Comparing priority, weighted and queries with threshold in PFSQL

Aleksandar Takačić  
Faculty of Technology  
University of Novi Sad  
Serbia  
atakaci@uns.ns.ac.yu

Srdjan Škribić  
Faculty of Natural Sciences  
University of Novi Sad  
skhrba@uns.ns.ac.yu

Abstract—PFSQL is the query language used for querying fuzzy relational databases. One of the most recognizable features of PFSQL is the possibility to prioritize conditions. Priorities are most often confused with weights. In this paper we compare queries with prioritized conditions with queries with weighted conditions. Since PFCSPl systems are the theoretical background for PFSQL and similarly WFCSP are the theoretical background for weighted queries we elaborate these two systems. Queries with thresholds are another feature of PFSQL. When a threshold is attached to a condition only the tuples that satisfy the condition with a higher value then the threshold are displayed in the result. Through examples we compare these features of PFSQL.

I. INTRODUCTION

The representation of imprecise, uncertain or inconsistent information is not possible in relational databases, thus they require add-ons to handle these types of information. One possible add-on is to allow the attributes to have values that are fuzzy sets on the attribute domain, which results in fuzzy relational databases (FRDB).

SQL is the most influential commercially marketed database query language. It uses a combination of relational algebra and relational calculus constructs to retrieve desired data from a database. FSQL is SQL that can handle fuzzy attribute values. The main difference between SQL and FSQL is that SQL returns a subset of the database as the query result. On the other hand, FSQL returns a value in the unit interval for each datarow.

When attributes with fuzzy values appear in the query it is transformed into a query that can be handled by SQL and finally results obtained from the SQL query are then post processed in order to obtain the desired information.

Priority is implemented within Prioritized fuzzy constraint satisfaction problem (PFCSPl). PFCSPl is actually a fuzzy constraint satisfaction problem (FCSP) in which the notion of priority is introduced. Perhaps, the key factors in that implementation are priority t-norms. They are introduced in such a way that the smallest value, usually the value with the biggest priority, has the largest impact on the result given by a priority t-norm. It is introduced by an axiomatic framework. More details about PFCSPl are given later in the paper. The theoretical background for incorporating priority into FSQL is PFCSPl.

PFSQL allows conditions in the WHERE clause of the FSQL query to have a certain priority i.e. importance degree. Priorities are most often confused with weights. We compare weighted FSQL queries with PFSQL queries. Also each condition in the WHERE clause can have a threshold. If the threshold is not satisfied the data row is dropped from query result.

II. PFCSPl, PFCSP AND THRESHOLD

We will first define FCSP as the background of PFCSPl and WFCSP.

Definition 1. A fuzzy constraint satisfaction problem (FCSP) is defined as a 3-tuple \( (X, D, C^f) \) where:

1) \( X = \{ x_i | i = 1, 2, \ldots, n \} \) is a set of variables.
2) \( D = \{ d_i | i = 1, 2, \ldots, n \} \) is a finite set of domains. Each domain \( d_i \) is a finite set containing the possible values for the corresponding variable \( x_i \) in \( X \).
3) \( C^f \) is a set of fuzzy constraints. That is,

\[
C^f = \{ R^f | \mu_{R^f} : \prod_{x_j \in \text{var}(R^f)} d_j \rightarrow [0, 1] \}
\]

where \( i = \{1, 2, \ldots, n\} \).

PFCSPl systems are an extension of FCSP and they are introduced axiomatically. Instead of giving the formal definition of axioms we will briefly explain each of them.

The first axiom states that a zero value of the local satisfaction degree of the constraint with the maximum priority implies a zero value of the local satisfaction degree. The second axiom states that, in the case of equal priorities, the PFCSPl becomes a FCSP. The third axiom captures the notion of the priority. If one constraint has a larger priority then, the increase of the value on that constraint should result in a bigger increase of the global satisfaction degree than when the value with the smaller priority has the same increase. It captures the concept of priority in a linear sense. For example take two investments where one of them results in a bigger profit (larger priority) then it is expected that it is better to invest in a more profitable investment then in a less profitable one, if the profit increase is linear to the investment sum.

The fourth axiom is the monotonicity property, and finally the fifth is the upper boundary condition.
When the t-norm \( T_L \) is used together with the s-norm \( S_P \) we obtain the system that satisfies the given axiomatic framework. The global satisfaction degree in this system is calculated using the following formula:

\[
\alpha_P(v_X) = \bigoplus (\frac{\rho(R)}{\rho_{max}} \circ \mu_{R'}(v_{ear(R')}) | R' \in C')
\]

where \( \bigoplus (x, y) = T_L(x, y) \), and \( \circ (x, y) = S_P(1 - x, y) \). We will call this system \( T_L - S_P \).

Similarly, we obtain the min - max system if we take \( \bigoplus (x, y) = T_M(x, y) \), and \( \circ (x, y) = S_M(1 - x, y) \).

Now, describe how a PFCSP works. Priority of every constraint \( R_f \) is evaluated by function \( \rho : R_f \rightarrow [0, \infty) \). The larger the value of \( \rho \) is the larger the priority. After the normalization of the priority values which is done by dividing each priority by \( \rho_{max} = \max(\rho(R_f), R_f \in C_f) \) every priority gets a value in the unit interval. Standard implication aggregates priority of each constraint with its value. This is done in a way that the larger the priority, the more chance it has for the resulting value to stay the same as it was before aggregation. If the priority of constraint is small, then the aggregated value is closer to 1. This leads to greater values for constraints with the smaller priority. It makes sense, since when these aggregated values are again aggregated with a Scour-concave t-norm \( T \), the smaller values have more impact on the result due to properties of Scour-concave t-norms. We have given two concrete PFCSP systems, min - max and \( T_L - S_P \) that satisfy the previously given axioms. Now, we will describe how the global satisfaction degree of this system is calculated.

The function \( \rho \) represents the priority of each constraint. Operator \( \circ \) aggregates priority of each constraint with the value of that constraint. These are then aggregated by the operator \( \bigoplus \), which results in the satisfaction degree of an evaluation.

Priorities in PFCSP are most confused with the concept of weights. We can define a WFCSP weighted fuzzy constraint satisfaction problem, where for each constraint \( C_i \) we have an assigned weight \( w_i \). The global satisfaction degree for a valuation \( v_X \), \( \alpha_W(v_X) \) in WFCSP is calculated by a known formula:

\[
\alpha_W(v_X) = T(c_1 * w_1, c_2 * w_2, \ldots, c_n * w_n)
\]

where \( c_i = \mu_{C_i}(v_X) \) is the local satisfaction degree of a constraint \( C_i \) and \( T \) is a t-norm. In order to have an adequate comparison between WFCSP and PFCSP we take \( T = T_M \) and \( T = T_L \). When is used we get the global satisfaction degree \( \alpha_{W'}(v_X) \) and analogously when \( T_L \) is used we get \( \alpha_{W''}(v_X) \).

In FSQL we can assign a threshold (THOLD) to each constraint. We will now point out the difference between threshold and priority in order to avoid any confusion. If there is a THOLD quantifier attached to a condition, FSQOL automatically discards the data row which does not satisfy the condition with a given threshold. On the other hand, if the value of the PRIORITY exists, FSQL calculates the satisfaction degree for each data row regardless of its satisfaction degree as it will be shown in an example in the following section.

### III. FRDB AND FSQL

If we allow the attributes in classical RDB to have values that are fuzzy subsets of the attribute domain, the result will be fuzzy relational databases (FRDB).

Our idea is to have the most common fuzzy set type implementations and that the attribute values in FRDB are most often standard fuzzy sets, and only a small percentage of attribute values are generalized fuzzy sets specified by the user, though our model works with general fuzzy sets in every aspect of FRDB - storing, querying, etc. We introduce one more extension of the attribute value, the linguistic label. Linguistic labels are used to represent most common and widely used expressions of a natural language (such as tall people, small salary or mediocre result). Linguistic labels are in fact named fuzzy values from the domain. In order to use a linguistic label on some domain, first we must define this label. For instance, we can define the linguistic label tall man as a fuzzy quantity that has an increasing linear membership function from the point (185,0) to the point (200,1). Considering these extensions, we can define a domain of a fuzzy attribute as:

\[
D = D_C \cup F_D \cup L_L
\]

where \( D_C \) is a classical attribute domain, \( F_D \) is a set of all fuzzy subsets of the domain, and \( L_L \) is the set of linguistic labels. In our model we allow triangular fuzzy numbers and fuzzy quantities for \( F_D \).

The basic difference between SQL and FSQL is in the way the database processes records. In a classical relational database, queries are executed so that a tuple is either accepted in the result set if it fulfills conditions given in a query, or removed from the result set if it does not fulfill the conditions. In other words, every tuple is given a value true (1) or false (0). On the other hand, as the result set FSQL returns a fuzzy relation on the database. Every tuple considered in the query is given a value from the unit interval. This value is calculated using operators of fuzzy logic. The question is what elements of the classical SQL should be extended. Because variables can have both crisp and fuzzy values, it is necessary to allow comparison between different types of fuzzy values as well as between fuzzy and crisp values. In other words, FSQL has to be able to calculate expressions like \( \text{height} = \text{triangle}(180,11,8,\text{lin}) \), regardless of what value of height is in the database - fuzzy or crisp.

In classical SQL it is clear how to assign truth value to every elementary condition. With fuzzy attributes, situation is more complex. At first, we assign truth value from the unit interval to every elementary condition. Only way to do this is to give algorithm that calculates truth value for every possible combination of values in query and values in the database.
For instance, if a query contains condition that compares a fuzzy quantity value with a triangular fuzzy number in the database, we must have algorithm to calculate compatibility of the two fuzzy sets using a similarity relation. After the truth values from unit interval are assigned, they are aggregated using fuzzy logic. We use t-norm in case of operator AND, and its dual t-conorm in case of operator OR. For negation we use strict negation: \( N(x) = 1 - x \). In case of priority statements, we use PFSQL systems to calculate the result. We will now describe processes that allow PFSQL queries to be executed. The basic idea is to first transform PFSQL query in something that classical SQL interpreter understands. Namely, conditions with fuzzy attributes are removed from WHERE clause and those fuzzy attributes are moved up in the SELECT clause. In this way, conditions containing fuzzy constructs are eliminated, so that the database will return all the tuples ones that fulfil fuzzy conditions as well as the ones that do not. As a result of this transformation, we get a classical SQL query. Then, when this query is executed against the database, results are interpreted using fuzzy mechanisms. These mechanisms assign a value from unit interval to every tuple in the result set.

More precisely, processing the PFSQL query comprises of four phases: query syntax checking, loading the query into memory structure, transformation of the query, and fuzzy interpretation of the results returned by the database. First, the given query is checked for correct syntax using scanner and parser constructed for this task. If the query is correct, the result of syntax analysis done by the parser is a memory structure that represents this query. Next step is the transformation of this structure in already described fashion. We need to check whether an attribute is fuzzy or not. When the fuzzy attributes are identified, conditions in the WHERE clause that they appear in are removed, and the attributes are added to the SELECT clause. Removed conditions are put in another memory structure, because they will be used to interpret the result set. Result is a classical SQL query which can be directly executed against the database. After the query is executed, returned results are further processed. A measure of condition fulfillment is assigned to every tuple in the result set. This measure is a value from the unit interval which is defined by a similarity relation described in the following section. In this phase we use a memory structure with fuzzy conditions removed from WHERE clause to calculate measures using priority fuzzy logic.

IV. EXAMPLE

Now we will describe the scenario. Suppose we have to pick a soccer player and a basketball player. We evaluate candidates based on their Height, Speed and Stamina. Depending on the sort, each attribute will have a certain priority. We will suppose that for the soccer player Speed is the most important, Stamina is mildly important and Height is the least important. For the basketball player Height is the most important, Stamina is mildly important and Speed is the least important. The min-max and \( T_L - S_P \) systems will be used for evaluation.

The results for five athletes are given in the following tables. Table 1 represents evaluations for the soccer player. We assume that the priority of Speed constraint is 1, Stamina has priority 0.7 and finally Height has priority 0.2. Similarly, Table 2 represents evaluations for the basketball player where priority Speed constraint is 0.4, Stamina has priority 0.6 and finally Height has priority 1.

The example for the soccer player can be interpreted as the following PFSQL query.

```
SELECT *
FROM Athletes
WHERE (Height='tall') PR 0.2
AND (Stamina='excelent') PR 0.6
AND (Speed='fast') PR 1
```

<table>
<thead>
<tr>
<th>no.</th>
<th>Spd</th>
<th>Sta</th>
<th>Hei</th>
<th>( T_M )</th>
<th>( T_L )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.6</td>
<td>0.2</td>
<td>0.6</td>
<td>0.53</td>
</tr>
<tr>
<td>2</td>
<td>0.55</td>
<td>0.65</td>
<td>0.7</td>
<td>0.55</td>
<td>0.31</td>
</tr>
<tr>
<td>3</td>
<td>0.1</td>
<td>0.6</td>
<td>0.1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.8</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.59</td>
</tr>
<tr>
<td>5</td>
<td>0.9</td>
<td>0.6</td>
<td>0.3</td>
<td>0.6</td>
<td>0.59</td>
</tr>
</tbody>
</table>

Table 1. Satisfaction degree for the Soccer player.

Similarly, the example for the basketball player can be interpreted as the following PFSQL query.

```
SELECT *
FROM Athletes
WHERE (Height='tall') PR 1
AND (Stamina='excelent') PR 0.6
AND (Speed='fast') PR 0.4
```

<table>
<thead>
<tr>
<th>no.</th>
<th>Spd</th>
<th>Sta</th>
<th>Hei</th>
<th>( T_M )</th>
<th>( T_L )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.6</td>
<td>0.2</td>
<td>0.7</td>
<td>0.575</td>
</tr>
<tr>
<td>2</td>
<td>0.55</td>
<td>0.65</td>
<td>0.7</td>
<td>0.25</td>
<td>0.145</td>
</tr>
<tr>
<td>3</td>
<td>0.1</td>
<td>0.6</td>
<td>0.1</td>
<td>0</td>
<td>0.07</td>
</tr>
<tr>
<td>4</td>
<td>0.8</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.5</td>
</tr>
<tr>
<td>5</td>
<td>0.9</td>
<td>0.6</td>
<td>0.3</td>
<td>0.6</td>
<td>0.51</td>
</tr>
</tbody>
</table>

Table 2. Satisfaction degree for the Basketball player.

If we want to use a WPF CSP - \( T_M \) query for choosing the athletes, the query for the soccer player would be the following.

```
SELECT (MIN(Height*0.2,Stamina*0.6,Speed*1))
FROM Athletes
WHERE (Height='tall')
AND (Stamina='excelent')
AND (Speed='fast')
```

Similarly, for WPF CSP - \( T_L \) the query should have the following form:

```
SELECT (MAX(Height*0.2+Stamina*0.6+Speed*1,0))
FROM Athletes
WHERE (Height='tall')
AND (Stamina='excelent')
AND (Speed='fast')
```
The satisfaction degree for the queries are given in Table 3.

<table>
<thead>
<tr>
<th>no.</th>
<th>Spd</th>
<th>Sta</th>
<th>Hgt</th>
<th>TM</th>
<th>TL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.6</td>
<td>0.2</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.55</td>
<td>0.65</td>
<td>0.7</td>
<td>0.11</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0.1</td>
<td>0.6</td>
<td>1</td>
<td>0.002</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0.8</td>
<td>0.7</td>
<td>0.7</td>
<td>0.16</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0.9</td>
<td>0.6</td>
<td>0.3</td>
<td>0.18</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3. Weighted satisfaction degrees for the Soccer player.

We see that the results in Table 3 differ completely from Table 1. Moreover, if we would run the weighed query for the basketball player we would obtain similar results. This leads to a conclusion that weighted queries are completely different from priority queries.

Finally we can add a threshold to each of the constrains. Assume that we insist that a basketball player must be tall, has good stamina with the degree of 0.6 and fast with the degree of 0.4. This would translate into the following query.

```
SELECT *
FROM Athletes
WHERE (Height='tall') THRESHOLD 1
AND (Stamina='Excellent') THRESHOLD 0.6
AND (Speed='fast') THRESHOLD 0.4
```

It is obvious that none of the athletes would fulfill these requirements which is completely different that when we use priority or weighted queries.

V. Conclusion

In this paper we have presented a comparison of priority, weighted and threshold queries using an example of choosing athletes. We have shown that these three types of queries make different evaluations for the same data i.e. they are essentially different. The threshold is the most strict decision mechanism. Weighed and priority make are based on different assumptions and must not be confused.

Acknowledgment

The authors would like to acknowledge the support of the Serbian Ministry of Science and Environmental Protection, project Mathematical models of non-linearity, uncertainty and decision making, No. 144012 and project Abstract Methods and Applications in Computer Science No. 144017A, also the support of the Ministry of Science, Technology and Environmental Protection of Vojvodina.

References