0 and hence for any S and t_{ik} we have h_k ≥ $\tilde{w}_k$ > 0. Since our overall aggregation is Min(h_1, ..., h_q) we can conclude that in this case of weighted "anding" aggregation we can not get an overall satisfaction of zero unless at least one of the criteria conditions has importance weight of one.

An interesting formulation for the weighted "oring" aggregation occurs when we use S(a, b) = Min(a + b, 1) and T(a, b) = a b. In this case (P_1, w_1) or (P_2, w_2) or ...or (P_q, w_q) evaluates to Min($\sum_{k=1}^{q} w_k t_{ik,1}$). Thus here we take a simple weighted sum and then bound it by the value one.

A correspondingly interesting formulation for the weighted "anding" can be had if we use T(a, b) = Max(a + b - 1, 0) and S($\bar{a}$, b) = a + b - ab. In this case

$$T(g_1, ..., g_q) = 1 - \text{Min}[\sum_{k=1}^{q} w_k \bar{t}_{ik}, 1]$$

The OWA operator [14] can provide a useful formulation for the aggregation process in flexible querying of databases. We recall the OWA operator is aggregation operator F: $I^n \rightarrow I$ so that F(a_1, ..., a_n) = Σ_j $w_j b_j$ where b_j is the j^{th} largest of the a_i and w_j are collection of *n* weights such that $w_j \in [0, 1]$ and Σ_j w_j = 1. We note that if π is an index function so π(j) is the index of the j^{th} largest a_i, then $b_j = a_{\pi(j)}$ and hence F(a_1, ..., a_n) = Σ_j w_j $a_{\pi(j)}$. It is common to refer to the collection of w_j using an *n* vector W whose j^{th} component is w_j, in this case we refer to W as the OWA weighting vector.

We observe that if w_1= 1 and w_j = 0 for j ≠ 1 then F(a_1, ..., a_n) = $\text{Max}_i[a_i]$. If w_n = 1 and w_j = 0 for j ≠ n then F(a_1, ..., a_n) = $\text{Min}_i[a]$. If w_j = 1/n for all the j then F(a_1, …, a_n) = $\frac{1}{n}\Sigma_i$ a_i. The OWA operator is a mean operator, it is monotonic, symmetric and bounded, It is also idempotent

The OWA operator can be very directly used in flexible querying. Assume we have a query with n component pairs, P_k = ($V_{Q(k)}$, $F_{Q(k)}$) where $F_{Q(k)}$ is a fuzzy subset over the domain of $V_{Q(k)}$. As in the preceding for any object D_i we can obtain t_{ik} = $F_{Q(k)}$($d_{iQ(k)}$). We can then provide an aggregation of these using the OWA operator, F(t_{i1}, t_{i2}, …, t_{in}). Essentially this provides a kind an average of the individual satisfactions. We can denote this $\text{OWA}_W(D_i)$. Different choices of the OWA weights will result in different types of aggregation. If we use the weights such that w_1 = 1 then we get $\text{Max}_k[t_{ik}]$ which is essentially an "oring" of the components. If we use a weighting vector W such that w_n = 1 we get $\text{Min}_k[t_{ik}]$ which is an "anding" of the conditions. If w_j = 1/n then we are taking a simple average of the criteria satisfactions.

Using quantifiers and particularly linguistic quantifiers we can use the OWA operator to provide a very rich class of aggregation operators [15]. A quantifier or proportion can be seen as a value in the unit interval. In [16] Zadeh generalized the concept of quantifier by introducing the concept of linguistic quantifiers. Examples of these linguistic quantifiers are: *most*, *about half*, *all*, *few* and *more then α percent*. Zadeh [16] suggested that one can represented these linguistic quantifiers as fuzzy subsets of the unit interval. This if $\mathcal{R}$ is a linguistic quantifier we can represent it as a fuzzy subset R of the unit interval so that for any y ∈ I, R(y) is the degree to which the proportion y satisfies the concept $\mathcal{R}$. An important class of quantifiers are regular monotonic quantifiers. These quantifiers have the properties: R(0) = 0, R(1) = 1 and R(x) ≥ R(y) for x > y. Thus for these quantifiers the satisfaction increases as the proportion increases. Examples of these are "at least p", "most", "some", "all" and "at least one". In [15] Yager showed how to use the fuzzy subsets associated with these quantifiers to generate the weights of an OWA operator. In particular he suggested we obtain the weights as

$$w_j = R(\frac{j}{n}) - R(\frac{j-1}{n}).$$

It can be shown if R is regular monotonic then the weights sum to one and lie in the unit interval.

Using these ideas we can obtain a quantifier guided approach to flexible querying. We can express a query as collection of n pairs, P_k = ($V_{Q(k)}$, $F_{Q(k)}$) and an aggregation imperative in terms of a linguistic quantifier $\mathcal{R}$ describing the proportion of these pairs we require to be satisfied. We then express this quantifier $\mathcal{R}$ as a fuzzy subset of the unit interval, R. Using this R we obtain a collection of OWA weights, w_j = $R(\frac{j}{n}) - R(\frac{j-1}{n})$ for j = 1 to n. Once having these weights we can evaluate the satisfaction of each element D_i as $\text{OWA}_R(D_i)$. In [17] we showed how to extend this to the case importance weighted criteria.

4. Lexicographic formulated queries

In the database query structure presented here we impose a priority ordering on the conditions composing the query. In this approach conditions higher in priority ordering play a

more important role in determining the overall satisfaction of an object to the query in the same way that letters in earlier positions a play in determining the alphabetical order. The technique we shall develop will be called a **LEX**icographic **A**ggregation (LEXA) query and it will make use of the prioritized aggregation operator introduced in [18-20]. This approach will allow us to model different query imperatives than in the preceding.

Here again we shall consider a query Q to consist of a collection of q pairs P_k and an aggregation imperative. Each pair as in the preceding is of the form $(V_{Q(k)}, F_{Q(k)})$ where $V_{Q(k)}$ indicates an attribute name and $F_{Q(k)}$ is a fuzzy subset of the domain of $V_{Q(k)}$ denoting the desired values for $V_{Q(k)}$. Again for each object $D_i = (d_{i1}, ..., d_{iq})$ we can obtain the values $t_{ik} = F_{Q(k)}(d_{ik})$, the satisfaction of the pair P_k by D_i. Here we will consider an aggregation imperative in the spirit of a lexicographic ordering. This aggregation imperative requires an ordering of the P_k indicating their priority in formulating the overall satisfaction to the query. While our lexicographic aggregation query framework will work in the flexible environment we shall initially consider its performance in the binary situation.

We now describe the basic binary LEXA query. In this case it is assumed that the t_{ik} are binary either 1 or 0, P_k is satisfied by D_i or not. In the following we shall assume there are m elements in the database, D_i for i = 1 to m. As we subsequently see the output of a LEXA query is an ordering of the elements in D_i according to their satisfaction to lexicographic query.

A lexicographic aggregation query assumes a linear priority among the criteria, the P_k, with respect to their importance. For simplicity we shall assume that the P_k have been indexed according to this priority ordering:

$$P_1 > P_2 > ... > P_q$$

The following algorithm describes the process for forming the ordered list L

1) Initiate: k = 1, D = $\{D_i \mid i = 1 \text{ to } m\}$, list L empty
2) Set F = ∅
3) Test all elements $D_i \in D$ with respect P_k and place all those for which $t_{ik} = 0$ in F.
4) Place all elements in F tied at the top available level of L.
5) Set D = D - F
6) If D = ∅ Stop
7) Set k = k + 1
8) If k > q
   a) Place all remaining elements in D tied as top level of L
   b) Stop
9) Go to step 2

The result of this is the list L which is an ordered list of the satisfactions of the elements in D to the lexicographic query Q. We note the higher up the list the better the satisfaction.

There is another way we can generate the satisfaction ordering of the elements in D under the priority ordering $P_1 > ... > P_q$. For each D_i, starting from P_1 and preceding in increasing order find the first index k for which $t_{ik} = 0$. Assign D_i a score S_i where

$S_i = q$ if D_i meets no failures
$S_i = k - 1$ otherwise

Using this method each D_i gets a score $S_i \in \{0, 1, ..., q\}$. If we order the elements by their value for S_i we get L. We note that S_i is the number of criteria D_i satisfied before it meets failure. This method for evaluating the score of satisfaction of an element D_i to a LEXA query is more useful then the preceding as it provides a score is addition to an ordering.

It is interesting to note there is only one situation in which an element D_i can attain the maximal score $S_i = q$, if all $t_{ik} = 1$. On the other hand there are many ways D_i can get the minimal score of $S_i = 0$. In particular any D_i which has $t_{i1} = 0$ will have $S_i = 0$ regardless of the values of t_{ik} for $k \neq 1$.

We further observe that since $0 \leq S_i \leq q$ we can provide a kind of normalization by defining $G_i = \frac{S_i}{q}$. Here we are getting a value in the unit interval.

We shall introduce a more general implementation of the lexicographic aggregation type query that will be appropriate for the case where the t_{ik} are values from [0, 1] rather then being binary values from {0, 1}. In this environment for each element D_i we shall obtain a score G_i indicating its satisfaction to our LEXA query.

Again here we have a query Q consisting of q condition pairs, $P_k = (V_{Q(k)}, F_{Q(k)})$. In addition we have a priority ordering over the pairs guiding the lexicographic aggregation,

$$P_1 > P_2 > P_3 ... > P_q$$

Assume for object D_i we have $t_{ik} = F_{Q(k)}(d_{ik})$ as the degree to which D_i satisfies P_k. Our procedure for determining G_i is to calculate

$$G_i = \sum_{k=1}^{q} w_{ik} t_{ik}$$

Where the w_{ik} are determined as follows.

1) Set $u_{i1} = 1$
2) $u_{ik} = \prod_{j=2}^{k} t_{i(j-1)}$ for k = 2 to q
3) $w_{ik} = \frac{1}{q}\ u_{ik}$

The important observation here is that the w_{ik} is proportional to the product of the satisfactions of the higher priority conditions. We note that we can express $u_{ik} = u_{ik-1} H_{ik-1}$.

We can make some observations. For $k_1 < k_2$ we always have $w_{ik_1} \geq w_{ik_2}$. Thus a lower priority condition can't have a bigger weight than one that is higher.

We observe that if $t_{ik_1} = 0$ then $w_{ik_2} = 0$ for all $k_2 > k_1$

We can show that $G_i = 1$ if and only $t_{ik} = 1$ for all k.

We also observe that $G_i = 0$ iff $t_{i1} = 0$. It is important to emphasize that this is independent of the satisfaction to the other criteria. Zero satisfaction to the highest priority criterion means zero overall satisfaction, there is no possibility for compensation by other criteria.

Let us look at this process for the binary case to see that it correctly generalizes the evaluation in the binary case we presented earlier. Consider the list of satisfactions: t_{i1}, t_{i2}, ..., t_{iq} which here we assume are either one or zero. Assume t_{ik} is the first one of these that is zero, hence $t_{ij} = 1$ for $j < k$. In this case we get that $u_{ij} = 1$ for $j \le k$ and $u_{jk} = 0$ for $j > k$.

Hence

$$G_i = \sum_{j=1}^{q} w_{ij}t_{ij} = \frac{1}{q}\sum_{j=1}^{q} u_{ij}t_{ij} = \frac{1}{q}\sum_{j=1}^{q} t_{ij}$$

Furthermore since $t_{ij} = 1$ for $j < k$ and $t_{ij} = 0$ for $j = k$ we get

$$G_i = \frac{1}{q}\sum_{j=1}^{k} t_{ij} = \frac{1}{q}\sum_{j=1}^{k-1} 1 = \frac{k-1}{q}$$

as desired. In the special case where all $t_{ij} = 1$ then all $u_{ij} = 1$ and all $w_{ij} = \frac{1}{q}$ and hence $G_i = \frac{q}{q} = 1$.

As discussed in [18] the approach we suggested can be extended to the case when the ordering among the pairs is a weak ordering, we allow ties among the conditions pairs with respect to their priority. Here we shall denote a condition pair as $P_{(k, j)}$ where all pairs with the same k value are tied in the priority ordering. The j value is just an indexing distinguishing among the tied pairs. Thus here we are assuming the priority ordering is such that for $k_1 < k_2$ that $P_{(k_1, j)} > P_{(k_2, i)}$ and $P_{(k_1, j)} = P_{(k_1, i)}$ for all j and i. We let n_k denote the number of pairs with k in the first term, thus n_1 is the number of conditions with the highest priority. We shall also let $n = \sum_{k=1}^{q} n_k$.

Here we let $t_{i(k, j)}$ denote the satisfaction of the object D_i to the condition pair $P_{(k, j)}$. In this case we obtain the value G_i as

$$G_i = \sum_{k=1}^{q} w_{ik}\left(\sum_{r=1}^{n_k} t_{i(j,r)}\right)$$

Again it is a weighted sum of the satisfaction to each of the pairs. We emphasize that each pair with the same k has the same weight w_{ik}. The procedure we use to obtain the weights is similar to the earlier one except in the first step:

1. Calculate: $H_{ij} = \underset{r=1 \text{ to } n_j}{\text{Max}} [t_{i(j,r)}]$

(It is the satisfaction value of the least satisfied pair at the j level)

2. $u_{i1} = 1$

$$u_{ik} = \prod_{j=1}^{k} H_{i(j-1)} \quad \text{for } k = 2 \text{ to } q$$

3. $w_{ik} = \frac{1}{n} u_{ik}$

Let us see how this plays out in the pure binary case where all $t_{i(k, r)} \in \{0, 1\}$. Let b be the k value at which we meet the first condition for which $t_{i(k, r)} = 0$. In this case we have that $H_{ij} = 1$ for $j < b$ and $H_{ij} = 0$ for $j = 0$. Here then while $u_{i1} = 1$ we have for $k = 2$ to q that $u_{ik} = \prod_{j=1}^{n} H_{i(j-1)}$.

From the above we get that

$u_{ik} = 1$ $\quad 2 \le k \le b$

$u_{ik} = 0$ $\quad k > b$

Thus we get $w_{ik} = \frac{1}{n}$ for $k = 1$ to b and $w_{ik} = 0$ for $k > b$.

From this we get $G_i = \sum_{k=1}^{q} w_{ik}\left(\sum_{r=1}^{n_k} t_{i(k,r)}\right) = \sum_{k=1}^{b} \frac{1}{n}\left(\sum_{r=1}^{n_k} t_{i(k,r)}\right)$

For all $k < b$ all the elements have $t_{i(k,r)} = 1$ hence

$$G_i = \frac{1}{n}\left(\sum_{k=1}^{b-1} n_k + \sum_{r=1}^{n_b} t_{i(b,r)}\right)$$

We note at least one element in the second term is zero. We see that the numerator of G_i is equal to the number of elements in the priority classes higher than the first class where we meet failure plus all the pairs satisfied in the class that we meet failure. An alternative expression of this numerator is the sum of all satisfied pairs in priority classes up to and including the class when we meet our first failure.

We here point to related work by Chomicki [22-26] on preferences in databases and the on bipolar queries by Dubois and Prade [27, 28] and Zadrozny and Kacprzyk [29, 30]

5. Querying probabilistic databases

In the preceding we have discussed several different paradigms for formulating questions to databases. In particular we considered a query to be a collection of q condition pairs, $P_k = (V_{Q(k)}, F_{Q(k)})$, and an agenda for aggregating the satisfactions to these pairs by a database element D_i. We added flexibility by allowing $F_{Q(k)}$ to be a fuzzy subset of the domain $X_{Q(k)}$ of $V_{Q(k)}$. In this flexible case we obtained the degree of satisfaction of P_k by D_i to be $t_{ik} = F_{Q(k)}(x_{iQ(k)}) \in [0, 1]$ where $x_{iQ(k)}$ is the value of $V_{Q(k)}$ for D_i. Thus t_{ik} is the degree to which $F_{Q(k)}$ is true given the value of $V_{Q(k)}$ for D_i is $x_{iQ(k)}$. We now shall consider the situation in which the value of $V_{Q(k)}$ for D_i, $x_{iQ(k)}$, is random. More specifically if the domain $X_{Q(k)} = \{y_{Q(k)1}, y_{Q(k)2}, ..., y_{Q(k)r_{Q(k)}}\}$ then the knowledge of the value of $V_{Q(k)}$ for object D_i, $x_{iQ(k)}$, is best expressed as a probability distribution $\mathbf{P_{iQ(k)}}$ where $\text{Prob}(x_{iQ(k)} = y_{Q(k)j}) = p_{j/iQ(k)}$. In the special case where

$P_{j/qQ(k)} = 1$ for some j then we have the preceding situation in which $x_{iQ(k)}$ is exactly $y_{Q(k)j}$.

Initially in the following discussion we shall assume all the Q(k) are distinct. We shall subsequently consider the case where a query can involve multiple occurrences of the same attribute.

The approach we shall follow in this probabilistic situation is in the spirit of the possible words approach used in [31-39]. We shall associate with every D_i a collection Z of q tuples. In particular $Z = X_{Q(1)} \times X_{Q(2)} \times ..... \times X_{Q(q)}$. That is each element $z \in Z$ is of the form $z = (z_1, z_2, ..., z_q)$ where $z_k \in X_{Q(k)}$. For each object D_i in the database we now associate a probability distribution $\mathbf{P D_i}$ over the space Z so that the probability of z, $PD_i(z) = \prod_{k=1}^{q} \text{Prob}(x_{iQ(k)} = z_k)$.

We want to make one comment here. In the preceding we assumed all the $V_{Q(k)}$ in a query where distinct. This is not a necessary requirement. However in the case where the we have a multiple occurrences of the same attribute in the query care must be taken in the formulation of the possible worlds. In the preceding we formulated $Z = X_{Q(1)} \times X_{Q(2)} \times X_{Q(q)}$, here each element $z \in Z$ is of the form $z = (z_1, z_2, , z_q)$ where each $z_k \in X_{Q(k)}$. In order to understand which happens when all $V_{Q(k)}$ are not distinct we consider the situation where Q(1) = Q(2), here then $V_{Q(1)} = V_{Q(2)}$. Here the associated Z space has the additional required that for all $z \in Z$, $z_1 = z_2$. Thus any element in Z must have the same value of $V_{Q(1)}$ and $V_{Q(2)}$. In addition we must make some modifications in the calculation of $PD_i(z)$. In particular we must note duplicate the probabilities and avoid using them twice, thus $PD_i(z)$ must be based on the product of the distinct probabilities and hence we have

$$PD_i(z) = \prod_{\substack{k=1 \text{ for all} \\ \text{distinct } Q(k)}}^{q} \text{Prob}(x_{iQ(k)} = z_k)$$

Another comment we want to make is regarding the relationship between distinct attribute values. Implicit in the preceding has been an assumption of independence between the probability distribution of distinct attributes. In some cases this may not be true. For example some values of V_1 may not be allowable under V_2 while some values for V_1 may mandate a particular value for V_2. Such relationships require us to modify to possible elements in Z and condition the probability distribution in known ways. Nevertheless when including these special conditions we still end up with a subset of tuples from Z with associated probabilities, as a result in the following we shall neglect any special relationships between the attributes as they don't effect the subsequent discussion nor the basic ideas of the approach introduced.

We now recall that our query consists of a collection of q constraints $P_k = (V_{Q(k)}, F_{Q(k)})$ where each $F_{Q(k)}$ is a fuzzy subset over the space $X_{Q(k)}$. We now apply these constraints to the space of possible worlds Z. In particular we transform the space Z to F so that each tuple $z = (z_1, ..., z_q)$ is transformed to new tuple

$$F(z) = (F_{Q(1)}(z_1), F_{Q(2)}(x_2), , F_{Q(q)}(z_q))$$

We note each element $F_{Q(k)}(x_k) \in [0, 1]$. Thus each term in F(z) is a tuple of q values drawn from the unit interval, F(z) is a subset the space I^q. In addition for each element D_i in the database we can associate a probability distribution over the space F. In particular for each D_i we associate $PD_i(z)$ with the tuple F(z). Thus now for any element D_i we have a probability distribution on the space F of satisfactions to the query components. We note that if two z tuples, z_1 and z_2, transform into the same value, $F(z_1) = F(z_2)$, we represent these in F with just one, $F(z_1)$, and use the sum of the probabilities z_1 and z_2

In order to provide an intuitive understanding of the discussion to follow we shall use the following database to illustrate our ideas

Example 3. We consider a database with three attributes V_1, V_2 and V_3. The domain of these are respectfully:

$$X_1 = \{a, b, c\}, X_2 = \{red, blue\}, X_3 = \{10, 20\}$$

Let D be one object in the database and let x_1, x_2, x_3 denote the values of V_1, V_2, V_3 for this object. In particular for each of these, x_1, x_2, x_3, we have a probability distribution

x_1: Prob(a) = 0.4, Prob(b) = 0.5 and Prob(c) = 0.1
x_2: Prob(red) = 0.7 and Prob(blue) = 0.3
x_3: Prob(10) = 0.6 and Prob(20) = 0.4

In this example the set of Z of possible worlds are Z = {(a, red, 10), (a, red, 20), (a, blue, 10), (a, blue, 20), (b, red, 10), (b, red, 20), (b, blue, 10), (b, blue, 20), (c, red, 10), (c, red, 20), (c, blue, 10), (c, blue, 20)}.

For the element D we get the probability of each of these components as shown in Table 1.

Before preceding we want to point out that for any other object D* in the database the set of possible worlds Z will be the same, the difference between D and D* will be in the probabilities associated with the elements in Z.

We now consider our query as consisting of three components: $P_1 = (V_1, F_1)$, $P_2 = (V_2, F_2)$ and $P_3 = (V_3, F_3)$ where each F_k is a fuzzy set over the space X_k defining the required condition. Below are the associated fuzzy subsets

$$F_1 = \{\frac{1}{a}, \frac{0.6}{b}, \frac{0.2}{c}\}, F_2 = \{\frac{0.8}{red}, \frac{1}{blue}\}, F_3 = \{\frac{1}{10}, \frac{0.2}{20}\}$$

We now can apply these constraints to our set Z of possibilities and also use the probabilities for D and shown the results in Table 2.

Table 1. Probabilities of Elments in Z

Element in Z	Probability
(a, red, 10)	(0.4) (0.7) (0.6) = 0.167
(a, red, 20)	(0.4) (0.7) (0.4) = 0.112
(a, blue, 10)	(0.4) (0.3) (0.6) = 0.072
(a, blue, 20)	(0.4) (0.3) (0.4) = 0.048
(b, red, 10)	(0.5) (0.7) (0.6) = 0.21

(b, red, 20)	(0.5) (0.7) (0.4) = 0.14
(b, blue, 10)	(0.5) (0.3) (0.6) = 0.09
(b, blue, 20)	(0.5) (0.3) (0.4) = 0.06
(c, red, 10)	(0.1) (0.7) (0.6) = 0.042
(c, red, 20)	(0.1) (0.7) (0.4) = 0.028
(c, blue, 10)	(0.1) (0.3) (0.6) = 0.018
(c, blue, 20)	(0.1) (0.3) (0.4) = 0.012

Table 2. Probabiities of Transformed Elements

Element in Z	Transform F(z)	Probability
(a, red, 10)	(1, 0.8, 1)	0.167
(a, red, 20)	(1, 0.8, 0.2)	0.112
(a, blue, 10)	(1, 1, 1)	0.072
(a, blue, 20)	(1, 1, 0.2)	0.048
(b, red, 10)	(0.6, 0.8, 1)	0.21
(b, red, 20)	(0.6, 0.8, 0.2)	0.14
(b, blue, 10)	(0.6, 1, 1)	0.09
(b, blue, 20)	(0.6, 1, 0.2)	0.06
(c, red, 10)	(0.2, 0.8, 1)	0.042
(c, red, 20)	(0.2, 0.8, 0.2)	0.028
(c, blue, 10)	(0.2, 1, 1)	0.018
(c, blue, 20)	(0.2, 1, 0.2)	0.012

We now consider the issue of evaluating the query Q for the object D in the database. Let us recapitulate the situation. Associated with D we have a space F of tuples $T_j = (F_1(z_1), \ldots, F_q(z_q))$, each tuple is an element from the space I^q. That is each tuple is a collection of q values that the unit interval. In addition associated with each tuple in F we have a probability. Parenthetically we note that the space F of tuples is the same for each element D_i in the database, the only distinction between the elements in the database are the probabilities associated with each of the tuples.

At this point we can reformulate more simply. We have a subset $F \subseteq I^q$ with arbitrary element $T_j = (t_{j1}, t_{j2}, \ldots, t_{jq})$. Furthermore for any database element D_i we have a probability distribution over the space F where p_{ij} is the probability associated with the tuple T_j for the database object D_i. We note that $\sum_{j=1}^{n_z} p_{ij} = 1$ where n_z is the cardinality of the space Z.

In addition associated with a query is an aggregation imperative that dictates how we aggregate the satisfactions to the individual components, in particular $Agg(T_j) = Agg(t_{j1}, t_{j2}, \ldots, t_{jq})$. After applying the Agg operation to the tuples in the space F we end up with a collection of scalar values, Agg(F), consisting of the elements $Agg(T_j)$. For each D_i we have a probability distribution over the set of $Agg(T_j)$. In particular for each D_i and each $Agg(T_j)$ we have p_{ij} as its probability. We now illustrate the preceding with our earlier example and consider the application of different aggregation imperatives.

Example 4. We have the collection of 12 tuples and their associated probabilities. In Table 3 we consider three aggregation imperatives the first imperative is an "anding" of all conditions, here $Agg(T_j) = Min_k(t_{jk})$, the second imperative is an average of all conditions, $Agg(T_j) = \frac{1}{q}\sum_{j=1}^{q} t_{jk}$ and the third imperative is an "oring" of all conditions, here $Agg(T_j) = Max_k(t_{jk})$. .

Table 3. Aggregated Value of Elements in F

Tuple in F	Probability	and	Average	oring
T_1 = (1, 0.8, 1)	0.167	0.8	0.933	1
T_2 = (1, 0.8, 0.2)	0.112	0.2	0.66	1
T_3 = (1, 1, 1)	0.072	1	1	1
T_4 = (1, 1, 0.2)	0.048	0.2	0.73	1
T_5 = (0.6, 0.8, 1)	0.21	0.6	0.9	1
T_6 = (0.6, 0.8, 0.2)	0.14	0.2	0.53	0.8
T_7 = (0.6, 1, 1)	0.09	0.6	0.866	1
T_8 = (0.6, 1, 0.2)	0.06	0.2	0.6	1
T_9 = (0.2, 0.8, 1)	0.042	0.2	0.66	1
T_{10} = (0.2, 0.8, 0.2)	0.028	0.2	0.4	0.8
T_{11} = (0.2, 1, 1)	0.018	0.2	0.73	1
T_{12} = (0.2, 1, 0.2)	0.012	0.2	0.466	1

In a similar way we can implement any aggregation imperative.

Combining the probabilities of tuples with the same value for their ***anding*** we get Table 4.

Table 4. Probabilities of "anded" Value

Value	Prob
0.2	0.461
0.6	0.3
0.8	0.167
1	0.072

Combining the probabilities of the tuples with the same value for their ***oring*** we get Table 5

Table 5. Probabilities of "ored" Value

Value	Prob
1	0.82
0.8	0.168

Let us now summarize our situation for a query we obtained a collection T_j of tuples. Associated with the query is an aggregation operation that converts T_j into a single value $Agg(T_j)$ indicating the overall satisfaction of the tuple. For simplicity we denote these as $Agg(T_j) = a_j$. Essentially we have a collection of degrees of satisfaction, a_j, of each possible world to the query. Finally for each element D_i in the database we have a probability distribution over the a_j. Thus the satisfaction of the query by the database object D_i is a probability distribution

a_1 p_{i1}
a_2 p_{i2}
.....
a_r p_{ir}.

We emphasize that the set a_j, which are the satisfactions of a possible world to the query, is the same for all D_i, the difference between the D_i is reflected in the probability

distribution over the a_j, the probabilities they assign to a possible world.

A natural question that arises is how to order the D_i with respect to their satisfaction to the query. Here we look to [40] for some ideas. If D and $\hat{D}$ are two database objects then for each of these we have a probability distribution over the set A = $\{a_1, ..., a_r\}$.

	P	$\hat{\mathbf{P}}$
a_1	p_1	$\hat{P}_1$
a_2	P_2	$\hat{P}_2$
a_j	p_j	$\hat{P}_j$
a_r	p_r	$\hat{P}_r$

In the following for simplicity we shall assume the a_j have been indexed so that they are in increasing order, $a_j > a_k$ if $j > k$. In the following we shall use the notation $D > \hat{D}$ if D is a more satisfying alternative than $\hat{D}$. What is clear is that if $p_j = 1$ and $\hat{P}_k = 1$ and $j > k$ than $D > \hat{D}$.

One approach to ordering the P and $\hat{P}$ is to use the cumulative distribution function, CDF, and the idea of stochastic dominance [41, 42].

We define $CDF_P(a_j) = \sum_{i=1}^{j} p_i$, this the probability that for object D the actual satisfaction will be less or equal a_j. Similarly we define $CDF\,\hat{p}\,(a_j) = \sum_{i=1}^{j} \hat{p}_i$. Using this we shall say $D > \hat{D}$ if

$CDF_P(a_j) \le CDF\,\hat{p}\,(a_j)$ for all j

$CDF_P(a_j) < CDF\,\hat{p}\,(a_j)$ for at least one j

We note that if $CDF_P(a_j) \le CDF\,\hat{p}\,(a_j)$ then $\sum_{i=1}^{j} p_i \le \sum_{i=1}^{j} \hat{p}_i$ this implies

$$1 - \sum_{i=j+1}^{r} p_i \le 1 - \sum_{i=j+1}^{r} \hat{p}_i$$

which further implies that $\sum_{i=j+1}^{r} p_i \ge \sum_{i=j+1}^{r} \hat{p}_i$. Thus we see that under this condition object D never has less probability associated with the higher satisfaction then $\hat{D}$. This provides a reasonable justification for asserting that $D > \hat{D}$.

While the CDF provides a way for ordering the elements if the conditions are met, usually the condition $CDF_P(a_j) \le CDF\,\hat{p}\,(a_j)$ is not satisfied for all j. This effectively means that the use of stochastic dominance does not provide the kind of general approach necessary to cover all cases.

Another approach to comparing the two database elements is a scalarization. Here we associate with each element a distinct value and then compare these values. Since these are scalar values we are able to order them. One approach to scalarization is to use the expected value of satisfaction of each the elements. Here $EV(D) = \sum_{j=1}^{r} a_j p_j$ and $EV(\hat{D}) = \sum_{j=1}^{r} a_j \hat{p}_j$. We then say $D > \hat{D}$ if $EV(D) > EV(\hat{D})$. If $EV(D) = EV(D)$ then we say they are tied.

It is interesting to show that if $CDF_P(a_j) \le CDF\,\hat{p}\,(a_j)$ for all j then $EV(D) \ge EV(\hat{D})$. We shall illustrate this for the case when r = 4, the extension to the more general case of r will be obvious. Since we assumed $a_i > a_j$ for $i > j$ then we can express

$a_2 = a_1 + \Delta_2$ $\quad \Delta_2 > 0$

$a_3 = a_2 + \Delta_3$ $\quad \Delta_3 > 0$

$a_4 = a_3 + \Delta_4$ $\quad \Delta_4 > 0$

We see

$$EV(D) = \sum_{j=1}^{4} a_j p_j = a_1p_1 + (a_1 + \Delta_2)p_2 + (a_1 + \Delta_2 + \Delta_3)p_3 + (a_1 + \Delta_2 + \Delta_3 + \Delta_4)p_4$$

$$EV(D) = a_1 \sum_{j=1}^{4} p_j + \Delta_2 \sum_{j=2}^{4} p_j + \Delta_3 \sum_{j=3}^{4} p_j + \Delta_4 \sum_{j=4}^{4} p_j$$

As we have already shown if $CDF_P(a_i) \le CDF\,\hat{p}\,(a_i)$ for all i then $\sum_{j=4}^{4} p_j \ge \sum_{j=4}^{4} \hat{p}_j$ for all i. From this we see that if $CDF_P(a_i) \le CDF\,\hat{p}\,(a_i)$ for all i then $EV(D) \ge EV(\hat{D})$.

A more general approach to scalarization of the probabilities of the database elements can be had if we use some ideas from the OWA aggregation operator. Here we let $\alpha \in [0, 1]$ be a measure our optimism. The more optimistic the more we are anticipating the higher valued satisfactions to occur. Here then if $\alpha = 1$ we are always anticipating that a_r will occur. While if $\alpha = 0$ we are always anticipating a_1 will occur.

We now associate with a given α a function $f(x) = x^q$ where $q = \frac{1-\alpha}{\alpha}$. We see that if $\alpha \to 0$ then $q \to \infty$ and if $\alpha \to 1$ then $q \to 0$ and if $\alpha = 1/2$ then $q = 1$.

Using this function we obtain a set of OWA weights as follows. Letting $S_j = \sum_{k=i}^{r} p_k$ we get

$$w_j = f(S_j) - f(S_{j+1}) \qquad \text{for } j = 1 \text{ to } r_r$$

Using these weights we obtain $EV_\alpha(D) = \sum_{j=1}^{r} w_j a_j$. We first see that if $\alpha = 1/2$ then $q = 1$ and hence $w_j = \sum_{k=j}^{r} p_k - \sum_{k=j+1}^{r} p_k = p_j$. Here we get the usual expected

value. We see that if $\alpha \to 0$, $q = \infty$, since $S_j < 1$ for all $j > 1$ we have $(S_j)^{\infty} \to 0$ for $j < 1$ and $(S_i)^{\infty} = 1$ hence in this case

$w_1 = 1$

$w_j = 0$ for $j > 1$

From this we obtain $EV_0(D) = a_1$, it is the least satisfaction. If $\alpha \to 1$, $q = 0$, $w_r = 1$, if $p_r > 0$ and hence $ED_1(D) = a_r$. Here we see that in using EV_α for the extreme optimistic and pessimistic of α all database elements D_k will have the same value for $EV_\alpha(D_k)$ however for other values $0 < \alpha < 1$ as in the case of $\alpha = 0.5$ each of the D_k will get its own unique value for $EV_\alpha(D_k)$. Here then choosing α will determine the ordering of the D_k.

6. Conclusion

Here we provided a framework for soft querying of databases. We described a soft query as a collection of required conditions and an imperative for combining an objects satisfaction to individual conditions to get its overall satisfaction. We investigated tools that can enrich this process by enabling the inclusion of more human focused considerations. We described some more sophisticated techniques for aggregating the satisfactions of the individual conditions based on the inclusion of importances and the use of the OWA operator.. In addition to considering more human focused aspects of the query we looked at databases in which the information in the database can have some uncertainty. We particularly considered probabilistic

7. References

[1]. Kacprzyk, J. and Ziolkowsi, A., "Database queries with fuzzy linguistic quantifiers," IEEE Transactions on Systems, Man and Cybernetics 16, 474-479, 1986.

edited by Prade, H. and Negoita, C. V., Verlag TUV Rheinland: Cologne, 46-57, 1986.

[2]. Bosc, P., Galibourg, M. and Hamon, G., "Fuzzy quering with SQL: extensions and implementation aspects," Fuzzy Sets and Systems 28, 333-349, 1988.

[3]. Bosc, P. and Pivert, O., "SQLf: a relational database language for fuzzy quering," IEEE Transactions on Fuzzy Systems 3, 1-17, 1995.

[4]. Kraft, D. H., Bordogna, G. and Pasi, G., "Fuzzy set techniques in information retrieval," in Fuzzy Sets in Approximate Reasoning and Information Systems, edited by Bezdek, J. C., Dubois, D. and Prade, H., Kluwer Academic Publishers: Norwell, MA, 469-510, 1999.

[5]. Galindo, J., Handbook of Research on Fuzzy Information Processing in Databases, Information Science Reference: Hershey. PA, 2008.

[6]. Zadrozny, S., de Tré, G., de Caluwe, R. and Kacprzyk, J., "An overview of fuzzy approaches to flexible database querying," in Handbook of Research on Fuzzy Information Processing in Databases Vol. 1, edited by Galindo, J., Information Science Reference: Hershey. PA, 34-54, 2008.

[7]. Petry, F. E., Fuzzy Databases Principles and Applications, Kluwer: Boston, 1996.

[8]. Pivert, O. and Bosc, P., Fuzzy Preference Queries to Relational Databases, World Scientific: Singapore, 2012.

[9]. Beliakov, G., Pradera, A. and Calvo, T., Aggregation Functions: A Guide for Practitioners, Springer: Heidelberg, 2007.

[10]. Yager, R. R., "A note on weighted queries in information retrieval systems," J. of the American Society of Information Sciences 38, 23-24, 1987.

[11]. Dubois, D., Prade, H. and Testemale, C., "Weighted fuzzy pattern matching," Fuzzy Sets and Systems 28, 313-331, 1988.

[12]. Yager, R. R., "A new methodology for ordinal multiple aspect decisions based on fuzzy sets," Decision Sciences 12, 589-600, 1981.

[13]. Sanchez, E., "Importance in knowledge systems," Information Systems 14, 455-464, 1989.

[14]. Yager, R. R., "On ordered weighted averaging aggregation operators in multi-criteria decision making," IEEE Transactions on Systems, Man and Cybernetics 18, 183-190, 1988.

[15]. Yager, R. R., "Quantifier guided aggregation using OWA operators," International Journal of Intelligent Systems 11, 49-73, 1996.

[16]. Zadeh, L. A., "A computational approach to fuzzy quantifiers in natural languages," Computing and Mathematics with Applications 9, 149-184, 1983.

[17]. Yager, R. R., "Including importances in OWA aggregations using fuzzy systems modeling," IEEE Transactions on Fuzzy Systems 6, 286-294, 1998.

[18]. Yager, R. R., "Prioritized aggregation operators," International Journal of Approximate Reasoning 48, 263-274, 2008.

[19]. Yager, R. R., "Lexicographic ordinal OWA aggregation of multiple criteria," Information Fusion 11, 374-380, 2010.

[20]. Yager, R. R., Reformat, M. and Ly, C., "Using a web personal evaluation tool - PET for Lexicographic multi-criteria service selection," Knowledge-Based Systems 24, 929-942, 2011.

[21]. Chomicki, J., "Preference formulas in relational queries," ACM Transactions on Database Systems 28, 427-466, 2003.

[22]. Chomicki, J., "Database querying under changing preferences," Annals of Mathematics and Arti cial Intelligence 50, 79-109, 2007.

[23]. Mindolin, D. and Chomicki, J., "Preference elicitation in prioritized skyline queries.," Very Large Data Base Journal 20, 157-182, 2011.

[24]. Mindolin, D. and Chomicki, J., "Contracting Preference Relations for Database Applications.," Artificial Intelligence Journal 175, 1092-1121, 2011.

[25]. Staworko, S., Chomicki, J. and Marcinkowski, J., "Prioritized repairing and consistent query answering in relational databases," Annals of Mathematics and Artificial Intelligence, 64(2-3, 209-246, 2012.

[26]. Staworko, S., Chomicki, J. and Marcinkowski, J., "Prioritized repairing and consistent query answering in relational databases," Annals of Mathematics and Artificial Intelligence, 64(2-3, 209-246, 2012.

[27]. Dubois, D. and Prade, H., "Bipolarity in flexible querying," Proceedings of the 5th International Conference on Flexible Query Answering Systems, 174-182, 2002.

[28]. Dubois, D. and Prade, H., "Handling bipolar queries in fuzzy information processing," in Handbook of Research on Fuzzy Information Processing in Databases Vol. 1, edited by Galindo, J., Information Science Reference: Hershey, PA, 99-114, 2008.

[29]. Zadrozny, S., "Bipolar queries revisited," in Modeling Decisions for Artificial Intelligence, LNCE 0302-9743, Springer, Heidelberg, 387-398, 2005.

[30]. Zadrozny, S. L. and Kacprzyk, J., "Bipolar queries and queries with preferences," Proceedings of the 17th International Conference on Database and Expert Systems Applications, 415-419, 2006.

[31]. Cavallo, R. and Pittarelli, M., "The theory of probabilistic databases," Proc. of 13th Int. Conference Very Large Databases (VLDB), 71-81, 1987.

[32]. Barbará, D., Garcia-Molina, H. and Porter, D., "The management of probabilistic data," IEEE Transactions on Knowledge and Data Engineering 4, 487-502, 1992.

[33]. Re, C., Dalvi, N. and Suciu, D., "Query evaluation on probabilistic data bases," IEEE Data Engineering Bulletin 29, 25-31, 2006.
[34]. Suciu, D., "Probabilistic databases," SIGACT News 39, 111-124, 2008.
[35]. Re, C. and Suciu, D., "Management of data with uncertainties," CIKM, Lisbon, 3-8, 2008.
[36]. Dalvi, N., Re, C. and Suciu, D., "Probabilistic databases: diamonds in the dirt," Journal of the Association of Computing Machinery 52(7), 86-94, 2009.
[37]. Dalvi, N., Re, C. and Suciu, D., "Queries and materialized views on probabilistic databases," J. Comput. Syst. Sci. 77(3), 473-490, 2011.
[38]. Suciu, D., Olteanu, D., Re, C. and Koch, C., Probabilistic Databases. Synthesis Lectures on Data Management, Morgan & Claypool Publishers: San Rafael, CA, 2011.
[39]. Dalvi, N. and Suciu, D., "The dichotomy of probabilistic inference for unions of conjunctive queries," Journal of the Association of Computing Machinery 59(6), 30, 2012.
[40]. Yager, R. R., "Ordering ordinal probability distributions," Fuzzy Economic Review 6, 3-18, 2001.
[41]. Hadar, J. and Russell, W., "Rules for ordering uncertain prospects," American Economic Review 59, 25-34, 1969.
[42]. Bawa, V. S., "Optimal rules for ordering uncertain prospects," Journal of Financial Economics 2, 95-121, 1975.

ACKNOWLEDGEMENT

Ronald Yager was supported by a Multidisciplinary University Research Initiative (MURI) grant (Number W911NF-09-1-0392) and by the ONR grant award number N00014-13-1-0626. We gratefully appreciate this support.