Available online at www.sciencedirect.com
ScienceDirect
Fuzzy Sets and Systems 160 (2009) 251–268
FUZZY sets and systems
www.elsevier.com/locate/fss

Fuzzy matchmaking in e-marketplaces of peer entities using Datalog

Azzurra Ragone[a,*], Umberto Straccia[b], Tommaso Di Noia[a], Eugenio Di Sciascio[a], Francesco M. Donini[c]

[a] *Electrical and Electronics Engineering Department, SisinfLab, Politecnico di Bari, via E. Orabona, 4, Bari, Italy*
[b] *ISTI - CNR, Pisa, Italy*
[c] *Università della Tuscia, Viterbo, Italy*

Received 17 July 2007; received in revised form 1 July 2008; accepted 1 July 2008
Available online 11 July 2008

Abstract

We present an approach to matchmaking in electronic marketplaces of peer entities, which mixes in a formal and principled way Datalog, fuzzy sets and utility theory, in order to determine the most promising matches between prospective counterparts. The use of Datalog ensures the scalability of our approach to large marketplaces, while fuzzy logic provides a neat connection with logical specifications and allows to model soft constraints and *how well* they could be satisfied by an agreement. Noteworthy is that our approach takes into account in the peer-to-peer matchmaking also preferences of each counterpart and their utilities. This allows to rule out of the match list those counteroffers that, although seemingly appealing for the buyer, would probably lead to failure due to contrasting preferences of the seller, and paves the way to the actual negotiation stage.

Keywords: Matchmaking; Electronic markets; Fuzzy logic programming; Databases; Top-*k* retrieval

1. Introduction

We consider a peer-to-peer (P2P) e-marketplace basically as a system where peer users—buyers and sellers—can submit their advertisements, browse through available ads, and—hopefully—be assisted in finding the best available counterparts to meet their needs and initiate a transaction. Obviously, such e-marketplaces, while sharing common features with well-known business-to-business and business-to-consumer systems, have deep differences w.r.t. marketplaces of commodities and undifferentiated goods. In our framework usually both offers/requests refer to goods/services that cannot be simply described in a machine understandable way without the help of some knowledge representation language, and price is not the single feature to look and/or negotiate on. For example, descriptions could have logical implications, e.g. *If a car has leather seats then it is also provided with air conditioning*, or bundles e.g. *Sports car with optional package including both GPS system and alarm system*, and some kind of logical theory, able to let

* Corresponding author. Tel.: +39 805963515; fax: +39 805963410.
E-mail addresses: a.ragone@poliba.it (A. Ragone), straccia@isti.cnr.it (U. Straccia), t.dinoia@poliba.it (T. Di Noia), disciascio@poliba.it (E. Di Sciascio), donini@unitus.it (F.M. Donini).

0165-0114/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.fss.2008.07.002

users express their needs/offers, could surely help. Also, when descriptions refer to complex needs, we should take into account preferences, distinguishing them from hard—mandatory—constraints, e.g. *I would like a black station wagon*, *preferably with GPS system*. The possibility to handle some of the abovementioned issues in some electronic facility may help not only in the discovery/matchmaking stage of a transaction process, thus selecting most promising counterparts to initiate a negotiation, but also in the actual negotiation stage.

Roughly speaking, matchmaking is the process of finding "good" counterparts for a given entry in the marketplace. In our setting an entry amounts to the request of a candidate buyer, and the counterpart is a seller, together with an offer of that seller. Of course, the evaluation of how "good" a counterpart constitutes most of the effectiveness of a matchmaking system. Several presently available commercial sites force the buyer to enter her request following a predefined classification that may be completely unsuitable for the characteristics the buyer might have in mind—e.g. eBay [12], Sunday Times [37] require to enter a brand first, then a model of that brand, etc. while a buyer may be interested in a general kind of product available from several brands, with some limitations on price. In this respect, one may say that they provide no matchmaking assistance: the matchmaker is the buyer herself. The escape of a textual search engine (as in eBay) does not help a lot, since the result is a (sometimes very long and tedious) list to browse. Other matchmaking sites provide a rank of counterparts based on some numerical computation on features—e.g. Yahoo [44]—but no explanation of the ranking is possible, obscuring the motivations of suggestion, and, hence, also possible refinements of a request.

To assist buyers and sellers in marketplaces, several research proposals on matchmaking systems were issued. They either try to compute a score of possible counterparts, based on textual information [42], or to compare the logical representations of supply and demand [40,27], or combine both scores and logic in some way [20,39,7,22]. Our proposal falls in this last category, mixing in a formal way Datalog, fuzzy sets and utility theory. While the above logic-based proposals do not tackle the scalability problem, our resort on Datalog ensures the scalability of our approach to large marketplaces. On the other hand, the resort on fuzzy logic ensures a neat connection with the logical specification, while allowing the system to give an explanation of suggestions in terms of how well the preferences could be satisfied if an agreement should be sought.

But the most prominent feature that distinguishes our proposal is the fact that the matchmaker takes into account the preferences of *each* counterpart: in fact, each score is the maximum value of the *product* of the weighted utility of the buyer u_β times the weighted utility of the seller u_σ. This product is maximized over all possible matches between the buyer and that seller. In this way, all those matches that seem appealing for the buyer but do not fit the seller's preferences are discarded when the match list is presented to the buyer.

Moreover, since the score corresponds also to a prospective "best-tradeoff" match between parties, the candidate match itself can be used as a pre-negotiation phase, to be integrated with other information and preferences that could not have been mentioned in the initial statements—e.g. a secondary time-cost tradeoff in shipping, or restrictions on the method of payment.

The remaining of the paper is as follows: to set the stage, Section 2 introduces basics of languages and technologies we adopt. In Section 3 requirements for the matchmaking process, including preferences, utility functions and reservation value, are illustrated. Then fuzzy soft constraints are presented in Section 4, followed by the description of fuzzy matchmaking process in Top-k-Datalog in Section 5. Next, in Section 6 rules for item classification in P2P marketplaces are outlined and an illustrative example is presented. Section 7 extends our framework to a general marketplace; an analysis of relevant related work and conclusions close the paper.

2. Basic technologies

We use an extension of *Datalog* (cf. [1,5,41]) as our representation and query tool. Datalog is a very powerful, well-studied declarative language based on Horn clause logic. Datalog adapts the paradigm of logic programming [21] to the database setting. We extend it by allowing soft constraint predicates to appear in rules and queries (we call the language Top-k-Datalog). The proposal here extends the work [35] in which soft constraint predicates, i.e. fuzzy predicates, may appear in queries only and is conceptually equivalent to [23,36]. Basically, we allow vague/fuzzy predicates to occur in rule bodies, which have the effect that each tuple in the answer set of a query has a score in [0, 1]. Top-k-Datalog addresses the problem to compute the top-k answers in case the set of facts is huge, without evaluating all the tuples' score. As matching a buyer's request with a seller's offer is a matter of degree, we will use Top-k-Datalog to find the top-k matchings. An initial prototype has been developed on top of the Prolog system XSB [43].

To make the paper self-contained, we first present Datalog and then its extension. The experienced reader may just skip the next subsection.

2.1. Datalog

Datalog rule: A Datalog *rule* is a Horn clause of the form

$$A \leftarrow A_1 \wedge \cdots \wedge A_n,$$

where A is the *head* of the rule, and $A_1 \wedge \cdots \wedge A_n$ is the *body* of the rule. A and all A_i are atoms. An *atom* is of the form $p(t_1, \ldots, t_n)$, where p is an n-ary predicate symbol and all t_j are terms. A *term* is either a *variable* or a *constant*. Each variable in the head of a rule must also occur in the body of the rule.

For instance,

$$\mathit{UncomfortableCar}(x) \leftarrow \mathit{Car}(x) \wedge \mathit{LeatherSeats}(x) \wedge \mathit{NoAirConditioning}(x)$$

is a rule stating that a car with leather seats and no air conditioning is an uncomfortable car.

We also provide a set of built-in predicates, such as $+, *, -, /, \geqslant, \leqslant, =$, with (obvious) fixed interpretation, which may appear in a rule body. Each variable occurring in a built-in predicate must occur at least once in an atom of the rule body.

Datalog fact: A *fact* is a Datalog rule with empty body (here we omit the $\leftarrow$ symbol). We further require that no variable occurs in facts (the fact is ground). Facts are usually stored in relational tables of an underlying database. For instance,

$$\mathit{CarTable}(544, \mathit{FiatPunto}, 2004, 15,000)$$

is a fact stating that item 544 is a Fiat Punto, built in 2004 and having 15,000 km.

Extensional/intentional database: An *extensional database* (EDB) is a finite set of Datalog facts, while an *intentional database* (IDB) is a finite set of Datalog rules without facts.

Datalog program: A Datalog program $\mathcal{P} = \langle \mathcal{P}_\mathrm{I}, \mathcal{P}_\mathrm{E} \rangle$ is given by an EDB $\mathcal{P}_\mathrm{E}$ and an IDB $\mathcal{P}_\mathrm{I}$ in which the predicates occurring in the EDB do not occur in the head of rules of the IDB. Essentially, we do not allow that the fact predicates occurring in $\mathcal{P}_\mathrm{E}$ can be redefined by $\mathcal{P}_\mathrm{I}$.

For instance,

$$\mathit{Sedan}(x) \leftarrow \mathit{CarTable}(x, y, z, t) \wedge (y = \mathit{FiatPunto})$$

is a rule stating that if an item in EDB is a Fiat Punto, then it is a Sedan.

Herbrand universe: From the semantics point of view [21], the *Herbrand universe* $H_\mathcal{P}$ of $\mathcal{P}$ is the set of constants appearing in $\mathcal{P}$. If there is no constant symbol in $\mathcal{P}$ then consider $H_\mathcal{P} = \{a\}$, where a is an arbitrary chosen constant.

Herbrand base: The *Herbrand base* $B_\mathcal{P}$ of $\mathcal{P}$ is the set of ground instantiations of atoms appearing in $\mathcal{P}$ (ground instantiations are obtained by replacing all variable symbols with constants of the Herbrand universe).

Herbrand interpretation: A *Herbrand interpretation* of $\mathcal{P}$ is any subset $I \subseteq B_\mathcal{P}$ of its Herbrand base. Intuitively, the atoms in I are true, and all others are false. That is, I satisfies a ground atom A iff $A \in I$.

Model: Let $\mathcal{P}^*$ be the set of ground rule instantiations obtained from $\mathcal{P}$. An interpretation I *satisfies* (is a *model* of) $\mathcal{P}$ iff I *satisfies* (is a *model* of) $\mathcal{P}^*$. I *satisfies* (is a *model* of) $\mathcal{P}^*$ iff I *satisfies* (is a *model* of) all rules in $\mathcal{P}^*$. I *satisfies* (is a *model* of) a rule $A \leftarrow A_1 \wedge \cdots \wedge A_n$ occurring in $\mathcal{P}^*$ iff $A \in I$ whenever $\{A_1, \ldots, A_n\} \subseteq I$.

Minimal model/immediate consequence operator: It is well known that each Datalog program has a unique *minimal* model $M_\mathcal{P}$, which coincides with the intersection of all models of $\mathcal{P}$. $M_\mathcal{P}$ is also the least fixed-point of the immediate consequence operator $T_\mathcal{P}: 2^{\mathcal{B}_\mathcal{P}} \rightarrow 2^{\mathcal{B}_\mathcal{P}}$ defined as

$$T_\mathcal{P}(I) = \{A \in B_\mathcal{P} | \mathcal{P}^* \text{ contains a rule } A \leftarrow A_1 \wedge \cdots \wedge A_n \text{ such that } \{A_1, \ldots, A_n\} \subseteq I \text{ holds}\}.$$

Intuitively, it yields all atoms that can be derived by a single application of some rule in $\mathcal{P}$ given the atoms in I. Since $T_\mathcal{P}$ is monotone, by the Knaster–Tarski Theorem it has a least fixed-point, denoted by $T_\mathcal{P}^\infty$; since, moreover, $T_\mathcal{P}$ is also

continuous, by Kleene's Theorem $T_{\mathcal{P}}^{\infty}$ is the limit of the sequence

$$T_{\mathcal{P}}^{0} = \emptyset,$$
$$T_{\mathcal{P}}^{i+1} = T_{\mathcal{P}}(T_{\mathcal{P}}^{i}), \quad i \geqslant 0.$$

Then $M_{\mathcal{P}} = T_{\mathcal{P}}^{\infty}$.

Entailment: We further say that $\mathcal{P}$ *entails* a ground atom A (denoted $\mathcal{P} \models A$) iff $A \in M_{\mathcal{P}}$.

Given a Datalog program $\mathcal{P} = \langle \mathcal{P}_{\mathrm{I}}, \mathcal{P}_{\mathrm{E}} \rangle$, a Datalog *query predicate* q is a designated n-ary predicate symbol appearing in the head of a rule in $\mathcal{P}_{\mathrm{I}}$.

For instance,

$$q(x, p) \leftarrow Car(x) \wedge Price(x, p) \wedge (p \leqslant 15,000)$$

is a query asking for cars whose price is less than or equal to 15,000.

Answer set: The *answer set* of q with respect to $\mathcal{P}$ is the set of tuples $\mathbf{c}$, such that $q(\mathbf{c})$ is entailed by $\mathcal{P}$, i.e.

$$ans(\mathcal{P}, q) = \{\mathbf{c} | \mathcal{P} \models q(\mathbf{c})\}.$$

2.2. Top-k Datalog

Top-k-Datalog is like Datalog except that we additionally allow fuzzy predicates to occur in Datalog rule bodies.

Top-k-Datalog rule: Specifically, let q be an $n+1$-ary predicate. A Top-k-Datalog *rule* is of the form

$$q(\mathbf{x}, s) \leftarrow \exists \mathbf{y} body(\mathbf{x}, \mathbf{y}) \wedge s = f(p_1(\mathbf{z}_1), \ldots, p_m(\mathbf{z}_m)),$$

where

(1) $\mathbf{x}$ are the *n distinguished variables*;
(2) s is the *score variable*, taking values in $[0, 1]$, and q is functional on $\mathbf{x}$;
(3) $\mathbf{y}$ are the so-called *non-distinguished variables* and are distinct from the variables in $\mathbf{x}$;
(4) $body(\mathbf{x}, \mathbf{y})$ is a conjunction of Datalog atoms;
(5) $\mathbf{z_i}$ are tuples of constants or variables in $\mathbf{x}$ or $\mathbf{y}$;
(6) p_i is an n_i-ary *fuzzy predicate* assigning to each n_i-ary tuple $\mathbf{c}_i$ as *score* $p_i(\mathbf{c}_i) \in [0, 1]$;
(7) f is a *scoring* function $f : [0, 1]^m \to [0, 1]$, which combines the scores of the m fuzzy predicates p_i into and overall *query score* to be assigned to the score variable s. We assume that f is *monotone*, i.e. for each vector $\mathbf{v} = (v_1, \ldots, v_m)$ and $\mathbf{v}' = (v'_1, \ldots, v'_m)$ in $[0, 1]^m$, if $\mathbf{v} \leqslant \mathbf{v}'$ then $f(\mathbf{v}) \leqslant f(\mathbf{v}')$, where $(v_1, \ldots, v_m) \leqslant (v'_1, \ldots, v'_m)$ iff $v_i \leqslant v'_i$ for all i;
(8) We assume that the computational cost of f and all fuzzy predicates p_i is bounded by a constant.

We call $s = f(p_1(\mathbf{z}_1), \ldots, p_m(\mathbf{z}_m))$ a *scoring atom*. For instance,

$$CheapCar(x, p, s) \leftarrow NewCar(x) \wedge CarPrice(x, p) \wedge s = \max(0, 1 - p/15,000),$$
$$CheapCar(x, p, s) \leftarrow SecondHandCar(x) \wedge CarPrice(x, p) \wedge s = \max(0, 1 - p/7500)$$

are two Top-k-Datalog rules looking for cheap cars, assigning to each car a score depending on its price. If the price of a new car is above 15,000 the car is not considered as a cheap one, while the scoring function is increasing as the price lowers. Hence, it is quite natural that if we are looking for cheap cars one wants that the retrieved cars are sorted in decreasing order with respect to its score, i.e. degree of cheapness. Furthermore, as the database may contain thousands of tuples, one usually wants to retrieve just the top-k ranked ones.

Top-k-Datalog fuzzy predicate: Concerning fuzzy predicates involved in scoring atoms, we recall that in fuzzy set theory and practice there are many membership functions for fuzzy sets membership specification. However, the *trapezoidal* $trz(k_1, k_2, a, b, c, d, x)$, the *triangular* $tri(k_1, k_2, a, b, c, x)$, the L-function (left shoulder function) $LS(k_1, k_2, a, b, x)$ and the R-function (right shoulder function) $RS(k_1, k_2, a, b, x)$ are simple, yet most frequently used to specify membership degrees (see Fig. 1). k_1, k_2 is the domain of the functions, a, b, c, d are the parameters and x is the input variable. For instance, $LS(k_1, k_2, a, b, a) = 1$, $LS(k_1, k_2, a, b, b) = 0$ and $LS(k_1, k_2, a, b, x) = (b-x)/(b-a)$ for $a < x < b$. To the ease of presentation, we will write $LS(k_1, k_2, a, b, x, y)$ to denote that the value of the L-function applied to x is y, and similarly for the other fuzzy predicates.

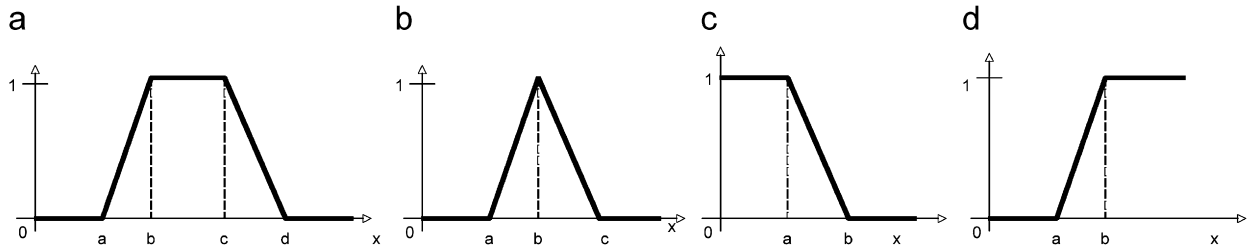

Fig. 1. (a) Trapezoidal function; (b) triangular function; (c) *L*-function; (d) *R*-function.

Of course, many other fuzzy predicates p can be defined as long as they assign to each tuple $\mathbf{c}$ a score $p(\mathbf{c}) \in [0, 1]$. For instance, we may define the unary fuzzy predicate *RipeTomato*, defined over constant representing colors, as

$$RipeTomato(red) = 1.0,$$
$$RipeTomato(slightlyred) = 0.75,$$
$$RipeTomato(greenred) = 0.5,$$
$$RipeTomato(slightlygreen) = 0.25,$$
$$RipeTomato(green) = 0.0.$$

Top-k-Datalog program: A Top-k-Datalog program is like a Datalog program except that Top-k-Datalog rules are considered in place of Datalog rules (i.e. scoring atoms may appear now in the rule body of "rules").

Semantics of Top-k-Datalog rules: From the semantics point of view, we have to take into account the additional scoring atom $s = f(p_1(\mathbf{z}_1), \ldots, p_m(\mathbf{z}_m))$. We note that, informally, a rule

$$q(\mathbf{x}, s) \leftarrow \exists \mathbf{y} body(\mathbf{x}, \mathbf{y}) \wedge s = f(p_1(\mathbf{z}_1), \ldots, p_m(\mathbf{z}_m))$$

is interpreted in an interpretation I as the set $q^{\mathcal{I}}$ of tuples $\langle \mathbf{c}, v \rangle$, such that when we substitute the variables $\mathbf{x}$ and s with the constants $\mathbf{c}$ and the score value $v \in [0, 1]$, the formula $\exists \mathbf{y} body(\mathbf{x}, \mathbf{y}) \wedge s = f(p_1(\mathbf{z}_1), \ldots, p_m(\mathbf{z}_m))$ is true in I.

Due to the existential quantification $\exists \mathbf{y}$, for a fixed $\mathbf{c}$, there may be many substitutions $\mathbf{c}'$ for $\mathbf{y}$ and, thus, we may have many possible scores for the tuple $\mathbf{c}$. Among all these scores for $\mathbf{c}$, we select the highest one, i.e. the *supremum* (sup).

In case that the atom $q(\mathbf{x}, s)$ is the head of multiple rules, for each tuple $\mathbf{c}$ there may be a score v_i computed by each of these rules. In that case, we assume that the overall score for $\mathbf{c}$ is the *maximum* among the scores v_i.

Now, let $\theta^{\mathbf{cc}'v}_{\mathbf{xy}s} = \{\mathbf{x}/\mathbf{c}, \mathbf{y}/\mathbf{c}', s/v\}$ be a substitution of the variables $\mathbf{x}$, $\mathbf{y}$ and s with the tuples $\mathbf{c}$, $\mathbf{c}'$ and score value $v \in [0, 1]$. Let $\psi(\mathbf{x}, \mathbf{y}, s)$ be $body(\mathbf{x}, \mathbf{y}) \wedge s = f(p_1(\mathbf{z}_1), \ldots, p_m(\mathbf{z}_m))$. With $\psi(\mathbf{x}, \mathbf{y}, s)\theta^{\mathbf{cc}'v}_{\mathbf{xy}s}$ we denote the ground formula obtained by applying the substitution $\theta^{\mathbf{cc}'v}_{\mathbf{xy}s}$ to $\psi(\mathbf{x}, \mathbf{y}, s)$.

We say that an interpretation I is a *model* of $\psi(\mathbf{x}, \mathbf{y}, s)\theta^{\mathbf{cc}'v}_{\mathbf{xy}s}$ iff $\psi(\mathbf{x}, \mathbf{y}, s)\theta^{\mathbf{cc}'v}_{\mathbf{xy}s}$ is true in I, i.e. all ground atoms and the grounded scoring atom occurring in $\psi(\mathbf{x}, \mathbf{y}, s)\theta^{\mathbf{cc}'v}_{\mathbf{xy}s}$ are true. We will write $I \models \psi(\mathbf{x}, \mathbf{y}, s)\theta^{\mathbf{cc}'v}_{\mathbf{xy}s}$ in this case.

Then, for a predicate symbol q, the interpretation $\mathbf{q}^{\mathcal{I}}$ of a set of rules $R_q = \{r_1, \ldots, r_n\}$, where each rule $r_i \in R_q$ is of the form $q(\mathbf{x}, s) \leftarrow \exists \mathbf{y}_i \psi_i(\mathbf{x}, \mathbf{y}_i, s)$, is

$$\mathbf{q}^{\mathcal{I}} = \left\{ \langle \mathbf{c}, v \rangle | v = \max(v_1, \ldots, v_n), v_i = \sup_{\mathbf{c}'} \{ v' | I \models \psi_i(\mathbf{x}, \mathbf{y}_i, s)\theta^{\mathbf{cc}'v'}_{\mathbf{xy}_i s} \} \right\}, \tag{1}$$

where $\sup \emptyset$ is undefined, and $\max(v_1, \ldots, v_n)$ is undefined iff all its arguments are undefined.

Note that some tuples $\mathbf{c}$ may not have a score in I and, thus, $\langle \mathbf{c}, v \rangle \notin \mathbf{r}^{\mathcal{I}}$ for no $v \in [0, 1]$. Alternatively we may define $\sup \emptyset = 0$ and, thus, all tuples $\mathbf{c}$ have a score in I, i.e. $\langle \mathbf{c}, v \rangle \in \mathbf{q}^{\mathcal{I}}$ for some $v \in [0, 1]$. We use the former formulation to distinguish the case where a tuple $\mathbf{c}$ is retrieved, though the score is 0, from the tuples which do not satisfy the query and, thus, are not retrieved.

Finally, for all $\mathbf{c}$ and for all $v \in [0, 1]$, we say that I is a *model* of $q(\mathbf{c}, v)$ iff $\langle \mathbf{c}, v \rangle \in \mathbf{q}^{\mathcal{I}}$.

It is not difficult to verify that, as for Datalog, Top-k-Datalog programs have an unique minimal model $M_{\mathcal{P}}$, which can be obtained as least fixed-point of the $T_{\mathcal{P}}$ operator. This follows immediately from [33] and the fact that scoring functions are monotone. We say that a Top-k-Datalog program $\mathcal{P}$ *entails* $q(\mathbf{c}, v)$, written $\mathcal{P} \models q(\mathbf{c}, v)$, iff $M_{\mathcal{P}} \models q(\mathbf{c}, v)$.

The basic inference service that concerns us is the top-k retrieval problem:

Top-k retrieval: Given a Top-k-Datalog program $\mathcal{P}$, retrieve the top-k ranked tuples $\langle \mathbf{c}, v \rangle$ that instantiate the query atom q and rank them in decreasing order w.r.t. the score v, i.e. find the top-k ranked tuples of the answer set of q, denoted

$$ans_k(\mathcal{P}, q) = \mathtt{Top}_k\{\langle \mathbf{c}, v \rangle | \mathcal{P} \models q(\mathbf{c}, v)\},$$

where $\mathtt{Top}_k$ is defined as follows: $S = \mathtt{Top}_k R$ iff

(1) $|S| \leqslant k$;
(2) for each $\langle \mathbf{c}, v \rangle \in S$ there is no $\langle \mathbf{c}', v' \rangle \in R$ such that $v' < v$.

For instance,

$$q(x, p, s) \leftarrow CheapCar(x, p, s)$$

is a query asking for cheap cars. The top-k ranked cars, according to the score (that depends on their price), are obtained by $ans_k(\mathcal{P}, q)$.

Finally, note that due to the scoring functions, it may happen that a finite number of steps is not sufficient to compute the minimal model. A well-known example is

$$r(x, s) \leftarrow r(x, s_1) \wedge s = (s_1 + 1)/2,$$
$$r(a, 0).$$

It is not difficult to see that $\mathcal{P} \models r(a, 1)$ holds, which is attained after ω $T_{\mathcal{P}}$ iterations only. To *always* guarantee the termination of a program, we will assume that the score has a fixed precision, i.e. the score s has always to belong to a fixed and finite set $\{0, 1/n, 2/n, \ldots, (n-1)/n, 1\}$ for a natural number n, e.g. $n = 100$. Rounding of the score is done in the usual way, e.g. for $n = 100$, 0.374 becomes 0.37, while 0.375 becomes 0.38. Decidability of the top-k retrieval problem is guaranteed as $\mathcal{P}^*$ is finite and the complexity of this problem is exponential in combined complexity (w.r.t. the size of $\mathcal{P}$) and polynomial in data complexity (w.r.t. the size of $\mathcal{P}_{\mathrm{E}}$) [8], where we assume that the scoring functions have constant cost.

3. Matchmaking requirements

In the rest of the paper, examples refer to an automobile marketplace; we motivate our work in this domain, without loss of generality. An automobile P2P e-marketplace supporting users during the bargaining process has to represent features such as e.g. look, comfort, optionals, model as well as numerical features, such as price, warranty or delivery time. In fact, based on these features both the buyer is able to formulate her request and the seller to describe the good to be sold. On the other hand, they can express different preferences on several features, i.e. a buyer can specify conditional preferences, such as

(1) *if the car is a sports one*, *then the fuel type has to be gasoline*;
or she can state that she prefers
(2) *a cheap car*, *yet if the car is provided with a GPS system she is ready to pay up to* 17,000€;
or the seller can offer
(3) *a sedan with* 4 *year warranty or* 120,000 km *warranty*.

Looking at the above examples, we observe that constraints can involve only numerical features (# 3), or non-numerical ones (# 1), as well as both of them (# 2). Furthermore, we notice that the buyer's requests (# 1 and # 2) can be split into two different parts. In fact, the buyer expresses as a *hard* requirement that if the car is a sports one then the fuel type has to be gasoline and as a preference, a *soft* constraint, that she is willing to buy a cheap car but that she could spend some more if the car is equipped with a GPS system. Of course, the same can be stated for a seller's supply (Sedan can be a hard constraint, while warranty can be specified as soft).

Hard/soft constraints: In a typical e-marketplace scenario, the issues within both the buyer's request and the seller's offer can hence be split into *strict requirements* and *preferences*. Strict requirements represent what the buyer and the seller want necessarily to be satisfied in order to accept an agreement—in our framework we call strict requirements *hard constraints*. Preferences denote issues they are willing to negotiate on—this is what we call *soft constraints*. Hence, the matchmaker has to be able to handle both hard and soft specifications of both the buyer and the seller. Let us now introduce an example request, we will use to explain some aspects of our approach:

Example 1. Suppose to have a buyer's request like "I want a black or grey station wagon. Preferably I would like to pay less than 14, 000€ furthermore I'm willing to pay up to 17, 000€ if the warranty is greater or equal than 100,000 km. (I do not want to pay more than 19, 000€ and I don't want a car with a warranty less than 60,000 km)". In this example we identify:

`hard constraints` = {**Body Type**: *Station wagon*. **Color**: *Black or Grey*. **Price**: $\leqslant 19,000$€. **Warranty**: $\geqslant 60,000$ km}.

`soft constraints`={**Price**: $\leqslant 14,000$€. **Warranty-Price**: *if Warranty* $\geqslant 100,000$ km *then Price* $\leqslant 17,000$€}.

3.1. Preferences and utilities

In a matchmaking process, retrieving supplies matching the request taking into account only *hard constraints* would be trivial. Instead, given a request and a set of retrieved supplies, the matchmaker should find—and rank—the most *suitable* or *promising* matches to propose to both parties, by exploiting *soft constraints* expressed both by the buyer and the seller.

Then, among the supplies completely satisfying all the requirements modeled in *hard constraints*, the top-ranked ones will be those best satisfying features expressed in the *soft constraints* proposed both by the buyer and the seller.

For what concerns *soft constraints*, we observe that the buyer's and seller's satisfaction degree, with respect to a final agreement, depends on which parts of her preference specifications have been satisfied. For instance, w.r.t. Example 1 suppose to have the following two supplies [1]:

σ' : **Body Type**: *Station wagon*. **Color**: *Grey*. **Price**: 16, 000€. **Warranty**: 200,000 km.
σ'' : **Body Type**: *Station wagon*. **Color**: *Black*. **Price**: 13, 000€. **Warranty**: 50,000 km.

Comparing these supplies with buyer's *soft constraints* we note that σ' satisfies the preference $\beta_2 = \{$**Warranty-Price** : if Warranty $\geqslant 100,000$ km then Price $\leqslant 17,000$€$\}$, while σ'' the preference $\beta_1 = \{$**Price** : $\leqslant 14,000$€$\}$. How to evaluate the best one? We expect the buyer to assign a positive utility value representing the preference relevance to sub-parts of *soft constraints*. In this case we assume utility values—call them $u(\beta_1)$ and $u(\beta_2)$—both for β_1 and β_2. It is not in the scope of this paper to investigate on how to compute $u(\beta_1)$ and $u(\beta_2)$; we might assume, without loss of generality, they are determined in advance by means of either direct assignment methods (ordering, simple assessing or ratio comparison) or pairwise comparison methods (like AHP and geometric mean) [29].

Actually, the same holds from the seller's perspective. In fact, in a P2P e-marketplace the seller may express his preferences—*soft constraints* e.g. on selling price, warranty, delivery time—with corresponding utilities $u(\sigma_j)$, as well as his *hard constraints* (e.g. color, model, engine fuel, etc.).

The only constraint on utility values is that both seller's and buyer's ones are normalized to 1 to eliminate outliers, and make them comparable [18]:

$$\sum u(\beta_i) = 1, \tag{2}$$

$$\sum u(\sigma_j) = 1. \tag{3}$$

[1] For the sake of simplicity in this section we consider supplies where *soft constraints* are not specified.

3.2. Utility functions

As pointed out so far, both buyer and seller may express some preferences (*soft constraints*) on features, or their combination. In Section 3.1 we also stated that usually a buyer sets a utility value $u(\beta_i)$ for each of the preferences β_i she expresses—the same can be stated if also the seller expresses preferences σ_j with practically no change. Usually the utility function is defined considering user preferences and allowing to assess a user's interest in an item, or a bundle of them [18]. Since we assume utilities on preferences as additive, here we can write the global utility of the buyer u_β and of the seller u_σ as just a sum of the utilities of preferences satisfied in the match. If we are able, for each preference β_i of the buyer and σ_j of the seller, to evaluate a score s_i and s_j representing a degree of preference satisfaction, then it is reasonable to think that the global utility has to take into account also this information. Hence, in formulas, we write the two utility functions as:

$$u_\beta = \sum\{s_i * u(\beta_i) | \beta_i \text{ is satisfied in the final agreement}\}, \tag{4}$$

$$u_\sigma = \sum\{s_j * u(\sigma_j) | \sigma_j \text{ is satisfied in the final agreement}\}. \tag{5}$$

3.3. Reservation values

From Example 1, we note that while considering numerical features, it is still possible to express hard and soft constraints on them. A hard constraint on a numerical feature can be considered as a *reservation value* [30] on the feature itself. In Example 1 the buyer expresses two reservation values, the one on price "more than 19, 000€", the other on warranty "less than 60,000 km". Both buyer and seller have their own reservation values on each feature involved in the negotiation process. It is the maximum (or minimum) value in the range of possible feature values to reach an agreement, e.g. the maximum price the buyer wants to pay for a car or the minimum warranty required, as well as, from the seller's perspective, the minimum price he will accept to sell the car or the minimum delivery time. Usually the reservation value is *private information*: each part knows its own reservation value and ignores the opponent's one. Keeping the price example, let us suppose that the maximum price the buyer is willing to pay is 19,000, while the seller minimum allowable price is 12,000, then we can set the two reservation values: $r_{\beta,\texttt{price}} = 19,000$ and $r_{\sigma,\texttt{price}} = 12,000$, so the *agreement price* will be in the interval $[12,000 \ldots 19,000]$. Otherwise if $r_{\beta,\texttt{price}} \leqslant r_{\sigma,\texttt{price}}$ no agreement will be possible and the supply will not be retrieved. As *private information* the reservation value is not revealed to the other party, but will be taken into account by the matchmaker retrieving the most promising supplies. Since setting a reservation value on a numerical feature is equivalent to set a hard constraint, then, once the buyer and the seller express their hard constraints, reservation values have to be added to them (see Example 1).

3.4. The matchmaking process

In this section, based on the notions of preferences, utility, and reservation values introduced so far, we begin outlining the actual matchmaking process. We start describing the seller's advertisement publication within the marketplace; then we continue following the buyer formulating her request; we close outlining the retrieval of most promising matches, i.e. the matchmaking process itself, which will be detailed next.

(1) Every time a seller enters the marketplace, he proposes his supply expressing both *hard constraints* and *soft constraints* (preferences). Then, he sets his reservation value $r_{\sigma,f}$ on each numerical feature f. Eventually, for each preference σ_j (if any) he expresses the corresponding utility $u(\sigma_j)$.
(2) Similarly to the seller, once the buyer enters the marketplace, she formulates her request defining *hard constraints* and preferences. After she sets a numerical value $r_{\beta,f}$ as her own reservation value on each f, she sets the utility $u(\beta_i)$ on each preference β_i.
(3) Based on buyer's and seller's specifications, the matchmaker returns a ranked list of supplies such that: (a) they satisfy both the *hard constraints* in the request and conversely their *hard constraints* are satisfied by the request; (b) the rank is evaluated taking into account preferences and utility functions u_β and u_σ as defined by Eqs. (4) and (5).

In a P2P e-marketplace, usually the aim is to find matches maximizing not only the buyer's satisfaction or the seller's one, but trying to make them equally satisfied. So the matchmaker has to propose matches mutually beneficial for both of them. Such matches are computed considering the higher values of the product $u_\beta * u_\sigma$ [26].

4. Representation of fuzzy requirements

Marketplaces are typical scenarios where the notion of fuzziness is often involved. The concept of *Cheap* or *Expensive* is quite usual. Similarly, numerical variables involved in a commercial transaction expose a fuzzy behavior. For instance, suppose to have a buyer looking for a car provided with a warranty greater than 100,000 km and a supplier selling his car with a 80,000 km warranty. We cannot say they do not match at all. Instead we can say they match with a certain degree. A logical language able to deal with fuzzy information would be then a good choice to model matchmaking.

To represent buyer/seller requirements in a Top-k-Datalog setting, hereafter we use the following notation:

$$\textit{hard constraints} = \begin{cases} \beta(x, \mathbf{y}) \text{ or } \beta(x) & \text{buyer's strict requirements,} \\ \sigma(x, \mathbf{y}) \text{ or } \sigma(x) & \text{seller's strict requirements,} \end{cases}$$

$$\textit{soft constraints} = \begin{cases} \beta_i(x, \mathbf{y}, s) \text{ or } \beta_i(x, s) & \text{buyer's preferences,} \\ \sigma_j(x, \mathbf{y}, s) \text{ or } \sigma_j(x, s) & \text{seller's preferences,} \end{cases}$$

where x is a single variable, $\mathbf{y}$ (if present) is a vector of variables and s represents the score variable as defined in Section 2.2. Here we consider fuzzy predicates only in *soft constraints*, while we do not allow them in *hard constraints*. In fact, because of the nature of fuzzy predicates they can suitably model preferences, while they are inadequate to model hard constraints. Moreover, since a score is associated to each fuzzy predicate, we can compute the global utility based on the two utility functions (4) and (5) in Section 3.2.

The two predicates σ and β model the minimal requirements the buyer and the seller want to be satisfied in order to accept an agreement. Notice that, if seller and buyer set hard constraints in conflict with each other, the corresponding supply will not be retrieved and no agreement will be reached. *Soft constraints* are modeled via Datalog predicates β_i for the buyer and σ_j for the seller, where each of them represents a sub-part of the buyer/seller preferences.

The use of x, $\mathbf{y}$ and s should be clearer looking at how the buyer's request in Example 1 is formalized:

$$\begin{aligned}
\beta_A(x) &\leftarrow \mathit{StationWagon}(x), \\
\beta_B(x, p) &\leftarrow \mathit{CarPrice}(x, p) \wedge (0 \leqslant p \leqslant 19,000), \\
\beta_C(x, kmw) &\leftarrow \mathit{KmWarranty}(x, kmw) \wedge (kmw \geqslant 60,000), \\
\beta_D(x) &\leftarrow \mathit{Grey}(x), \\
\beta_D(x) &\leftarrow \mathit{Black}(x), \\
\beta(x, p, kmw) &\leftarrow \beta_A(x) \wedge \beta_B(x, p) \wedge \beta_C(x, kmw) \wedge \beta_D(x), \\
\beta_1(x, p, s) &\leftarrow \mathit{CarPrice}(x, p) \wedge \mathit{LS}(0, 100,000, 14,000, 16,000, p, s), \\
\beta_2(x, p, kmw, s) &\leftarrow \mathit{KmWarranty}(x, kmw) \wedge \mathit{CarPrice}(x, p) \wedge \mathit{RS}(0, 400,000, 80,000, 100,000, kmw, s_1) \\
&\quad \wedge \mathit{LS}(0, 100,000, 17,000, 19,000, p, s_2) \wedge s = \max(1 - s_1, s_2).
\end{aligned}$$

With respect to the previous encoding we notice that:

(1) both in β and in β_2 we have $\mathbf{y} = (p, kmw)$ while in β_1 there is $\mathbf{y} = (p)$;
(2) defining β_1 and β_2 here we use two different *L-functions* to specify membership degrees for the variable p;
(3) looking at price and the corresponding variable p we note the Datalog encoding of a requirement changes whether it is a soft or a hard constraint—a simple inequality in β_B, an *L-function* in β_2;
(4) reservation values, both on price and on kilometers warranty, are encoded within strict requirements—as β_B for price and β_C for km warranty.

5. Fuzzy matchmaking in Top-k-Datalog

Now we have all what is needed to model the matchmaking framework in a Top-k-Datalog setting. First of all we show how to write the corresponding Top-k-Datalog program $\mathcal{P}_{\text{match}} = \langle \mathcal{P}_{\text{I}}, \mathcal{P}_{\text{E}} \rangle$.

(1) for each supply write the corresponding Datalog fact and add it to the EDB $\mathcal{P}_{\text{E}}$ as shown in Section 2.1;
(2) encode buyer's *hard constraints* requirements as a Datalog rule where the head contains the predicate $\beta(x, \mathbf{y})$ as shown in Section 4; add the rule to the IDB $\mathcal{P}_{\text{I}}$;
(3) encode seller's *hard constraints* requirements as a Datalog rule where the head contains the predicate $\sigma(x, \mathbf{y})$; add the rule to $\mathcal{P}_{\text{I}}$;
(4) for each buyer's preference β_i, write the corresponding rule in Top-k-Datalog where the head contains the predicate $\beta_i(x, \mathbf{y}, s)$ as shown in Section 4; add the rule to $\mathcal{P}_{\text{I}}$; set the utility value $u(\beta_i)$ as shown in Section 3.1;
(5) for each seller's preference σ_j, write the corresponding rule in Top-k-Datalog where the head contains the predicate $\sigma_j(x, \mathbf{y}, s)$ as shown in Section 4; add the rule to $\mathcal{P}_{\text{I}}$; set the utility value $u(\sigma_j)$ as shown in Section 3.1;
(6) add to $\mathcal{P}_{\text{I}}$ the rules:

$$Buyer(x, \mathbf{y}, u_\beta) \leftarrow \beta(x, \mathbf{y_0}) \wedge \beta_1(x, \mathbf{y_1}, s_1) \wedge \beta_2(x, \mathbf{y_2}, s_2) \wedge \cdots \wedge u_\beta = u(\beta_1) * s_1 + u(\beta_2) * s_2 + \cdots \tag{6}$$

$$Seller(x, \mathbf{y}, u_\sigma) \leftarrow \sigma(x, \mathbf{y_0}) \wedge \sigma_1(x, \mathbf{y_1}, s_1) \wedge \sigma_2(x, \mathbf{y_2}, s_2) \wedge \cdots \wedge u_\sigma = u(\sigma_1) * s_1 + u(\sigma_2) * s_2 + \cdots \tag{7}$$

where for each variable in $\mathbf{y}$ in the head of one of the two previous rules, the same variable occurs in at least one of the vectors of the corresponding body: $\mathbf{y_0}, \mathbf{y_1}, \mathbf{y_2}, \ldots$;
(7) add to $\mathcal{P}_{\text{I}}$ the rule:

$$Match(x, \mathbf{y}, u) \leftarrow Buyer(x, \mathbf{y_\beta}, u_\beta) \wedge Seller(x, \mathbf{y_\sigma}, u_\sigma) \wedge u = u_\beta * u_\sigma, \tag{8}$$

where for each variable in $\mathbf{y}$ in the head of the rule, the same variable occurs in $\mathbf{y_\beta}$ or $\mathbf{y_\sigma}$

Once we have the Top-k-Datalog program $\mathcal{P}_{\text{match}}$, then we solve the following **Top-k retrieval** problem:

$$ans_k(\mathcal{P}_{\text{match}}, Match) = \texttt{Top}_k\{\langle x, \mathbf{y}, u\rangle | \langle \mathbf{y}, u\rangle \in \texttt{Top}_1\{\langle x, \mathbf{y}', u'\rangle | \mathcal{P} \models Match(x, \mathbf{y}', u')\}\}.$$

Basically, for each key value x of the database, we compute the best matching $\langle \mathbf{y}, u\rangle$ for it, i.e. $\langle \mathbf{y}, u\rangle \in \texttt{Top}_1\{\langle x, \mathbf{y}', u'\rangle | \mathcal{P} \models Match(x, \mathbf{y}', u')\}$, and then rank the top-$k$ key values.

Notice that the rank is computed considering the product of buyer's and seller's utilities as stated at the end of Section 3.4.

We point out that in the above definition we used nested top-k, top-1 computation. The reason is that for an item x there may be different values for the parameters $\mathbf{y}$, for which there is a non-zero utility u (e.g. for a car, the price, km_warranty, delivery_time, etc. may be parameters and potentially lead to several optimal configurations). With the nested top-k formulation, we have chosen that we present for each x, an optimal configuration of the parameters $\mathbf{y}$ (the top-1 part). This will avoid that e.g. for the same car, several other options are presented in the top-k list. Of course, this choice can trivially be modified to a simple top-k definition $\texttt{Top}_k\{\langle x, \mathbf{y}, u\rangle | \mathcal{P} \models Match(x, \mathbf{y}, u)\}$ or even a nested top-k, top-n definition (for each x consider the top-n options for it) of the form $\texttt{Top}_k\{\langle x, \mathbf{y}, u\rangle | \langle \mathbf{y}, u\rangle \in \texttt{Top}_n\{\langle x, \mathbf{y}', u'\rangle | \mathcal{P} \models Match(x, \mathbf{y}', u')\}\}$.

6. Rules for e-marketplaces classification

Item classification is a very useful tool for P2P marketplaces. Looking at the most famous on-line e-marketplaces e.g. Google Base (Fig. 2a), Autos.com (Fig. 2b), eBay (Fig. 2c) we note that they all expose a personal classification to guide the user during the search process. The search can be performed based exclusively on a classification hierarchy or mixed with either a keyword based or feature value-based search.

In this section we show that e-marketplace categorization trees can be formalized as Top-k-Datalog rules. By adding these rules to the program $\mathcal{P}_{\text{match}}$ it will then be possible to compute the final agreement taking into account this background domain knowledge. In order to show the approach we start with an example taken from Autos.com. Here we find that *Mazda* 3 is produced by Mazda and is classified as *Compact Cars*. Furthermore Autos.com classifies

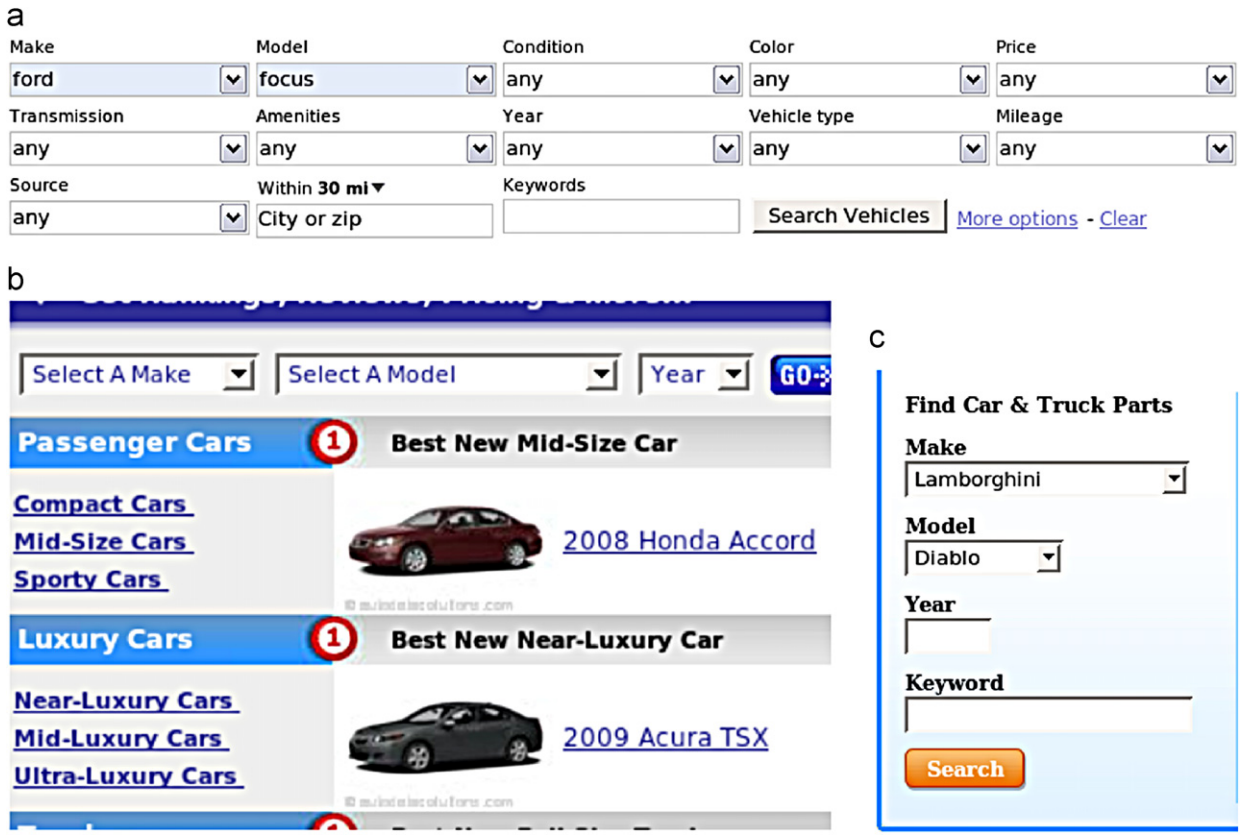

Fig. 2. (a) Google Base Vehicles, (b) Autos.com categories, (c) eBay Motors.

Table 1
The *CarTable* relation with a sample set of offers

ID	Model	Year	Price	Discount (%)	Km	Color	Airbag	Interior type	Air cond.	Engine fuel
455	MAZDA 3	2004	15,000	10	18,000	Red	0	VelvetSeats	1	Gasoline
34	ALFA 156	2002	12,000	20	25,000	Black	1	LeatherSeats	0	Diesel
1812	FORD FOCUS	2001	13,000	20	20,000	Gray	1	LeatherSeats	1	Gasoline

Compact Cars as *Passenger Cars* (see Fig. 2b). With respect to the schema of the example database table sketched in Table 1, we write the Top-k-Datalog rules:

$$\begin{aligned} Car(x) &\leftarrow PassengerCar(x),\\ PassengerCar(x) &\leftarrow CompactCar(x),\\ CompactCar(ID) &\leftarrow CarTable(ID, MODEL, YEAR, PRICE, \ldots) \wedge MODEL = \text{“}Mazda\ 3\text{”},\\ Mazda(ID) &\leftarrow CarTable(ID, MODEL, YEAR, PRICE, \ldots) \wedge MODEL = \text{“}Mazda\ 3\text{”}. \end{aligned}$$

Also notice that within the same Top-k-Datalog program you can also add category information from different marketplaces. For example Mazda 3 is classified by Google Base Vehicles and eBay Motors as a sedan. Hence, to model also these classifications we add to the Top-k-Datalog program the rule:

$$Sedan(ID) \leftarrow CarTable(ID, MODEL, YEAR, PRICE, \ldots) \wedge MODEL = \text{“}Mazda\ 3\text{”}.$$

$$CarTable(455, FORD\ FOCUS, 2004, 15{,}000, 10\%, 18{,}000, Red, 0, VelvetSeats, 1, Gasoline)$$
$$CarTable(34, ALFA\ 156, 2002, 12{,}000, 20\%, 25{,}000, Black, 1, LeatherSeats, 0, Diesel)$$
$$CarTable(1812, MAZDA\ 3, 2001, 13{,}000, 20\%, 20000, Gray, 1, \texttt{LeatherSeats}, 1, Gasoline)$$

$$\begin{aligned}
Car(x_1) &\leftarrow CarTable(x_1, \ldots, x_{11}) \\
Sedan(x_1) &\leftarrow CarTable(x_1, \ldots, x_{11}) \wedge x_2 = MAZDA\ 3 \\
Mazda(x_1) &\leftarrow CarTable(x_1, \ldots, x_{11}) \wedge x_2 = MAZDA\ 3 \\
Sedan(x_1) &\leftarrow CarTable(x_1, \ldots, x_{11}) \wedge x_2 = ALFA\ 156 \\
AlfaRomeo(x_1) &\leftarrow CarTable(x_1, \ldots, x_{11}) \wedge x_2 = ALFA\ 156 \\
StationWagon(x_1) &\leftarrow CarTable(x_1, \ldots, x_{11}) \wedge x_2 = FORD\ FOCUS \\
Ford(x_1) &\leftarrow CarTable(x_1, \ldots, x_{11}) \wedge x_2 = FORD\ FOCUS
\end{aligned}$$

$$\begin{aligned}
AirConditioning(x_1) &\leftarrow CarTable(x_1 \ldots, x_{11}) \wedge x_{10} = 1 \\
NoAirConditioning(x_1) &\leftarrow CarTable(x_1, \ldots, x_{11}) \wedge x_{10} = 0 \\
LeatherSeats(x_1) &\leftarrow CarTable(x_1, \ldots, x_{11}) \wedge x_9 = LeatherSeats \\
NoLeatherSeats(x_1) &\leftarrow CarTable(x_1, \ldots, x_{11}) \wedge x_9 \neq LeatherSeats \\
VelvetSeats(x_1) &\leftarrow CarTable(x_1, \ldots, x_{11}) \wedge x_9 = VelvetSeats \\
NoVelvetSeats(x_1) &\leftarrow CarTable(x_1, \ldots, x_{11}) \wedge x_9 \neq VelvetSeats
\end{aligned}$$

$$\ldots$$

$$\begin{aligned}
CarPrice(x, p) &\leftarrow PossibleCarPrice(x, p, min) \\
PossibleCarPrice(x_1, p, min) &\leftarrow CarTable(x_1, \ldots, x_{11}) \wedge p = x_4 \wedge \\
&\qquad min = p - p \cdot x_5 \\
PossibleCarPrice(x_1, p, min) &\leftarrow PossibleCarPrice(x_1, y, min) \wedge p \geq min \wedge p = y - 100 \\
Kilometers(x_1, x_6), &\leftarrow CarTable(x_1, \ldots, x_{11}) \\
CatalogluePrice(x_1, x_4), &\leftarrow CarTable(x_1, \ldots, x_{11}) \\
MinimalPrice(x_1, y), &\leftarrow CarTable(x_1, \ldots, x_{11}) \wedge y = x_4 - x_4 \cdot x_5 \\
LS(k_1, k_2, a, b, p, 1) &\leftarrow k_1 \leq p \leq a \\
LS(k_1, k_2, a, b, p, 0) &\leftarrow b \leq p \leq k_2 \\
LS(k_1, k_2, a, b, p, s) &\leftarrow (a < p < b) \wedge s = (b - p)/(b - a) \\
RS(k_1, k_2, a, b, p, 0) &\leftarrow k_1 \leq p \leq a \\
RS(k_1, k_2, a, b, p, 1) &\leftarrow b \leq p \leq k_2 \\
RS(k_1, k_2, a, b, p, s) &\leftarrow (a < p < b) \wedge s = (p - a)/(b - a)
\end{aligned}$$

Fig. 3. A part of the Top-k-Datalog program $\mathcal{P}_{\text{match}}$ used in the example, representing the knowledge domain.

6.1. An illustrative example

Let us present a tiny example in order to better clarify the approach. In Table 1 some possible offers stored in an e-marketplace database are presented. We call *CarTable* the relation representing Table 1.

Based both on these information and some specific domain knowledge, we write the corresponding Top-k-Datalog program $\mathcal{P}_{\text{match}}$ as explained is Sections 5 and 6. In Fig 3 facts in the EDB $\mathcal{P}_{\text{E}}$ represent the three tuples in Table 1.

Now, suppose to have the following buyer's request:

Hard constraint:

$\beta =$ I want a *sedan* or a *station wagon*. I don't want to pay *more than* 13, 000€.

Soft constraint:

$\beta_1 =$ I would like *air conditioning* if the car has *leather seats*.
$\beta_2 =$ Preferably I would like to pay *less than* 11, 000€.
$\beta_3 =$ The car should have *less than* 15,000 km.

Preferences utilities:

$u(\beta_1) = 0.05$,
$u(\beta_2) = 0.5$,
$u(\beta_3) = 0.45$.

Then we add to the Top-k-Datalog program $\mathcal{P}_{\text{match}}$ the rules:

$$\beta_A(x) \leftarrow \mathit{Sedan}(x), \tag{9}$$
$$\beta_A(x) \leftarrow \mathit{StationWagon}(x), \tag{10}$$
$$\beta_B(x, p) \leftarrow \mathit{CarPrice}(x, p) \wedge (0 \leqslant p \leqslant 13,000), \tag{11}$$
$$\beta(x, p) \leftarrow \beta_A(x) \wedge \beta_B(x, p), \tag{12}$$
$$\beta_1(x, 1) \leftarrow \mathit{NoLeatherSeats}(x), \tag{13}$$
$$\beta_1(x, 1) \leftarrow \mathit{LeatherSeats}(x) \wedge \mathit{AirConditioning}(x), \tag{14}$$
$$\beta_1(x, 0) \leftarrow \mathit{LeatherSeats}(x) \wedge \mathit{NoAirConditioning}(x), \tag{15}$$
$$\beta_2(x, p, s) \leftarrow \mathit{CarPrice}(x, p) \wedge \mathit{LS}(0, 100,000, 11,000, 13,000, p, s), \tag{16}$$
$$\beta_3(x, s) \leftarrow \mathit{Kilometers}(x, k) \wedge \mathit{LS}(0, 400,000, 15,000, 20,000, k, s), \tag{17}$$
$$\mathit{Buyer}(x, p, u_\beta) \leftarrow \beta(x, p) \wedge \beta_1(x, s_1) \wedge \beta_2(x, p, s_2) \wedge \beta_3(x, s_3) \wedge u_\beta = 0.05 * s_1 + 0.5 * s_2 + 0.45 * s_3. \tag{18}$$

Since we are in a P2P e-marketplace, also the seller can express hard and soft constraints. Looking at the information modeled in Table 1 we see that a soft constraint is expressed on price: the seller prefers to sell the car at the catalogue price, furthermore he may apply a discount. In this case also a reservation value is set; in fact, he does not want to go down such defined discount (hard constraint).

Seller's requirements are then encoded in $\mathcal{P}_{\text{match}}$ as

$$\sigma(x, p) \leftarrow \mathit{CarPrice}(x, p), \mathit{CataloguePrice}(x, catP) \wedge \mathit{MinimalPrice}(x, minP) \wedge (minP \leqslant p \leqslant catP), \tag{19}$$

$$\sigma_1(x, p, s) \leftarrow \mathit{CarPrice}(x, p) \wedge \mathit{CataloguePrice}(x, catP) \wedge \mathit{MinimalPrice}(x, minP) \wedge (minP \leqslant p \leqslant catP) \wedge \mathit{RS}(0, 100,000, minP, catP, p, s), \tag{20}$$

$$\mathit{Seller}(x, p, u_\sigma) \leftarrow \sigma(x, p) \wedge \sigma_1(x, p, s_1) \wedge u_\sigma = s_1. \tag{21}$$

Notice that, in this particular case, the specification of $u(\sigma_1)$ is not required because of Eq. (3).

According to the encoding in Section 5 the last rule to be added to $\mathcal{P}_{\text{match}}$ is the one related to the predicate *Match*.

$$\mathit{Match}(x, p, u) \leftarrow \mathit{Buyer}(x, p, u_\beta) \wedge \mathit{Seller}(x, p, u_\sigma) \wedge u = u_\beta \cdot u_\sigma.$$

Solving a top-3 retrieval problem with respect to $\mathcal{P}_{\text{match}}$ defined in this example, the ranked list of matches is:

x	p	u
34	11,300	0.30
1812	11,800	0.19

corresponding to `ALFA 156` and `FORD FOCUS`. Notice that, even if we solved a top-3 retrieval problem, we retrieve only two tuples. *MAZDA* 3 is not in the result set because of the conflict between seller's and buyer's hard constraints on price.

7. A general marketplace

We now extend the previous approach to a more realistic case, in which many buyers and many sellers operate in the same marketplace. We assume that each buyer and each seller has a unique identifier, generically denoted by $\#\underline{b}$, $\#\underline{s}$,

Table 2
CarTable relation with attribute SELLER# added

ID	Seller	Model	Year	Price	Disc. (%)	Km	Color	Airb.	Int. type	Air cond.	Eng. fuel
278	#3	MAZDA 3	2005	15,000	15	18,000	Red	0	VelvetSeats	1	Gasoline
455	#4	MAZDA 3	2004	15,000	10	22,000	Green	0	VelvetSeats	1	Gasoline
34	#3	ALFA 156	2002	12,000	20	25,000	Black	1	LeatherSeats	0	Diesel
1812	#4	FORD FOCUS	2001	13,000	20	20,000	Gray	1	LeatherSeats	1	Gasoline

respectively. $\mathcal{P}_{\mathrm{E}}$ contains two more tables, recording buyers and sellers data, and $\#\underline{b}$, $\#\underline{s}$ are the values of the primary key of each table. Moreover, $\#\underline{s}$ is also added as the value of a foreign key for the products table (in the Database jargon, tables for products and sellers have a many–one relation). Moreover, each buyer has its own utility values $u^{\#\underline{b}}$, and the same for sellers ($u^{\#\underline{s}}$). We restate Steps (1)–(7) of Section 5, pointing out the additions by extending the example, supposing now two buyers (#1, #2) and two sellers (#3, #4).

(1) for each pair $\langle \#\underline{s}, \mathit{supply}\rangle$ write the corresponding Datalog fact and add it to the EDB $\mathcal{P}_{\mathrm{E}}$; the preceding Table 1, once normalized, becomes the following one (2).[2] Of course, we assume that the *CarTable* predicate has now 12 arguments, and that all clauses involving it are adjusted accordingly Table 2.

(2) for each buyer $\#\underline{b}$, encode its *hard constraints* as a Datalog rule where the head contains the predicate $\beta(\#\underline{b}, x, \mathbf{y})$; add the rule to the IDB $\mathcal{P}_{\mathrm{I}}$. Note that the first argument $\#\underline{b}$ is a *constant* in the rule head; it *reifies* a relation $\beta^{\#\underline{b}}$. Supposing that the example in Section 6.1 referred to Buyer #1, the hard constraint (12) becomes

$$\beta(\#1, x, p) \leftarrow \beta_A(x) \wedge \beta_B(x, p).$$

Note that $\beta_A(x)$ and $\beta_B(x, p)$ are not affected by this extension; in fact, they represent constraints that could be used by other buyers too, as in

$$\beta(\#2, x, p) \leftarrow \beta_A(x) \wedge \beta_C(x, p)$$

expressing for Buyer #2 the same car type constraint as for Buyer #1, but a different constraint regarding its maximum price.

(3) for each seller $\#\underline{s}$, encode its *hard constraints* as a Datalog rule where the head contains the predicate $\sigma(\#\underline{s}, x, \mathbf{y})$; add the rule to $\mathcal{P}_{\mathrm{I}}$; supposing that the hard constraints (19) of the previous section referred to Seller #3, they would now be expressed as

$$\sigma(\#3, x, p) \leftarrow \mathit{CarPrice}(x, p), \mathit{CataloguePrice}(x, \mathit{catP}) \wedge \mathit{MinimalPrice}(x, \mathit{minP}) \wedge (\mathit{minP} \leqslant p \leqslant \mathit{catP})$$

and similarly for Seller #4.

(4) for each buyer $\#\underline{b}$, and each buyer's preference β_i, write the corresponding rule in Top-k-Datalog where the head contains the predicate $\beta_i(\#\underline{b}, x, \mathbf{y}, s)$; add the rule to $\mathcal{P}_{\mathrm{I}}$; set the utility value $u^{\#\underline{b}}(\beta_i)$ as shown in Section 3.1; continuing to modify the previous example, preferences (14) and (17) for Buyer #1 could be modified as follows:

$$\beta_1(\#1, x, 1) \leftarrow \mathit{LeatherSeats}(x) \wedge \mathit{AirConditioning}(x),$$
$$\beta_3(\#1, x, s) \leftarrow \mathit{Kilometers}(x, k) \wedge \mathit{LS}(0, 400, 000, 15, 000, 20, 000, k, s)$$

while each utility value $u^{\#1}(\beta_1) = 0.05$, $u^{\#1}(\beta_2) = 0.5$, $u^{\#1}(\beta_3) = 0.45$ is now specific for Buyer #1, and Buyer #2 has its own utilities, e.g. $u^{\#2}(\beta_1) = 0.3$, $u^{\#2}(\beta_2) = 0.7$.

(5) for each seller $\#\underline{s}$, and each seller's preference σ_j, write the corresponding rule in Top-k-Datalog where the head contains the predicate $\sigma_j(\#\underline{s}, x, \mathbf{y}, s)$; add the rule to $\mathcal{P}_{\mathrm{I}}$; set the utility value $u^{\#\underline{s}}(\sigma_j)$ as shown in Section 3.1. For example, Rule (20) would be rewritten by substituting its head with $\sigma_1(\#3, x, p, s)$.

(6) for each buyer $\#\underline{b}$, and each seller $\#\underline{s}$, add to $\mathcal{P}_{\mathrm{I}}$ the rules:

$$\begin{aligned} \mathit{Buyer}(\#\underline{b}, x, \mathbf{y}, u_\beta) \leftarrow\ & \beta(\#\underline{b}, x, \mathbf{y_0}) \wedge \beta_1(\#\underline{b}, x, \mathbf{y_1}, s_1) \wedge \beta_2(\#\underline{b}, x, \mathbf{y_2}, s_2) \\ & \wedge \cdots \wedge u_\beta = u^{\#\underline{b}}(\beta_1) * s_1 + u^{\#\underline{b}}(\beta_2) * s_2 + \cdots, \end{aligned} \tag{22}$$

[2] Although we could introduce a new relation (ID,SELLER#), we prefer to add a field in *CarTable* to skip a join in the intensional program.

$$Seller(\#\underline{s}, x, \mathbf{y}, u_\sigma) \leftarrow \sigma(\#\underline{s}, x, \mathbf{y_0}) \wedge \sigma_1(\#\underline{s}, x, \mathbf{y_1}, s_1) \wedge \sigma_2(\#\underline{s}, x, \mathbf{y_2}, s_2) \wedge \cdots \wedge u_\sigma = u^{\#\underline{s}}(\sigma_1) * s_1 + u^{\#\underline{s}}(\sigma_2) * s_2 + \cdots, \tag{23}$$

where for each variable in $\mathbf{y}$, the same variable occurs in at least one of $\mathbf{y_0}, \mathbf{y_1}, \mathbf{y_2}, \ldots$; (observe that these rules must be written separately for each buyer and seller, since each actor has its own predicates β, σ, and values $u^{\#\underline{b}}, u^{\#\underline{s}}$; on the other hand, there is just one predicate *Buyer* and one predicate *Seller*). For the running example, we would have

$$Buyer(\#1, x, p, u_\beta) \leftarrow \beta(\#1, x, p) \wedge \beta_1(\#1, x, s_1) \wedge \beta_2(\#1, x, s_2) \wedge \beta_3(\#1, x, s_3) \wedge u_\beta = 0.05 * s_1 + 0.5 * s_2 + 0.45 * s_3,$$

$$Buyer(\#2, x, p, u_\beta) \leftarrow \beta(\#2, x, p) \wedge \cdots \wedge u_\beta = 0.3 * s_1 + 0.7 * s_2$$

and similarly for the predicate *Seller*().

(7) Finally, add to $\mathcal{P}_\mathrm{I}$ the rule:

$$Match(b, v, x, \mathbf{y}, u) \leftarrow Buyer(b, x, \mathbf{y_\beta}, u_\beta) \wedge Seller(v, x, \mathbf{y_\sigma}, u_\sigma) \wedge u = u_\beta * u_\sigma, \tag{24}$$

where for each variable in $\mathbf{y}$, the same variable occurs in $\mathbf{y_\beta}$ or $\mathbf{y_\sigma}$. Observe that b and v[3] are *variables*, that can unify only with a buyer and a seller, respectively, due to the new definitions (22)–(23) of *Buyer* and *Seller*.

Now we can have *two* matchmaking services in the marketplace. The first one is for a specific buyer $\#\underline{b}$, that is searching the k best sellers of a kind of product, according to its own preferences:

$$B\_ans_k(\mathcal{P}_\mathrm{match}, Match) = \mathtt{Top}_k\{\langle v, x, \mathbf{y}, u\rangle | \langle \mathbf{y}, u\rangle \in \mathtt{Top}_1\{\langle v, x, \mathbf{y}', u'\rangle | \mathcal{P} \models Match(\#\underline{b}, v, x, \mathbf{y}', u')\}\}.$$

The second matchmaking service is for a given seller $\#\underline{s}$, who is adopting a *push* marketing strategy, searching for the k most promising buyers for her products:

$$S\_ans_k(\mathcal{P}_\mathrm{match}, Match) = \mathtt{Top}_k\{\langle b, x, \mathbf{y}, u\rangle | \langle \mathbf{y}, u\rangle \in \mathtt{Top}_1\{\langle b, x, \mathbf{y}', u'\rangle | \mathcal{P} \models Match(b, \#\underline{s}, x, \mathbf{y}', u')\}\}.$$

7.1. Discussion

Having outlined our approach, we briefly discuss here some of its benefits and current limits. First, we estimate the scalability of our approach as the size of the e-marketplace grows. We note that both the size of $\mathcal{P}_\mathrm{I}$ and $\mathcal{P}_\mathrm{E}$ grows linearly in the number of buyers and sellers that enter the marketplace. This could seem a drawback, since the computational cost of the evaluation of a query B_ans_k, S_ans_k depends linearly on the size of $\mathcal{P}_\mathrm{I}$, and we expect the number of buyers and sellers to be quite large in a real application. However, the *depth* of the program *Match* is bounded by the depth of the predicates about preferences, which we expect, based on evaluation, to be a small constant (see also the examples presented in this paper). Hence, by standard Database optimization techniques working bottom-up, we can expect the cost of evaluating B_ans_k, S_ans_k to be only a small fraction of the total number of products in the marketplace.

Second, we trace some boundaries about the applicability of this methodology to different scenarios. Observe that our approach is strictly related to marketplaces regarding goods to be sold, since we need a relation about product characteristics, with a foreign key for the seller. We could not easily represent characteristics of the buyer, e.g. if the seller prefers buyers who never went bankrupt, or who live in the same city as the seller. Hence, our approach would not fit well to barter marketplaces, or, for instance, to dating services, since we would need two relations about characteristics of each actor.

8. Related work

Motivated by the information overload, which keeps growing with the Web's widespread use, the problem of matchmaking has been investigated under different perspectives and many approaches have been proposed. Obviously, issues arise when we are not dealing with trivial perfect matches. In fact, if a perfect match is not available, worse alternatives

[3] We use v for *v*endor—instead of s for seller—since s has already been used for scores.

may be accepted or a compromise may be negotiated [15], and finding best ones among possible alternatives is at the core of a matchmaking procedure. An initial approach to matchmaking can be dated back to vague query answering [25] where the need to go beyond pure relational databases was addressed using weights attributed to several search variables. More recently similar approaches have been proposed extending SQL with "preference" clauses, in order to allow relaxed queries in structured databases [15]. Preferences are modeled there as strict partial orders, still only buyer's preferences are taken into account while retrieving supplies. Final results are computed based on the "requester specification" only; no agreement is proposed as a result of the query process. Besides, being the approach grounded in a relational model, it seems tricky to model some background domain knowledge. Classified-ads matchmaking, at a syntactic level, was proposed in [31] to matchmake semi-structured descriptions advertising computational resources in a fashion anticipating grid resources brokering. Matchmaking was used in SIMS [3] to dynamically integrate queries; the approach used LOOM as description language. LOOM has also been used in the subsumption matching addressed in [13]. InfoSleuth [16], a system for discovery and integration of information, included an agent matchmaker, which adopted KIF and the deductive database language LDL + +. Constraint-based approaches to matchmaking have been proposed and implemented in several systems, e.g. [28,17,24]. Due to the growing interest in the Semantic Web initiative many approaches to matchmaking have been proposed in the framework of DAML + OIL, OWL and their grounding logical languages in particular description logics (DL). Matchmaking as satisfiability of concept conjunction in DLs was first proposed in [14,11]. In the framework of Retsina multiagent infrastructure [38], a specific language was defined for agent advertisement, and a matchmaking engine was developed [39,27], which carries out the process on five possible match levels. Di Noia [10] proposed algorithms to classify and semantically rank matches within classes, along with properties that a matchmaker should have in a DL-based framework. An initial DL-based approach, adopting penalty functions ranking, has been proposed in [4], in the framework of dating systems. An extended matchmaking approach, with negotiable and strict constraints in a DL framework has been proposed in [7], using both concept contraction and concept abduction. The formalization of concept contraction and concept abduction as non-monotonic reasoning tasks for semantic matchmaking have been presented in [9]. The need to work in someway with approximation and ranking in DL-based approaches to matchmaking has also recently led to adopting fuzzy-DLs, as in sMART [2] or hybrid approaches, as in the OWLS-MX matchmaker [19]. sMART is a semantic matchmaking portal, based on fuzzy-DLs, able to deal with approximation in the requests description—fuzzy request—handled by crisp DL-reasoners. Nevertheless in such approaches the matchmaking process is defined according to buyer's perspective: ranking a set of promising offers according to buyer's preferences. In our framework we model the matchmaking process in a P2P marketplace, taking into account not only the buyer's preferences, but also the seller's ones, finding the most promising agreements w.r.t. preferences of both. In [22] a language able to express conditional preferences is proposed to matchmake in DL based on *strength* values (weights) assigned to preferences. A ranking procedure is also proposed. The main aim of the approach is to retrieve a set of appealing available resources with respect to a request. Also in this case nothing is said on how to compute an agreement—as needed in P2P scenarios. Furthermore, the notion of fuzzy requirements is not addressed.

9. Conclusion

In this paper we presented a fuzzy matchmaking approach exploiting Datalog to find the most promising agreements in a P2P e-marketplace. More precisely, exploiting fuzzy rules we have been able to model *soft constraints*, and differently w.r.t. usual matchmaking approaches, by taking into account both buyer's and seller's preferences and utilities we are able to find most promising matches mutually beneficial for both peer entities. The actual P2P matchmaking process has been modeled as a Datalog program which is also able to consider background domain knowledge, while keeping the approach effective and scalable. While initial evaluations confirm the validity of the approach, the performance on large scale experiments are still ongoing on a prototypical platform implementing the whole proposed e-marketplace architecture. In future work, it might be interesting to explore the inclusion of non-monotonic negation into our framework, for which a top-k query answering procedure does not exist yet (while there are top-down algorithms for Normal Logic Programs managing vagueness [6,32–34]).

Acknowledgments

Some of the authors acknowledge support of EU FP-6 IST STREP TOWL co. 02689 project, and Apulia Region Strategic Projects PS_092, PS_121.

References

[1] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison Wesley Publishing Co., Reading, MA, 1995.
[2] S. Agarwal, S. Lamparter, sMart—a semantic matchmaking portal for electronic markets, in: Proc. 7th Internat. IEEE Conf. on E-Commerce Technology 2005, 2005.
[3] Y. Arens, C.A. Knoblock, W. Shen, Query reformulation for dynamic information integration, J. Intelligent Inform. Systems 6 (1996) 99–130.
[4] A. Calì, D. Calvanese, S. Colucci, T. Di Noia, F.M. Donini, A description logic based approach for matching user profiles, in: Proc. 17th Internat. Workshop on Description Logics (DL'04), CEUR Workshop Proceedings, Vol. 104, 2004.
[5] S. Ceri, G. Gottlob, L. Tanca, Logic Programming and Databases, Springer, Berlin, 1990.
[6] C. Chesnevar, G. Simari, T. Alsinet, L. Godo, A logic programming framework for possibilistic argumentation with vague knowledge, in: Proc. 20th Annu. Conf. on Uncertainty in Artificial Intelligence (UAI-04), AUAI Press, 2004.
[7] S. Colucci, T. Di Noia, E. Di Sciascio, F. Donini, M. Mongiello, Concept abduction and contraction for semantic-based discovery of matches and negotiation spaces in an e-marketplace, Electron. Commerce Res. Appl. 4 (4) (2005) 345–361.
[8] E. Dantsin, T. Eiter, G. Gottlob, A. Voronkov, Complexity and expressive power of logic programming, ACM Comput. Surveys 33 (3) (2001) 374–425.
[9] T. Di Noia, E. Di Sciascio, F. Donini, Semantic matchmaking as non-monotonic reasoning: a description logic approach, J. Artif. Intell. Res. 29 (2007) 269–307.
[10] T. Di Noia, E. Di Sciascio, F. Donini, M. Mongiello, A system for principled matchmaking in an electronic marketplace, Internat. J. Electron. Commerce 8 (4) (2004) 9–37.
[11] E. Di Sciascio, F. Donini, M. Mongiello, G. Piscitelli, A knowledge-based system for person-to-person e-commerce, in: Proc. of the KI-2001 Workshop on Applications of Description Logics (ADL-2001), CEUR Workshop Proceedings, Vol. 4, 2001.
[12] eBay Motors, viewed 1 March 2007 ⟨http://www.motors.eBay.com⟩.
[13] Y. Gil, S. Ramachandran, PHOSPHORUS: a task based agent matchmaker, in: Proc. Internat. Conf. on Autonomous Agents '01, ACM, New York, 2001.
[14] J. Gonzales-Castillo, D. Trastour, C. Bartolini, Description logics for matchmaking of services, in: Proc. of the KI-2001 Workshop on Applications of Description Logics (ADL-2001), CEUR Workshop Proceedings, Vol. 44, 2001.
[15] B. Hafenrichter, W. Kießling, Optimization of relational preference queries, in: Proc. of the 16th Australasian Database Conf. (ADC 2005), Newcastle, Australia, January 2005.
[16] N. Jacobs, R. Shea, Carnot and Infosleuth—database technology and the web, in: Proc. ACM SIGMOD Internat. Conf. on the Management of Data, 1995.
[17] Kasbah ⟨http://www.kasbah.com⟩.
[18] R.L. Keeney, H. Raiffa, Decisions with Multiple Objectives—Preferences and Value Trade-offs, Cambridge University Press, Cambridge, MA.
[19] M. Klusch, B. Fries, M. Khalid, K. Sycara, Owls-mx: hybrid owl-s service matchmaking, in: Proc. 1st Internat. AAAI Fall Symp. on Agents and the Semantic Web, 2005.
[20] D. Kuokka, L. Harada, Integrating information via matchmaking, J. Intelligent Inform. Systems 6 (1996) 261–279.
[21] J.W. Lloyd, Foundations of Logic Programming, Springer, Heidelberg, RG, 1987.
[22] T. Lukasiewicz, J. Schellhase, Variable-strength conditional preferences for matchmaking in description logics, in: Proc. 10th Internat. Conf. on Principles of Knowledge Representation and Reasoning (KR 06), 2006.
[23] T. Lukasiewicz, U. Straccia, Top-k retrieval in description logic programs under vagueness for the semantic web, in: Proc. 1st Internat. Conf. on Scalable Uncertainty Management (SUM-07), Lecture Notes in Computer Science, Vol. 4772, Springer, Berlin, 2007.
[24] P. Maes, R. Guttman, A. Moukas, Agents that buy and sell, Comm. ACM 42 (3) (1999) 81–91.
[25] A. Motro, VAGUE: a user interface to relational databases that permits vague queries, ACM Trans. Office Inform. Systems 6 (3) (1988) 187–214.
[26] J.F. Nash, The bargaining problem, Econometrica 18 (2) (1950) 155–162.
[27] M. Paolucci, T. Kawamura, T. Payne, K. Sycara, Semantic matching of web services capabilities, The Semantic Web—ISWC 2002, Vol. 2342, 2002, pp. 333–347.
[28] PersonaLogic ⟨http://www.PersonaLogic.com⟩.
[29] J. Pomerol, S. Barba-Romero, Multicriterion Decision Making in Management, Kluwer Series in Operation Research, Kluwer Academic Publishers, Dordrecht, 2000.
[30] H. Raiffa, J. Richardson, D. Metcalfe, Negotiation Analysis—the Science and Art of Collaborative Decision Making, The Belknap Press of Harvard University Press, 2002.
[31] R. Raman, M. Livny, M. Solomon, Matchmaking: distributed resource management for high throughput computing, in: Proc. of IEEE High Performance Distributed Computing Conf., 1998.
[32] M. Schroeder, R. Schweimeier, Fuzzy argumentation and extended logic programming, in: Proc. of ECSQARU Workshop Adventures in Argumentation, 2001.
[33] U. Straccia, Query answering in normal logic programs under uncertainty, in: 8th European Conf. on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-05), Lecture Notes in Computer Science, Vol. 3571, Springer, Barcelona, Spain, 2005.
[34] U. Straccia, Query answering under the any-world assumption for normal logic programs, in: Proc. 10th Internat. Conf. on Principles of Knowledge Representation (KR-06), AAAI Press, 2006.
[35] U. Straccia, Towards top-k query answering in deductive databases, in: Proc. 2006 IEEE Internat. Conf. on Systems, Man and Cybernetics (SMC-06), IEEE, New York, 2006.
[36] U. Straccia, Towards vague query answering in logic programming for logic-based information retrieval, in: World Congr. Internat. Fuzzy Systems Association (IFSA-07), Cancun, Mexico, 2007.

[37] Sunday Times, viewed 25 March 2007 ⟨http://driving.timesonline.co.uk⟩.
[38] K. Sycara, M. Paolucci, M. Van Velsen, J. Giampapa, The RETSINA MAS infrastructure, Autonomous Agents and Multi-Agent Systems 7 (2003) 29–48.
[39] K. Sycara, S. Widoff, M. Klusch, J. Lu, LARKS: dynamic matchmaking among heterogeneous software agents in cyberspace, Autonomous Agents and Multi-Agent Systems 5 (2002) 173–203.
[40] D. Trastour, C. Bartolini, C. Priest, Semantic web support for the business-to-business e-commerce lifecycle, in: Proc. Internat. World Wide Web Conf. (WWW) '02, ACM, New York, 2002.
[41] J.D. Ullman, Principles of Database and Knowledge Base Systems, Vols. 1 and 2, Computer Science Press, Potomac, MD, 1989.
[42] D. Veit, J. Muller, M. Schneider, B. Fiehn, Matchmaking for autonomous agents in electronic marketplaces, in: Proc. Internat. Conf. on Autonomous Agents '01, ACM, New York, 2001.
[43] XSB, viewed 12 March 2007 ⟨http://xsb.sourceforge.net⟩.
[44] Yahoo Personals, viewed 12 March 2007 ⟨http://personals.yahoo.com/⟩.