SYMPAS: A Database System for Managing Symbolic Preferences

Y. Harizi[1] A. Hadjali[2] H. Azzoune[1]
[1] LRIA/ USTHB, Alger, Algeria
[2] LIAS/ENSMA, Poitiers, France

LRIA/USTHB, Alger, Algeria, yharizi@usthb.dz
LIAS/ENSMA, Poitiers, France, allel.hadjali@ensma.fr
LRIA/USTHB, Alger, Algeria, azzoune@yahoo.fr

Résumé :
Dans cet article, on propose un système de base de données pour la gestion des préférences symboliques exprimées sous forme de déclarations conditionnelles. Ces préférences sont représentées au moyen de formules logiques possibilistes. Le processus de traitement de requêtes est discuté d'une manière explicite, en particulier, l'étape de sélection des top-k réponses. Pour montrer la faisabilité de notre système, une première série d'expérimentations a été menée et certaines mesures de performance sont effectuées et analysées.

Mots-clés :
Requêtes à préférences, Logique possibiliste, Top-k réponses.

Abstract:
In this paper, we propose a database system capable of managing symbolic preferences expressed under the form of conditional statements. Such preferences are translated into logic formulas in a possibilistic logic manner. Preference query processing is discussed in an explicit way, particularly, the step that consists in selecting the top-k answers. Some preliminary experiments are conducted to show the feasibility of our system.

Keywords:
Preference queries, Possibilistic logic, Top-k answers.

1 Introduction

In the last two decades, there has been a growing interest in preference queries in the database community [15][13]. Indeed, the use of preferences inside database systems has a number of potential advantages. First, it is desirable to offer more expressive query languages that are able to express user's requirements in a more faithful way. Second, the use of preferences in queries provides a basis for rank-ordering the retrieved items, which is especially valuable if a query is satisfied by a large set of items. Moreover, a classical query may also have an empty set of answers, while a relaxed (and thus less restrictive) version of the query can still be satisfied by several items in the database, at least to some degree.

Approaches to database preference queries may be classified into two categories according to their qualitative or quantitative nature [15]. In the latter, preferences are expressed quantitatively by a monotone scoring function, and the overall score is positively correlated with partial scores. Since the scoring function associates each tuple with a numerical score, tuple t1 is preferred to tuple t2 if the score of t1 is higher than the score of t2. Representatives of this family of approaches are top-k queries [7] and fuzzy-set-based approaches (e.g., [4]). In the qualitative approach, preferences are defined through binary preference relations. Since such relations can be defined in terms of scoring functions, the qualitative approach is more general than the quantitative one. Representatives of qualitative approaches are those relying on a dominance relationship, e.g. Pareto order, in particular Preference SQL [14], Skyline queries [2] and the approach presented in [8]. See also [5][3].

As users' preferences are more and more diverse and complex, the alternatives number described by means of a set of attributes are often very large. Facing this situation, preferences are not generally expressed in terms of explicit (pre)orders. It is then necessary to make their specification in a compact way. Compact

representation of preferences has raised a substantial interest in Artificial Intelligence [10][6] and, more recently, in the database field [13][5]. Graphical models (as CP-nets [5] and GAI [12]) and logical models (as possibilistic logic [11]) are particularly adapted to this kind of representation.

Let us note that conditional preference statements are often used for describing preferences in local, contextualized way [1]. To illustrate this approach let us consider the following example.

Example 1. Assume a schema instance of a relation Phone (make, model, color), where 'make', 'model' and 'color' take respectively their values in: {Apple(a), Samsung(s), Nokia (n), LG (l)...}, {iPhone 5, iPhone 4, iPhone 4s, iPhone 3g , iPhone 3gs, Galaxy s3, Galaxy s2, Galaxy y,..., N8, C7, C3} and {white(w), black(b), red (r), grey (y),....}. To buy a phone, a user can communicate her/his preferences as a set of pieces of information as follows:
Q_1 = *(i) "(s)he prefers Apple (a) phones to Samsung (s) phones,*
(ii) For Apple, (s)he prefers iPhone 5 (i_5) to iPhone 4 (i_4),
(iii) For Samsung, (s)he prefers Galaxy s3 (g_3) to Galaxy s2 (g_2) and Galaxy s2 (g_2) to Galaxy y (g_y),
(iv) s(he) prefers black (b) phones to white (w) phones".
The problem of interest is how to help this user to buy a phone by selecting and rank-ordering a set of phones from a database that better fit her/his preferences?

Statements of the above kind can be encoded with graphical or logical representations. For instance in [13], conditional preference statements are translated into classical logic formulas associated with symbolic priority levels, in a possibilistic logic manner. Only preferences on binary database attributes are considered in [13]. Starting from this work, we propose here a database system, called SYMPAS [2], that allows handling users' preferences expressed in a compact way under the form of conditional statements. In particular, we investigate the issue of selecting the top-k answers to the query at hand in the case where the target database may contain either binary or non-binary attributes.

The paper is organized as follows. Section 2 gives an overview of the SYMPAS architecture where the main modules of the system are illustrated and shortly described. Section 3 discusses the four steps (preferences translation, alternatives building, alternatives ranking, top-k answers selection) to query processing in SYMPAS. Section 4 addresses the empty answers issue. In Section 5, some preliminary experiments are provided to show the feasibility of the system SYMPAS and evaluate some performances. Section 6 concludes the paper and draws some perspectives for future work.

2 An overview of SYMPAS architecture

This section outlines the main components of the system SYMPAS (see Figure 1). As inputs, the system takes a set of conditional preferences statements and a number, k, of answers the user desires. As for outcomes, it provides the top-k databse tuples rank-ordered according to these preferences. Four components are designed:

- A module for *building the alternatives*. Alternatives are classes of tuples that can interest the user according to her/his preferences communicated to the system (see Table 1). Two methods have been implemented to build these alternatives: from the preferences communicated or from the content of the queried database.
- A module for *building the preference formulas*. It aims at representing the conditional preferences stated in possibilistic logic formulas.

[1] This means that preferences are (internal) context-dependent. Context here captures conditions that involve the data items stored in the database for which preferences are expressed.

[2] SYMbolic Preferences mAnagment System.

- A module for *rank-ordering the alternatives*. First, it computes the satisfaction level associated with each alternative by leveraging its violation of the possibilistic formulas. Then, a rank-ordering is established between the different alternatives.
- A module for *selecting the top-k answers*. For each alternative, an *SQL* query is built and then a set of tuples that satisfy it is selected. This evaluation process stops when k answers is obtained.

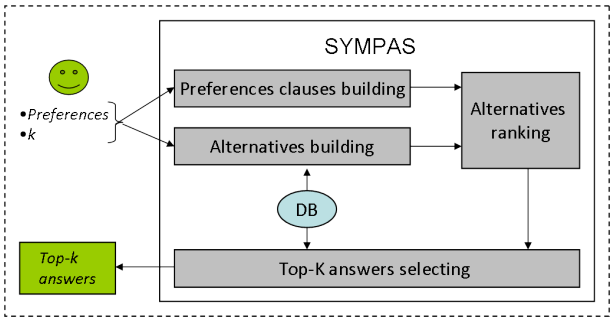

Figure 1: SYMPAS architecture

3 Conditional preference query processing

In the SYMPAS system, the conditional preferences considered are in the form: *"in context* c*,* a_1 *is preferred to* a_2*,* a_2 *is preferred to* a_3*, ...,* a_{n-1} *is preferred to* a_n*"*, where *{*a_1 *...* a_n*}* is a subset of values of a database attribute. In Example 1, the user prefers the models g_3 to g_2 and g_2 to g_y where 'g_3', 'g_2' and 'g_y' are a subset of values of the attribute 'model'. Note that a user is supposed to not be aware of all the values of an attribute to express her/his preferences.

The question of interest is how a set of answers can be selected and rank-ordered according to a set of preferences expressed in the above form? The idea is firstly to represent the preferences as possibilistic logic formulas using symbolic weights, in the spirit of the approach proposed in [13]. Secondly to identify the classes of tuples (alternatives) that interest the user and then to rank-order these alternatives according to their symbolic satisfaction levels w.r.t. the logic formulas. Finally to select from the target database the k best tuples that fit the possible alternatives. Below we provide details about each step for handling such conditional preferences.

3.1 Preference clauses building

As pointed out in [13], the possibilistic encoding of the conditional preferences of the form *"in context c, a is preferred to b"*, when $(b \equiv \neg a)$, is a possibilistic formula of the form: $(\neg c \vee a, 1-\alpha)$ [3] where the symbolic level $1-\alpha$ expresses a priority (rather than a certainly level). This encodes a constraint of the form $N(\neg c \vee a) \geq 1-\alpha$, here it is equivalent to a constraint on a conditional necessity measure $N(a|c) \geq 1-\alpha$. This is still equivalent to $\Pi(\neg a|c) \leq \alpha$, where Π is the dual possibility measure associated with N. It expresses that the possibility of not having a is upper bounded by α, i.e. $\neg a$ is all the more impossible as α is small. When a and b do not cover all the possible choices $(b \neq \neg a)$, the possibilistic formula: $(\neg c \vee a \vee b, 1-\beta)$, $1-\beta > 1-\alpha$, should be added.

More generally, the possibilistic encoding of the conditional preferences of the form "in context c, a_1 is preferred to a_2, a_2 is preferred to a_3, ..., a_{n-1} is preferred to a_n", with the assumption that a_1, ..., a_n do not necessarily cover all the possible choices, is equivalent to the following n possibilistic formulas:
$\{(\neg c \vee a_1 \vee ... \vee a_n, 1-\alpha_1), (\neg c \vee a_1 \vee ... \vee a_{n-1}, 1-\alpha_2), ... (\neg c \vee a_1, 1-\alpha_n)\}$, *with* $1-\alpha_1 > 1-\alpha_2 > ... > 1-\alpha_n$.
Note that when $a_1 \vee ... \vee a_n$ cover all the possible choices in the database, the first clause becomes a tautology and thus does not need to be written.

In algorithm 1, we show how the possibilistic clauses are computed from a user preference query. In our system, each preference $p(i)$ is represented by a quadruple $P(i) = \{A_c, A_p, V_{A_c}, V_{A_p}\}$, where A_c stands for the

[3] One can check that the formula $(\neg c \vee a \vee b, 1)$ is dropped since it is a tautology.

context attribute, V_{A_c} the value of the context attribute, A_p preference attribute and V_{A_p} the set of values of the preference attribute. For each preference $P(i)$, we compute $|P_i.V_{A_p}|$ [4] possibilistic clauses, each clause is represented by a quintuple $C(j) = \{A_c, V_{A_c}, A_p, V_{A_p}, s\}$ where s represents a symbolic priority weight. As can be seen, algorithm 1 returns both the set of possibilistic clauses encoding the preferences of the user and the induced partial order between the symbolic weights associated with these possibilistic clauses.

Algorithm 1 Clauses building

```
1: Symbol = {α, β, γ, δ...}
2: j ← 0 – the first clause
3: for each preference P_i, i: 1 to p do
4:    for t = |P_i.V_Ap| down to 1 do
5:       C_j.A_c ← P_i.A_c
6:       C_j.A_p ← P_i.A_p
7:       C_j.V_Ac ← P_i.V_Ac
8:       for k = 1 to t do
9:          C_j.V_Ap(k) ← P_i.V_Ap(k)
10:      end for
11:      C_j.s ← Symbol(j)
12:      O_ij = Symbol(j) – the partial order
13:      j ← j + 1 – to generate the next clause
14:   end for
15: end for
16: return {C_1, ..., C_j}
17: return O
```

Example 2 (Example 1 cont'd). One can check that algorithm 1 leads to the following encoding of the preferences involved in the query Q_1: $\{(a \vee s, 1-\alpha_1), (a, 1-\alpha_2), (\neg a \vee i_5 \vee i_4, 1-\beta_1), (\neg a \vee i_5, 1-\beta_2), (\neg s \vee g_3 \vee g_2 \vee g_y, 1-\gamma_1), (\neg s \vee g_3 \vee g_2, 1-\gamma_2), (\neg s \vee g_3, 1-\gamma_3), (b \vee w, 1-\delta_1), (b, 1-\delta_2), \}$, with the following partial order between weights $O = \{ \alpha_1 < \alpha_2, \beta_1 < \beta_2, \gamma_1 < \gamma_2 < \gamma_3, \delta_1 < \delta_2 \}$.

3.2 Alternatives building

Alternatives correspond to the different interpretations of the set of attributes stated in the preference query. They represent the classes of tuples that can interest the user. For instance, in the query Q_1, preferences are about three different attributes: 'make', 'model' and 'color' and ai_5b is an example of alternative. The alternatives are calculated from the preferences present in the user query. This means that we consider just the choices stated in the user preferences. To do this, our system takes as input the conditional preferences in the form of one or more independent trees. Possible alternatives are obtained by an in-depth scanning of the tree resulting from fusing all the independent trees (see further Figure 2).

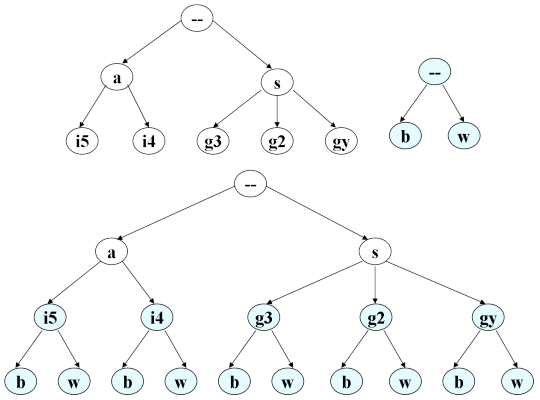

Figure 2: Preference representation

Example 3 (Example 1 cont'd). Let us consider the preference query Q_1. One can observe that preferences can be represented by two independent trees (as illustrated in Figure **??**). The first three preferences form the first tree and the fourth preference constitutes the second tree. To calculate the alternatives, we combine the two trees by considering the alternatives of the first tree as a context of the preferences of the second tree. To do this, the fourth preference (i.e., the user prefers black phones to white phones) can be replaced by the following preferences: (i) for iPhone 5 (resp. iPhone 4), (s)he prefers black phones to white phones ; (ii) for Galaxy s3 (resp. Galaxy s2, Galaxy y), (s)he prefers black phones to white phones.
Then, ten different types of phones are obtained: $T = \{ ai_5b, ai_4b, sg_3b, sg_2b, sg_yb, ai_5w, ai_4w, sg_3w, sg_2w, sg_yw \}$.

[4] This quantity stands for the cardinality of V_{A_p} related to the preference $P(i)$.

3.3 Alternatives ranking

To rank-order the possible alternatives, we use their satisfaction levels w.r.t. to the set of possibilistic formulas expressing the basic preferences of the user. Such satisfaction levels are calculated by leveraging the violation of these formulas by each alternative. In Table 1, we summarize all the satisfaction levels corresponding to the set of alternatives T. For instance, the vector of satisfaction of the alternative ai_4b is $(1, 1, 1, \beta_2, 1, 1, 1, 1, 1)$ which means that it satisfies all the formulas except the formula $(\neg a \vee i_5, 1 - \beta_2)$.

Alternatives	$(a \vee s, 1-\alpha_1)$	$(a, 1-\alpha_2)$	$(\neg a \vee i_5 \vee i_4, 1-\beta_1)$	$(\neg a \vee i_5, 1-\beta_2)$	$(\neg s \vee g_3 \vee g_2 \vee g_y, 1-\gamma_1)$	$(\neg s \vee g_3 \vee g_2, 1-\gamma_2)$	$(\neg s \vee g_3, 1-\gamma_3)$	$(b \vee w, 1-\delta_1)$	$(b, 1-\delta_2)$
ai_5b	1	1	1	1	1	1	1	1	1
ai_4b	1	1	1	β_2	1	1	1	1	1
sg_3b	1	α_2	1	1	1	1	1	1	1
sg_2b	1	α_2	1	1	1	1	γ_3	1	1
sg_yb	1	α_2	1	1	1	γ_2	γ_3	1	1
ai_5w	1	1	1	1	1	1	1	1	δ_2
ai_4w	1	1	1	β_2	1	1	1	1	δ_2
sg_3w	1	α_2	1	1	1	1	1	1	δ_2
sg_2w	1	α_2	1	1	1	1	γ_3	1	δ_2
sg_yw	1	α_2	1	1	1	γ_2	γ_3	1	δ_2

Table 1: Satisfaction levels of T

Alternative ranking is established using a technique called "extended Leximin" [13]: Let $\alpha = (\alpha_1, \ldots, \alpha_n)$ and $\beta = (\beta_1, \ldots, \beta_n)$ be two lists of weights attached, respectively, to two proofs of the same proposition, say q. Then α and β can be ordered using an "extended leximin" defined as follows: First, α and β must be increasingly reordered. Assume that the obtained reordered lists correspond to $(\lambda_1, \ldots, \lambda_n)$ and $(\delta_1, \ldots, \delta_n)$. Then the leximin ordering of the lists α and β writes: $\alpha \succ_{leximin} \beta$ iff $\lambda_1 > \delta_1$ or $\exists i$ such that $\forall j = 1 \cdots i; \lambda_j = \delta_j$ and $\lambda_{i+1} > \delta_{i+1}$.

Since the values of the weights (α_2, β_1, γ_1, $delta_1$, etc.) are unknown, no particular ordering is assumed between them, this technique leads to a partial order between possible alternatives.
Now, by applying this technique on T of example 3, we get the following pre-order:
$ai_5b >_{kc} \{ ai_4b, sg_3b, sg_2b, sg_yb, ai_5w, ai_4w, sg_3w, sg_2w, sg_yw \}$; $ai_4b >_{kc} ai_4w$; $ai_5w >_{kc} ai_4w$; $sg_3b >_{kc} sg_2b >_{kc} sg_yb$; $sg_3w >_{kc} sg_2w >_{kc} sg_yw$.

Some alternatives as sg_2b, sg_3w are incomparable because the user prefers: (i) the model g_3 to g_2 in case of *Samsung* and (ii) w.r.t color, black phone b to white phone w. But, if we add a pre-order between the symbolic weights by any consistent set of ordering constraints, possibly taking into account some priorities between the user's preferences, a supplementary pre-order between the set of alternatives can be obtained. Let us now consider available the priorities between the user's preferences. For instance, the priority order between the preferences is similar to their order when stated by the user. Then, the partial order between clauses is expressed with the following constraints $\{ \alpha_2 < \beta_1, \beta_2 < \gamma_1, \gamma_3 < \delta_1 \}$. Finally, we get the following order between the set of alternatives for the query Q_1: $ai_5b \succ_{lex} ai_5w \succ_{lex} ai_4b \succ_{lex} ai_4w \succ_{lex} sg_3b \succ_{lex} sg_3w \succ_{lex} sg_2b \succ_{lex} sg_2w \succ_{lex} sg_yb \succ_{lex} sg_yw$.

3.4 Top-k answers selection

Top-k answers are obtained by evaluating the rank-ordered set of alternatives over the queried database. To this end, we associate with each alternative an SQL query. It is worth noticing that in our system this evaluation process stops when a maximal number of answers, k, is retrieved. This means that the alternatives are not necessary all evaluated. See Algorithm 2 for the top-k answers selection procedure.

Example 4 (Example 1 cont'd). For the rank-ordered alternatives set $T_1 = \{ai_5b, ai_5w, ai_4b, ai_4w, sg_3b, sg_3w, sg_2b, sg_2w, sg_yb, sg_yw\}$, the corresponding SQL queries write:
For ai_5b: *"select * from* $Phone$ *where Make = 'Apple' and Model = 'iPhone 5' and Color =*

Algorithm 2 Top-k answers selecting

1: $T = \{t_1, t_2, ..., t_n\}$: the rank-ordered set of n alternatives
2: k: the maximum number of tuples to retrieve
3: $i \leftarrow 1$ – the first alternative
4: **while** $k > 0$ **do**
5: $s \leftarrow Select(t_i, K)$ – select s tuples correspond to the alternative t_i
6: $k \leftarrow k - s$;
7: $i \leftarrow i + 1$ – next alternative
8: **end while**

black",
For ai_5w: *"select * from* $Phone$ *where Make = 'Apple' and Model= 'iPhone 5' and Color = white"*,
And so on.

4 Empty answers case

One can observe that when the alternatives (T) is calculated from the user preferences, we have no ideas about the content of the queried database. Then, it may be happen that no tuples in the database correspond to the computed alternatives and then no tuples (partially) satisfy the (conditional) preferences. To overcome this problem, one way is to calculate the set of alternatives from the queried database. We consider all the choices about the attributes related to preferences stated, that may exist in the database. This calculation can be done by simply building an SQL query using the different attributes stated in the preferences. In the case of Example 1, to obtain the alternatives, one can use the following SQL query on the attributes 'make', 'model' and 'color':
Q_2 *="Select Make, Model, Color From* BD *Group by Make, Model, Color"*.
One can, for instance, get alternatives that exist in the database and not stated in user's preferences: black iPhone 3 (ai_3b), white iPhone 3 (ai_3w), red Samsung g_3 (sg_3r), black Samsung g_4 (sg_4b), white Samsung g_4 (sg_4w), black Nokia c_7 (nc_7b), white Nokia c_7 (nc_7w).
Such alternatives partially satisfy the user preferences and their satisfaction levels w.r.t the logic formulas are: (1,α_2,1,1,1,1,1,δ_1,δ_2), (1,1,β_1,β_2,1,1,1,1,1), (α_1,α_2,1,1,1,1,1,1,1), (1,α_2,1,1,γ_1,γ_2,γ_3,1,1), (α_1,α_2,1,1,1,1,1,1,δ_2), (1,α_2,1,1,γ_1,γ_2,γ_3,1, δ_2), (1,1,β_1,β_2,1,1,1,1,δ_2)
By applying the leximin order, we get the following order: $ai_3b \succ_{lex} ai_3w \succ_{lex} sg_3r \succ_{lex} sg_4b \succ_{lex} sg_4w \succ_{lex} nc_7b \succ_{lex} nc_7w$.

5 Experimentation study

Let us first precise that the system SYMPAS is implemented in Java and the experiments are run on Intel Core i3 CPU 2.3GHz with 4.0GB RAM under Windows 8. We have used five databases of different sizes $2ko, 20ko, 50ko, 100ko, 500ko, 1000ko$, on a relation Phone with the schema, *Phone (Make, Model, Color, Memory, Operator, Price)*.

5.1 First experiment

This experiment aims at measuring the execution time of a preference query [5] to select the top-k answers over databases of different sizes figure 3. As can be seen, the execution time to select 30 to 200 tuples from the databases of sizes $20ko, 50ko, 100ko$ is lower then the one of size $2ko$ The execution time does not (significantly) change from $k = 200$ tuples when queried the database of size $2ko$ (This behavior is also observed in the cases of sizes $20ko$ and $50ko$).
To provide some explanation about the above results, let us analyze the alternatives number used (i.e., the number of SQL queries sent to the databases) for satisfying the user preference query in each case. This analysis is illustrated in Figure 4.
From Figures 3 and 4, we have:
$T_5^1(100ko) \simeq T_5^1(50ko) \simeq T_5^1(20ko) \simeq T_5^1(2ko)$.
$T_{50}^6(2ko) > T_{50}^1(100ko) \simeq T_{50}^1(50ko) \simeq T_{50}^1(20ko)$.

As can be seen, there is some relationship between the executed alternatives number

[5] $T_k^n(|DB|)$: The execution time to select the top k tuples from the database of size $|DB|$, n is the number of alternatives used or the number of SQL queries sent to the database to select the k tuples.

used for satisfy the preference query and the execution time of the query. The higher the number of alternatives, the higher execution time is. This is due to the fact that in our system the set of alternatives are built from the preference query, and these alternatives are used for retrieving the top-k tuples desired as answers to the query at hand. This is why we have: $T^{10}_{100}(2ko) \simeq T^{10}_{200}(2ko) \simeq ... \simeq T^{10}_{2000}(2ko)$.

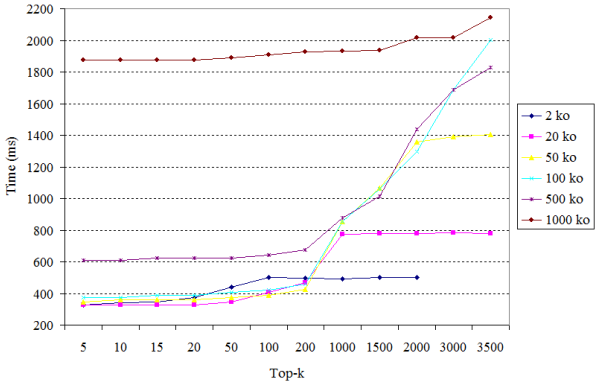

Figure 3: Different execution times

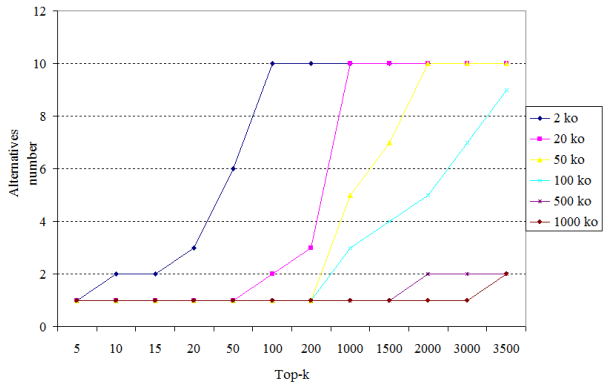

Figure 4: Executed alternatives number

5.2 Second experiment

The aim of this experiment is to compare the execution time of preference queries w.r.t. the two methods implemented in SYMPAS for building alternatives (from the user preference query and from the database content).
We use the following scenario: (i) we consider the preference query (Q_1) used in the first experiment, the alternatives calculated from the conditional preferences are $\{ai_5b,\ ai_5w, ai_4b,\ ai_4w,\ sg_3b,\ sg_3w,\ sg_2b,\ sg_2w,\ sg_yb,\ sg_yw\}$; (ii) we also consider a queried database of $50ko$ tuples. This database may contain up to 200 different types of phones corresponding to the attribute values stated in the preference query.
Figure 5 shows the execution time evolution of the query (Q_1, k), with $k \in \{5, 10, 15, 20, 50, 100\}$. One can observe that the execution time when we calculate the alternatives from preference query is lower than the one obtained by calculating the alternatives from the content of the database. However, the second approach could be useful in the case where the first one results in empty answers.

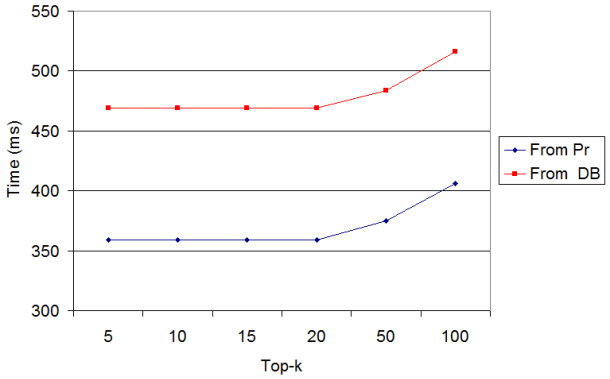

Figure 5: Second experiment

6 Conclusion

In this paper, the first foundations of a database system capable of handling user preferences expressed under the form of conditional statements, are discussed where possibilistic logic plays a key role for representing such conditional preferences. The top-k answers selection to a user query is investigated as well. Some preliminary experiments are conducted to show the feasibility of our proposal and to make some performance measures related to execution time. We plan to perform thorough experiments, in the one hand, to study the effectiveness and efficiency of the proposed system on large real databases and, on the other hand, to compare the system to other approaches. Symbolic priority expressed in an imprecise way [1]

will be also considered. A third line for future research is to investigate the issue of revising preferences in SYMPAS system in the spirit of [9].

References

[1] S. Benferhat, J. Hué, S. Lagrue, and J. Rossik. Interval-based possibilistic logic. In *Proc. IJCAI*, pages 750–755, 2011.

[2] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. of ICDE*, pages 421–430, 2001.

[3] P. Bosc, A. Hadjali, and O. Pivert. An approach to competitive conditional preferences for database flexible querying. *International Journal of Intelligent Systems*, 26(3):206–227, 2011.

[4] P. Bosc and O. Pivert. Sqlf: a relational database language for fuzzy querying. *IEEE Trans. on Fuzzy Sys.*, 3:1–17, 1995.

[5] R-I. Brafman and C. Domshlak. Database preference queries revisited. In *Technical Report TR2004-1934, Cornell University, Comput. and Info. Science*, 2004.

[6] R-I. Brafman and C. Domshlak. Preference handling - an introductory tutorial. In *Artificial Intelligence Magazine*, volume 30, pages 58–86, 2009.

[7] N. Bruno, S. Chaudhuri, and L. Gravano. Top-k selection queries over relational databases: mapping strategies and performance evaluation. *ACM Trans. on Database Sys.*, 27:153–187, 2002.

[8] J. Chomicki. Preference formulas in relational queries. *ACM Transactions on Database Systems*, 28(4):1–40, 2003.

[9] J. Chomicki. Database querying under changing preferences. In *Annals of Mathematics and Artificial Intelligence*, pages 79–109, 2007.

[10] C. Domshlak, E. Hüllermeier, S. Kaci, and H. Prade. Preferences in artificial intelligence: An overview. *Artificial Intelligence Journal (In Special Issue on Representing, Learning, and Processing Preferences: Theoretical and Practical Challenges)*, 175:7–8, 2011.

[11] D. Dubois and H. Prade. Possibilistic logic: a retrospective and prospective view. *Fuzzy Sets and Sys.*, 144:3–23, 2004.

[12] C. Gonzales and Perny P. Gai networks for utility elicitation. In *Proc. of the 9th Inter. Conf. on Principles of Knowledge Rep. and Reas.*, pages 224–234, 2004.

[13] A. Hadjali, S. Kaci, and H. Prade. Database preference queries - a possibilistic logic approach with symbolic priorities. *Annals of Mathematics and AI*, 63:357–383, 2011.

[14] W. Kiessling and G. Kostler. Preference sql— design, implementation, experiences. In *Proc. of VLDB*, pages 999–1001, 2002.

[15] K. Stefanidis, G. Koutrika, and E. Pitoura. A survey on representation, composition and application of preferences in database systems. *ACM Trans. on Database Sys.*, 36, 2011.