Enhancing Flexible Querying Using Criterion Trees

Guy De Tré[1], Jozo Dujmović[2], Joachim Nielandt[1], and Antoon Bronselaer[1]

[1] Dept. of Telecommunications and Information Processing, Ghent University, Sint-Pietersnieuwstraat 41, B-9000 Ghent, Belgium
{Guy.DeTre,Joachim.Nielandt,Antoon.Bronselaer}@UGent.be

[2] Dept. of Computer Science, San Francisco State University, 1600 Holloway Ave, San Francisco, CA 94132, U.S.A.
jozo@sfsu.edu

Abstract. Traditional query languages like SQL and OQL use a so-called WHERE clause to extract only those database records that fulfil a specified condition. Conditions can be simple or be composed of conditions that are connected through logical operators. Flexible querying approaches, among others, generalized this concept by allowing more flexible user preferences as well in the specification of the simple conditions (through the use of fuzzy sets), as in the specification of the logical aggregation (through the use of weights). In this paper, we study and propose a new technique to further enhance the use of weights by working with so-called criterion trees. Next to better facilities for specifying flexible queries, criterion trees also allow for a more general aggregation approach. In the paper we illustrate and discuss how LSP basic aggregation operators can be used in criterion trees.

Keywords: Fuzzy querying, criterion trees, LSP, GCD.

1 Introduction

1.1 Background

Traditionally, WHERE-clauses have been used in query languages to extract those database records that fulfil a specified condition. This condition should then reflect the user's preferences with respect to the records that should be retrieved in the query result. Most traditional query languages like SQL [10] and OQL [2] only allow WHERE-conditions which can be expressed by Boolean expressions. Such Boolean expression can be composed of simple expressions that are connected by logical conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$) operators. Parentheses can be used to alter the sequence of evaluation.

Adequately translating the user's needs and preferences into a representative Boolean expression is often considered to be a difficult and challenging task. This is especially the case when user requirements are complex and expressed in natural language. Soft computing techniques help developing fuzzy approaches

H.L. Larsen et al. (Eds.): FQAS 2013, LNAI 8132, pp. 364–375, 2013.

for flexible querying that help to solve these difficulties. An overview of 'fuzzy' querying techniques can, among others, be found in [18].

In this paper, 'fuzzy' querying of a regular relational database is considered. Such a database consists of a collection of relations, represented by tables [3], comprising of attributes (columns) and tuples (rows). Each relation R is defined by a relation schema

$$R(A_1 : T_1, \ldots, A_n : T_n)$$

where the $A_i : T_i$'s are the attributes of R, each consisting of a name A_i and an associated data type T_i. This data type, among others, determines the domain dom_{T_i} consisting of the allowed values for the attribute. Each tuple

$$t_i(A_1 : v_1, \ldots, A_n : v_n)$$

with $v_i \in dom_{T_i}$, $1 \leq i \leq n$ represents a particular entity of the (real) world modelled by the given relation.

The essence of 'fuzzy' querying techniques is that they allow to express user preferences with respect to query conditions using linguistic terms which are modelled by fuzzy sets. The basic kind of preferences, considered in 'fuzzy' database querying, are those which are expressed *inside* an elementary query condition that is defined on a single attribute $A : T$. Hereby, fuzzy sets are used to express in a gradual way that some values of the domain dom_T are more desirable to the user than others. For example, if a user is looking for 'cheap houses', a fuzzy set with membership function μ_{cheap} on the domain of prices, as depicted in Fig. 1 can be used to reflect what the user understands by its linguistically expressed preference 'cheap house'.

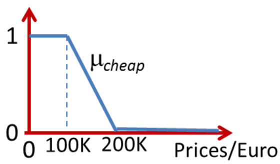

Fig. 1. The modelling of 'Cheap house prices'

During query processing, basically all relevant database tuples t are evaluated to determine whether they satisfy the user's preferences (to a certain extent) or not. Hereby, each elementary query criterion c_i, $i = 1, \ldots, m$ of the query is evaluated, resulting in an elementary matching degree $\gamma_{c_i}(t)$ which is usually modelled by a real number of the unit interval $[0, 1]$ (where $\gamma_{c_i}(t) = 1$ represents that the tuple t fully satisfies the criterion and $\gamma_{c_i}(t) = 0$ denotes no satisfaction). For example, evaluating the elementary criterion 'cheap price' for a tuple t with price attribute value $t[Price] = 110K$ results in an elementary satisfaction degree $\gamma_{cheap\_price}(t) = \mu_{cheap}(110K) = 0.9$, which expresses that a house of 110K satisfies the criterion 'cheap house' to an extent 0.9.

Next, the elementary degrees are aggregated to compute the overall matching degree $\gamma(t)$ of the tuple. In its simplest form, the aggregation of elementary matching degrees is determined by the fuzzy logical connectives conjunction, disjunction and negation which are respectively defined as follows:

$$\gamma_{c_1 \wedge c_2}(t) = i(\gamma_{c_1}(t), \gamma_{c_2}(t)) \tag{1}$$

$$\gamma_{c_1 \vee c_2}(t) = u(\gamma_{c_1}(t), \gamma_{c_2}(t)) \tag{2}$$

$$\gamma_{\neg c}(t) = 1 - \gamma_c(t) \tag{3}$$

where i and u resp. denote a t-norm and its corresponding t-conorm [12].

In a more complex approach, users are allowed to express their preferences related to the relative importance of the elementary conditions in a query, hereby indicating that the satisfaction of some query conditions is more desirable than the satisfaction of others. Such preferences are usually denoted by associating a relative weight w_i ($\in [0, 1]$) to each elementary criterion c_i, $i = 1, \ldots, m$ of the query. Hereby, as extreme cases, $w_i = 0$ models 'not important at all' (i.e., should be omitted), whereas $w_i = 1$ represents 'fully important'. Assume that the matching degree of a condition c_i with an importance weight w_i is denoted by $\gamma_{c_i^*}(t)$. In order to be meaningful, weights are assumed to satisfy the following requirements [4]:

- In order to have an appropriate scaling, at least one of the associated weights has to be 1, i.e., $\max_i w_i = 1$.
- If $w_i = 1$ and the associated elementary matching degree for c_i equals 0, i.e., $\gamma_{c_i}(t) = 0$, then the weight's impact should be 0, i.e., $\gamma_{c_i^*}(t) = 0$.
- If $w_i = 1$ and $\gamma_{c_i}(t) = 1$, then $\gamma_{c_i^*}(t) = 1$.
- If $w_i = 0$, then the weight's impact should be such as if c_i does not exist.

The impact of a weight can be computed by first matching the condition as if there is no weight and then second modifying the resulting matching degree in accordance with the weight. A modification function that strengthens the match of more important conditions and weakens the match of less important conditions is used for this purpose. From a conceptual point of view, a distinction has been made between static weights and dynamic weights.

Static weights are fixed, known in advance and can be directly derived from the formulation of the query. These weights are independent of the values of the tuple(s) on which the query criteria act and are not allowed to change during query processing. As described in [4], some of the most practical interpretations of static weights can be formalised in a universal scheme. Namely, let us assume that query condition c is a conjunction of weighted elementary query conditions c_i (for a disjunction a similar scheme has been offered). Then the matching degree $\gamma_{c_i^*}(t)$ of an elementary condition c_i with associated implicative importance weight w_i is computed by

$$\gamma_{c_i^*}(t) = (w_i \Rightarrow \gamma_{c_i}(t)) \tag{4}$$

where $\Rightarrow$ denotes a fuzzy implication connective. The overall matching degree of the whole query composed of the conjunction of conditions c_i is calculated using a standard t-norm operator. Implicative weighting schemes in the context of information retrieval and weights that have the maximum value 1 were mostly investigated by Larsen in [13,14].

The approach for static weights, has been refined to deal with a dynamic, variable importance $w_i \in [0, 1]$ depending on the matching degree of the associated elementary condition. Extreme (low or high) matching degrees could then for example result in an automatic adaptation of the weight.

A further, orthogonal distinction has been made between static weight assignments, where it is also known in advance with which condition a weight is associated (e.g., in a situation where the user explicitly states his/her preferences) and dynamic weight assignments, where the associations between weights and conditions depend on the actual attribute values of the record(s) on which the query conditions act (e.g., in a situation where most criteria have to be satisfied, but it is not important which ones). OWA operators [16] are an example of a technique with dynamic weight assignments.

Another aspect of 'fuzzy' querying concerns the aggregation of (partial) query conditions to be guided by a linguistic quantifier (see, e.g., [11,9]). In such approaches conditions of the following form are considered:

$$c = \Psi \text{ out of } \{c_1, \ldots, c_k\} \tag{5}$$

where Ψ is a linguistic (fuzzy) quantifier and c_i are elementary conditions to be aggregated. The overall matching degree $\gamma_c(t)$ of c can be computed in different ways. Commonly used techniques are for example based on liguistic quantifiers in the sense of Zadeh [17], OWA operators [16] and the Sugeno integral [1].

1.2 Problem Description

In many real-life situations users tend to group and structure their preferences when specifying selection criteria. Quite often, criteria are generalised or further specialised to obtain a better insight in what one is looking for. Such generalisations and specialisations then result in a hierarchically structured criteria specification, which will further on be called a *criterion tree.*

For example, for somebody who is searching for a house in a real estate database it is quite natural to require affordability (acceptable price and maintenance costs) and suitability (good comfort, good condition and a good location). Good comfort might be further specified by living comfort and basic facilities, where living comfort refers to at least two bathrooms, three bedrooms, garage etc. and basic facilities refer to gas, electricity, sewage, etc. Good condition might be specified by recent building and high quality building material. Finally, good location might be subdivided by accessibility, healthy environment, nearby facilities etc. The criterion tree corresponding to these user requirements is given in Fig. 2.

Query languages have no specific facilities for efficiently handling criterion trees. Indeed, criterion trees have to be translated to logical expressions, but for large criterion trees containing many criteria, this translation becomes difficult

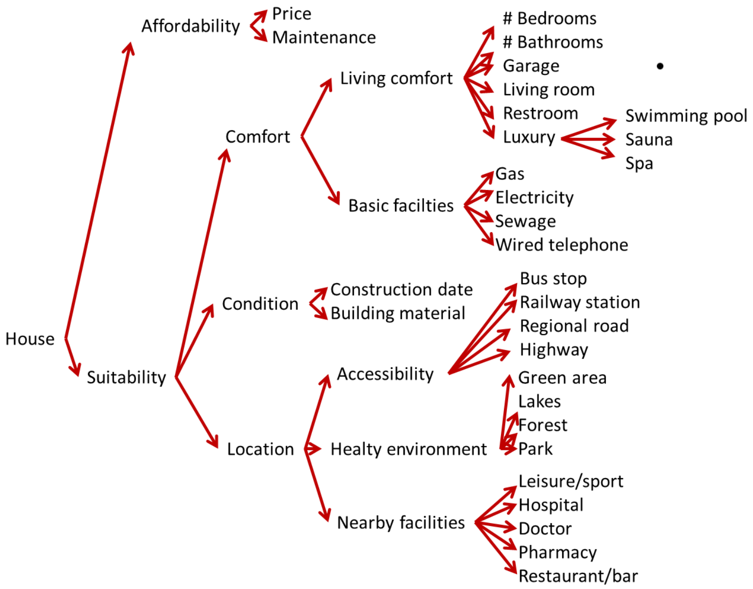

Fig. 2. Criterion tree for house selection

to interpret. Moreover, when working with weighted criteria, users often like to express preferences over subgroups of criteria. For example, a user might want to specify that the condition of a house is more important than its location. Such kinds of preferences require weight assignments to the internal nodes of a criterion tree. Translating such weights to weights for individual criteria is artificial and requires a significant effort as it becomes almost impossible to obtain a weight set which correctly reflects the preferences of the user. The latter even holds more generally: the more criteria we have to deal with, the more difficult it is to assign meaningful weights to the criteria [8]. Weight assignment is especially difficult for queries of high complexity which can contain hundreds of criteria and of which the weights should be easily adjustable. These kind of difficulties can be avoided by only considering manageable subsets of semantically related criteria to which weights are assigned in accordance with the user's preferences. Remark that humans use a similar approach when specifying complex requirements.

1.3 Objectives

In this paper we propose a novel flexible query specification and handling technique which is based on LSP (Logic Scoring of Preference) [5], a methodology which originates from decision support. The presented technique supports working with criterion trees and moreover allows for more flexibility in aggregating elementary degrees of satisfaction. The latter being obtained by providing the user with a selected number of generalized conjunction/disjunction (GCD) operators. Furthermore we illustrate that the use of criterion trees and GCD aggregation more adequately reflects the way how users reason while specifying their preferences related to their database search.

The remainder of the paper is organised as follows. In the next Section 2, criteria specification in criterion trees is discussed. The issues respectively dealt with are hierarchic query specification, weight specification and GCD selection. Next, the evaluation of a criterion tree is presented in Section 3. This evaluation is an important component of query processing and results in an associated overall satisfaction degree for each tuple that is relevant to the query result. In Section 4 we give an illustrative example based on house selection in order to justify the use of criterion trees with soft computing aggregation like GCD. Finally, in Section 5 the main contributions of the paper are summarised, conclusions are stated and some directions for future research are given.

2 Specification of Criterion Trees

A *criterion tree* is a hierarchical structure that is recursively defined as a collection of nodes starting at a root node. Each node can be seen as a container for information and can on its turn be connected with zero or more other nodes, called the child nodes of the node, which are one level lower in the tree hierarchy. A node that has a child is called the child's parent node. A node has at most one parent. A node that has no child nodes is called a leaf.

The leaf nodes of a criterion tree contain an elementary query condition c_A that is defined on a single database attribute $A : T$ as described in the introduction Section 1. This condition expresses the user's preferences related to the acceptable values for attribute $A : T$ in the answer set of the query.

All non-leaf nodes, i.e., the internal nodes, of a criterion tree contain a symbol representing an aggregation operator. Each child node n_i of a non-leaf node n has an associated weight w_i reflecting its relative importance within the subset of all child nodes of the non-leaf node. Hereby, for a non-leaf node with k child nodes it must hold that $\sum_{i=1}^{k} w_i = 1$. With this choice, we follow the semantics of the LSP methodology [5], which are different form those presented in [4].

Using Extended BNF (EBNF) notation [15], a criterion tree can be described by:

```
aggregator = "C" | "HPC" | "SPC" | "A" | "SPD" | "HPD" | "D"
criterion tree = elementary criterion | composed criterion
composed criterion = aggregator "(" criterion tree":"weight","
          criterion tree":"weight {"," criterion tree":"weight}")"
elementary criterion = attribute "IS {("min value"," suitability")"
      {",(" value"," suitability")" } ",("max value"," suitability")}"
```

where { } means 'repeat 0 or more times'. The values in elementary criterion must form a strictly increasing sequence.

The supported aggregators are denoted by 'C' (conjunction), 'HPC' (hard partial conjunction), 'SPC' (soft partial conjunction), 'A' (neutrality), 'SPD' (soft partial disjunction), 'HPD' (hard partial disjunction) and 'D' (disjunction).

This set is in fact a selection of seven special cases from the infinite range of generalized conjunction/disjunction (GCD) functions and can be easily extended when required.

The seven aggregators can be combined yielding nine combined aggregators as presented in Fig. 3.

Symbol	Aggregator
$\vee$	Disjunction (D)
$\bar{\bar{\nabla}}$	Hard partial disjunction (HPD)
$\underline{\nabla}$	Soft partial disjunction (SPD)
$\ominus$	Neutrality (A)
$\underline{\Delta}$	Soft partial conjunction (SPC)
$\bar{\Delta}$	Hard partial conjunction (HPC)
$\wedge$	Conjunction (C)
$\nabla \in \{\bar{\bar{\nabla}}, \underline{\nabla}\}$	Partial disjunction (PD)
$\Delta \in \{\bar{\Delta}, \underline{\Delta}\}$	Partial conjunction (PC)
$\bar{\bar{\bar{\nabla}}} \in \{\bar{\bar{\nabla}}, \vee\}$	Extended HPD (EHPD)
$\underline{\underline{\nabla}} \in \{\underline{\nabla}, \ominus\}$	Extended SPD (ESPD)
$\underline{\underline{\Delta}} \in \{\underline{\Delta}, \ominus\}$	Extended SPC (ESPC)
$\bar{\bar{\Delta}} \in \{\bar{\Delta}, \wedge\}$	Extended HPC (EHPC)
$\tilde{\nabla} \in \{\underline{\underline{\nabla}}, \bar{\bar{\bar{\nabla}}}\}$	Total disjunction (TD)
$\tilde{\Delta} \in \{\underline{\underline{\Delta}}, \bar{\bar{\Delta}}\}$	Total conjunction (TC)
$\Diamond \in \{\tilde{\nabla}, \tilde{\Delta}\}$	Generalized conjunction/ disjunction (GCD)

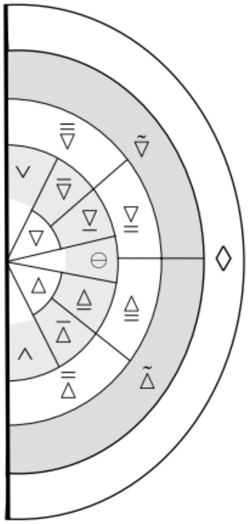

Fig. 3. Basic and combined simultaneity and replaceability operators

Two basic special cases of GCD are the partial conjunction (PC) and the partial disjunction (PD). Partial conjunction is a model of simultaneity, whereas partial disjunction is a model of replaceability. If we want to use GCD as an aggregator in a criterion tree, we have to select one of the supported aggregators based on the desired degree of simultaneity or replaceability.

Both 'C' and 'HPC' are models of high simultaneity and mandatory requirements. All inputs must be (partially) satisfied, and therefore they reflect mandatory requirements. If any input in an aggregated group of preferences is 0, the output is going to be 0. 'SPC' is also a model of simultaneity, but its (adjustable) level of simultaneity is lower than in the case of HPC. No input is mandatory. A single nonzero input is sufficient to produce a (small) nonzero output.

'D', 'HPD', and 'SPD' are models of replaceability symmetrical to 'C', 'HPC', and 'SPC'. 'D' and 'HPD' are models of high replaceability and sufficient requirements. If only one input is completely satisfied, that is sufficient to completely satisfy the whole group and the values of other inputs are insignificant. Each input can fully compensate (replace) all remaining inputs. 'SPD' is also a model of replaceability, but its (adjustable) level of replaceability is lower than in the case of HPD. No input is sufficient to completely satisfy the whole group, but any nonzero input is sufficient to produce a nonzero output.

The neutrality aggregator A (arithmetic mean) provides a perfect logic balance between simultaneity and replaceability. Thus, the logic interpretation of the arithmetic mean is that it represents a 50-50 mix of conjunctive and disjunctive properties; that is explicitly visible in the case of two inputs:

$$x_1 \theta x_2 = \frac{x_1 + x_2}{2} = \frac{(x_1 \wedge x_2) + (x_1 \vee x_2)}{2}. \tag{6}$$

For any number of inputs, all inputs are desired and each of them can partially compensate the insufficient quality of any other of them. No input is mandatory and no input is able to fully compensate the absence of all other inputs. In other words, the arithmetic mean simultaneously, with medium intensity, satisfies two contradictory requests: (1) to simultaneously have all good inputs, and (2) that each input has a moderate ability to replace any other input.

The arithmetic mean is located right in the middle of GCD aggregators but we cannot use it as a single best representative of all of them. The central location of the arithmetic mean is not sufficient to give credibility to additive scoring methods. Indeed, it is difficult to find an evaluation problem without mandatory requirements, or without the need to model various levels of simultaneity and/or replaceability. These features are ubiquitous and indispensable components of human evaluation reasoning. Unfortunately, these features are not supported by the arithmetic mean. Therefore, in the majority of evaluation problems the additive scoring represents a dangerous oversimplification because it is inconsistent with observable properties of human evaluation reasoning.

Once specified, criterion trees can be used in the specification of the WHERE-clause of a query. Their evaluation for a relevant database tuple t results in a criterion satisfaction specification, which can then be used in the further evaluation and processing of the query. In the next section, it is presented how criterion trees are evaluated.

3 Evaluation of Criterion Trees

Criterion trees are evaluated in a bottom-up way. This means that, when considering a relevant database tuple t, firstly, the elementary criteria c_i of the leaf nodes are evaluated. Any elementary criterion specification used in 'fuzzy' querying can be used. In its simplest form, c_i is specified by a fuzzy set F denoting the user's preferences related to an attribute $A : T$, as illustrated in Fig. 1. Criterion evaluation then boils down to determining the membership value of the actual value $t[A]$ of A for t, i.e.,

$$\gamma_{c_i}(t) = \mu_F(t[A]). \tag{7}$$

Next, all internal nodes (if any) are evaluated, bottom-up. An internal node n can be evaluated as soon as all its child nodes n_i, $i = 1, \ldots, k$ have been evaluated. For evaluation purposes, an implementation of GCD is required [7]. We can use the following implementation based on weighted power means (WPM):

$$M(x_1, \ldots, x_n; r) = \begin{cases} (\sum_{i=1}^{n} w_i x_i^r)^{1/r} & , \text{if } 0 < |r| < +\infty \\ \prod_{i=1}^{n} x_i^{w_i} & , \text{if } r = 0 \\ \min(x_1, \ldots, x_n) & , \text{if } r = -\infty \\ \max(x_1, \ldots, x_n) & , \text{if } r = +\infty \end{cases} \tag{8}$$

where $x_i \in [0, 1]$, $1 \leq i \leq n$ are the input values which in the context of flexible querying represent satisfaction degrees (hereby, 0 and 1 respectively denote 'not satisfied at all' and 'fully satisfied'); the normalised weights $0 < w_i \leq 1$, $1 \leq i \leq n$, $\sum_{i=1}^{n} w_i = 1$ specify the desired relative importance of the inputs and the computed exponent $r \in [-\infty, +\infty]$ determines the logic properties of the aggregator. Special cases of exponent values are: $+\infty$ corresponding to full disjunction 'D', $-\infty$ corresponding to full conjunction 'C', and 1 corresponding to weighted average 'A'. The other exponent values allow to model other aggregators, ranging continuously from full conjunction to full disjunction and can be computed from a desired value of orness (ω), and for this form of GCD function we can use the following numeric approximation [5]:

$$r(\omega) = \frac{0.25 + 1.89425x + 1.7044x^2 + 1.47532x^3 - 1.42532x^4}{\omega(1 - \omega)} \tag{9}$$

where

$$x = \omega - 1/2.$$

Suitable orness-values are the following: $\omega = 1/6$ for 'HPC', $\omega = 5/12$ for 'SPC', $\omega = 4/6$ for 'SPD' and $\omega = 5/6$ for 'HPD'.

Implication $w \Rightarrow x = \overline{w} \vee x = \overline{w \wedge \overline{x}}$ means that 'it is not acceptable that w is high and x is low', or 'important things must be satisfied'. The product wx used in Eq. (8) is also a form of implication because the effect is similar, i.e., again 'important things must be satisfied'.

Considering tuple t, the query satisfaction degree $\gamma_n(t)$ corresponding to n, computed using Eq. (8) with arguments $\gamma_{n_i}(t)$, $i = 1, \ldots, k$, w_i being the weight that has been associated with n_i, $i = 1, \ldots, k$, and r being the value that models the aggregator that is associated with n.

The overall satisfaction degree for tuple t using a criterion tree is obtained when the root node n_{root} of the tree is evaluated, i.e., this satisfaction degree yields

$$\gamma_{n_{root}}(t). \tag{10}$$

4 An Illustrative Example

As an example we reconsider the search for a house in a real estate database as presented in Fig. 2. Assuming that a user is looking for an affordable house with good comfort, good condition and a good location and wants to specify each of these subcriteria in more detail. Using GCD aggregators, the criterion tree given in Fig. 2 can be further detailed as shown in Fig. 4.

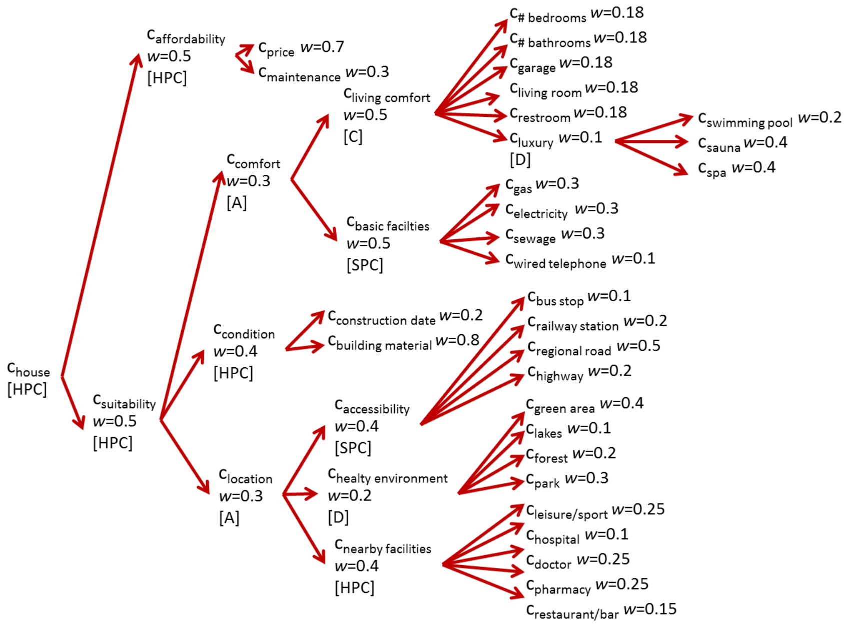

Fig. 4. Detailed criterion tree for house selection

Such a criterion tree can then be specified in an SQL statement that is used to query a regular relational database. Our approach is to use a predefined function *TREE* which takes a criterion tree as argument and computes the overall satisfaction degree of the tuples being processed by the query. This is illustrated with the following query which includes a regular join condition and a condition tree *c_house*.

```
SELECT id, address, price, TREE(c_house) AS satisfaction
FROM real_estates r, location l
WHERE (r.location_id=l.id) AND satisfaction>0.5
ORDER BY satisfaction
```

The tree condition tree *c_house* is further specified by

```
c_house=HPC(c_affordability:0.5,c_suitability:0.5)
```

where

```
c_affordability=HPC(c_price:0.7, c_maintenance:0.3)
c_suitability=HPC(c_comfort:0.3, c_condition:0.4, c_location:0.3)
```

Furthermore,

```
c_comfort=
A(C(c_#bedrooms:0.18, c_#bathrooms:0.18, c_garage:0.18,
    c_living_room:0.18, c_restroom:0.18, D(c_swimming_pool:0.2,
    sauna:0.4, c_spa:0.4):0.1):0.5,
  SPC(c_gas:0.3, c_electricity:0.3, c_sewage:0.3,
                                c_wired_telephone:0.1):0.5)
c_condition=HPC(c_construction_date:0.2, c_building_material:0.8)
c_location=
A(SPC(c_bus_stop:0.1, c_railway_station:0.2,
      c_regional_road:0.5, c_highway:0.2):0.4,
  D(c_green_area:0.4, c_lakes:0.1, c_forest:0.2, c_park:0.3):0.2,
  HPC(c_leisure/sport:0.25, c_hospital:0.1, c_doctor:0.25,
                   c_pharmacy:0.25, c_restaurant/bar:0.15):0.4)
```

The elementary criteria can generally be handled using soft computing techniques as presented in Fig. 1. For example, *c_price* can be specified by

```
r.price IS {(100,1), (200,0)}
```

where $\{(100, 1), (200, 0)\}$ is used to specify the fuzzy set that is depicted in Fig. 1. So the criterion *c_price* denotes that the price of the house should be compatible with the linguistic term 'cheap'. Compatibility is then determined by the membership grade of the stored *price* value of the house under consideration. For the other elementary conditions, similar preference specifications can be provided. Once all elementary conditions are evaluated, the criterion tree for the house under consideration can be evaluated as described in Section 3 and the resulting value will be returned by the function $TREE$ (in the example labelled as $satisfaction$). If preferred, a basic condition acting as a threshold condition on the satisfaction degrees can be added in the WHERE-clause of the query ($satisfaction > 0.5$ in the example). The satisfaction degrees can also be used to rank the tuples in the query result.

5 Conclusions and Future Work

In this paper, we proposed the concept of a criterion tree. Criterion trees offer flexible facilities for specifying complex query conditions. More specifically, they provide adjustable, generalized aggregators and weights denoting relative preferences among (sub)criteria can be assigned to all non-root criteria of the tree. These are requirements for adequately reflecting human decision making, to the best of our knowledge not being considered in flexible querying up to now. The proposed work is currently being implemented within the framework of the open source PostgreSQL object-relational database system.

In the presented work, only basic GCD aggregators have been considered. However, it is clear that not all criterion specifications necessary to reflect human reasoning can be modelled using the current approach. Therefore, the current work has to be extended with other aggregators, what will be subject to future work. One such extension concerns the handling of bipolarity and the ability to deal with mandatory, desired and optional conditions.

References

1. Bosc, P., Lietard, L., Pivert, O.: Sugeno fuzzy integral as a basis for the interpretation of flexible queries involving monotonic aggregates. Information Processing and Management 39(2), 287–306 (2003)
2. Cattell, R.G.G., Barry, D.K. (eds.): The Object Data Standard: ODMG 3.0. Morgan Kaufmann, San Francisco (2000)
3. Codd, E.F.: A Relational Model of Data for Large Shared Data Banks. Communications of the ACM 13(6), 377–387 (1970)
4. Dubois, D., Prade, H.: Using fuzzy sets in flexible querying: why and how? In: Andreasen, T., Christiansen, H., Larsen, H.L. (eds.) Flexible Query Answering Systems. Kluwer Academic Publishers, Dordrecht (1997)
5. Dujmović, J.J.: Preference Logic for System Evaluation. IEEE Transactions on Fuzzy Systems 15(6), 1082–1099 (2007)
6. Dujmović, J.J., Larsen, H.L.: Generalized conjunction/disjunction. Int. Journal of Approximate Reasoning 46, 423–446 (2007)
7. Dujmović, J.J.: Characteristic Forms of Generalized Conjunction/Disjunction. In: Proc. IEEE World Congress on Computational Intelligence, Hong Kong (2008)
8. Dujmović, J.J., De Tré, G.: Multicriteria Methods and Logic Aggregation in Suitability Maps. Int. Journal of Intelligent Systems 26(10), 971–1001 (2011)
9. Galindo, J., Medina, J.M., Cubero, J.C., Garcia, M.T.: Relaxing the Universal Quantifier of the Division in Fuzzy Relational Databases. Int. Journal of Intelligent Systems 16(6), 713–742 (2001)
10. ISO/IEC 9075-1:2011: Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework) (2011)
11. Kacprzyk, J.: Ziółkowski, A.: Database queries with fuzzy linguistic quantifiers. IEEE Transactions on Systems, Man and Cybernetics 16, 474–479 (1986)
12. Klement, E.P., Mesiar, R., Pap, E. (eds.): Triangular Norms. Kluwer Academic Publishers, Boston (2000)
13. Larsen, H.L.: Efficient Andness-directed Importance Weighted Averaging Operators. Int. Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 12(suppl.), 67–82 (2003)
14. Larsen, H.L.: Importance weighting and andness control in De Morgan dual power means and OWA operators. Fuzzy Sets and Systems 196(1), 17–32 (2012)
15. Wirth, N.: What Can We Do About the Unnecessary Diversity of Notation for Syntactic Definitions. Communications of the ACM 20(11), 822–823 (1977)
16. Yager, R.R., Kacprzyk, J.: The Ordered Weighted Averaging Operators: Theory and Applications. Kluwer Academic Publishers, Norwell (1997)
17. Zadeh, L.A.: A computational approach to fuzzy quantifiers in natural languages. Computational Mathematics Applications 9, 149–184 (1983)
18. Zadrozny, S., De Tré, G., De Caluwe, R., Kacprzyk, J.: An Overview of Fuzzy Approaches to Flexible Database Querying. In: Galindo, J. (ed.) Handbook of Research on Fuzzy Information Processing in Databases, pp. 34–54. IGI Global, Hershey (2008)