A Framework Recommending Top-k Web Service Compositions: A Fuzzy Set-Based Approach

Karim Benouaret, Djamal Benslimane, Allel Hadjali

▶ **To cite this version:**

Karim Benouaret, Djamal Benslimane, Allel Hadjali. A Framework Recommending Top-k Web Service Compositions: A Fuzzy Set-Based Approach. Applied Computing Review, 2011, 11, (3), pp.32-40. hal-00656140

HAL Id: hal-00656140
https://hal.science/hal-00656140

Submitted on 3 Jan 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Framework Recommending Top-k Web Service Compositions: A Fuzzy Set-Based Approach

Karim Benouaret
Lyon 1 University
69622, Villeurbanne, France
kbenouar@liris.cnrs.fr

Djamal Benslimane
Lyon 1 University
69622, Villeurbanne, France
dbenslim@liris.cnrs.fr

Allel Hadjali
Enssat-University of Rennes 1
22305, Lannion, France
hadjali@enssat.fr

ABSTRACT

Data Web services allow users to access information provided by different companies. Web users often need to compose different Web services to achieve a more complex task that can not be fulfilled by an individual Web services. In addition, user preferences are becoming increasingly important to personalize the composition process. In this paper, we propose an approach to compose data Web services in the context of preference queries where user preferences are modeled thanks to fuzzy sets that allow for a large variety of flexible terms such as "cheap", "affordable" and "fairly expensive". Our main objective is to find the top-k data Web service compositions that better satisfy the user preferences. The proposed approach is based on an RDF query rewriting algorithm to find the relevant data Web services that can contribute to the resolution of a given preference query. The constraints of the relevant data Web services are matched to the preferences involved in the query using a set of matching methods. A ranking criteria based on a fuzzyfication of Pareto dominance is defined in order to better rank the different data Web services/compositions. To select the top-k data Web services/compositions we develop a suitable algorithm that allows eliminating less relevant data Web services before the composition process. Finally, we evaluate our approach through a set of experiments.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Query formulation, Search process, Selection process; H.3.5 [**Online Information Services**]: Web-based services

General Terms

Measurement, Theory, Performance

Keywords

Fuzzy dominance, Web service, Top-k compositions, fuzzy preferences queries.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$10.00.

1. INTRODUCTION

Nowadays, an increasing among companies are moving towards a service-oriented architecture for data sharing on the Web by putting their data sources behind Web services, thereby providing a well-documented and interoperable method of interacting with their data [8, 16, 9]. In particular, if no single Web service can satisfy the functionality required by the user, there should be a possibility to combine existing services together in order to fulfill the request. In this context, we talk about *Data Web Service Composition*, where services correspond to calls over the data sources, i.e., the invocation of a data Web service results in the execution of a query over the data sources' schema.

On the other hand, user preferences are becoming increasingly important to personalize the composition process. For example, when looking for items to be purchased over the Web, customer preferences are critical in the search. A more general and suitable approach to model preferences is based on fuzzy sets theory [13]. Fuzzy sets are very well suited to the interpretation of linguistic terms, which constitute a convenient way for a user to express her/his preferences. For example, when expressing preferences about the price of a car, users often employ fuzzy terms like "rather cheap", "affordable" and "not expensive". However as data Web services and service providers proliferate, a large number of candidate compositions that would use different -most likely competing- services may be used to answer the same query. Hence, it is important to set up an effective service composition framework that would identify and retrieve the most relevant services and return the top-k compositions according to the user preferences.

The following example illustrates a typical scenario related to our previous discussion, showing the challenges in finding the top-k service compositions.

Example. Let us consider an e-commerce system supporting users to buy cars. The system has access to the set of services described in Table 1. The symbols "$" and "?" denote inputs and outputs of data services, respectively. Services providing the same functionality belong to the same service class. For instance, the services S_{21}, S_{22}, S_{23} and S_{24} belong to the same class $\mathcal{S}_2$. Each service has its (fuzzy) constraints on the data it manipulates. For instance, the cars returned by S_{21} are of *cheap* price and *short* warranty.

Assume that a user Bob wants to buy a car. He sets his preferences and submits the following query Q_1: "return the French cars, *preferably* at an affordable price with a warranty around 18 months and having a normal power with a medium consumption". Bob will have to invoke S_{11} to

Table 1: Example of Data Web Services

Service	Functionality	Constraints
$S_{11}(\$x, ?y)$	Returns the automakers y made in x	-
$S_{21}(\$x, ?y, ?z, ?t)$	Returns the cars y along with their prices z and warranties t for a given automaker x	z is *cheap*, t is *short*
$S_{22}(\$x, ?y, ?z, ?t)$		z is *accessible*, t is $[12, 24]$
$S_{23}(\$x, ?y, ?z, ?t)$		z is *expensive*, t is *long*
$S_{24}(\$x, ?y, ?z, ?t)$		z is $[9000, 14000]$, t is $[6, 24]$
$S_{31}(\$x, ?y, ?z)$	Returns the power y and the consumption z for a given car x	y is *weak*, z is *small*
$S_{32}(\$x, ?y, ?z)$		y is *ordinary*, z is *approximately* 4
$S_{33}(\$x, ?y, ?z)$		y is *powerful*, z is *high*
$S_{34}(\$x, ?y, ?z)$		y is $[60, 110]$, z is $[3.5, 5.5]$

retrieve the French automakers, he can then invoke one or more of the services $S_{21}, S_{22}, S_{23}, S_{24}$ to retrieve the French cars along with their prices and warranties, finally, he will invoke one or more of the services $S_{31}, S_{32}, S_{33}, S_{34}$ to retrieve the power and the consumption of retrieved cars. This manual process is painful and tedious. It raises the following challenges: (i) how to understand the semantics of the published data Web services to select the relevant ones that can contribute to answering the query at hand; (ii) how to retain the most relevant services (several similar data Web services offer the same functionality but are associated with different constraints) that better satisfy the user's preferences and (iii) how to generate the best k data Web service compositions that satisfy the whole user query.

Contributions. We already tackled the first challenge by proposing a semantic annotation of data Web services and an efficient RDF-based query rewriting approach that generates automatically the data Web service compositions that cover a user query (which does not include any preference constraints) [4]. In this paper, we focus on the second and third challenges. We select the services that cover a part of the query even if their constraints match only partially the user preference constraints. Different constraints inclusion methods are investigated to compute the matching degrees between the services' fuzzy constraints and the fuzzy preferences involved in the query. In order to select the most relevant services, a ranking criteria based on a multicriteria fuzzy dominance relationship is proposed. The selected services are then used to find the top-k service compositions that answer the user query.

Outline. The rest of this paper is organized as follows. In Section 2, we present the background of our paper, i.e., a remainder on fuzzy sets, that constitutes a basis to our work. In Section 3, we formally describe the studied problem. Section 4 describes the ranking approach of data Web services which is mainly based on the fuzzy dominance relationship. Section 5 is devoted to the top-k data Web service compositions generation method. Section 6 presents the global architecture of our Web service composition-based preference query answering and shows some experimental results. We review related work in Section 7. Finally, Section 8 concludes the paper.

2. BACKGROUND ON FUZZY SETS

2.1 Basic Notions

Fuzzy set theory was introduced by Zadeh [31] for modeling classes or sets whose boundaries are not quite defined. For such objects, the transition between full membership and full mismatch is gradual rather than crisp. Typical examples of such fuzzy classes are those described using adjectives of the natural language, such as "cheap", "affordable" and "expensive". Formally, a fuzzy set F on a referential X is characterized by a membership function $\mu_F : X \rightarrow [0, 1]$ where $\mu_F(x)$ denotes the grade of membership of x in F. In particular, $\mu_F(x) = 1$ reflects full membership of x in F, while $\mu_F(x) = 0$ expresses absolute non-membership. When $0 < \mu_F(x) < 1$, one speaks of partial membership. F is normalized if $\exists x \in X, \mu_F(x) = 1$. Two crisp sets are of particular interest when defining a fuzzy set F:

- the core $C(F) = \{x \in X \mid \mu_F(x) = 1\}$, which gathers the *prototypes* of F,
- the support $S(F) = \{x \in X \mid \mu_F(x) > 0\}$, which contains the elements that belong to some extent to F.

In practice, the membership function associated with F is often of a trapezoidal shape. Then, F is expressed by the quadruplet (A, B, a, b) where $C(F) = [A, B]$ and $S(F) = [A - a, B + b]$, see Figure 1. A regular interval [A, B] can be seen as a fuzzy set represented by the quadruplet $(A, B, 0, 0)$.

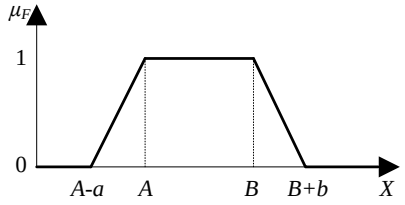

Figure 1: Trapezoidal membership function

Let F and G be two fuzzy sets on the universe (i.e., referential) X, we say that $F \subseteq G$ iff $\forall x \in X, \mu_F(x) \leq \mu_G(x)$. The complement of F, denoted by F^c, is defined by $\mu_{F^c}(x) = 1 - \mu_F(x)$. The cardinality of F is defined as $|F| = \sum_{x \in X} \mu_F(x)$. Furthermore, $F \cap G$ (resp. $F \cup G$) is defined the following way:

- $\mu_{F \cap G} = \top(\mu_F(x), \mu_G(x))$ where $\top$ is a t-norm operator that generalizes the conjunction operation (e.g., $\top(x, y) = min(x, y)$ and $\top(x, y) = x \cdot y$).
- $\mu_{F \cup G} = \bot(\mu_F(x), \mu_G(x)))$ where $\bot$ is a co-norm operator that generalizes the disjunction operation (e.g., $\bot(x, y) = max(x, y)$ and $\bot(x, y) = x + y - x \cdot y$).

As usual, the logical counterparts of the theoretical set operators $\cap$, $\cup$ and complementation operator correspond respectively to the conjunction $\wedge$, disjunction $\vee$ and negation $\neg$. See [13] for more details.

A fuzzy implication is any $[0, 1]^2 \rightarrow [0, 1]$ mapping I satisfying the boundary conditions $I(0, 0) = 1$ and $I(1, x) = x$for

all x in [0, 1]. Moreover, we require I to be decreasing in its first, and increasing in its second component. Two families of fuzzy implications are studied in the fuzzy community (due to their semantic properties and to the fact that their results are similar with the ones of usual implications, material implications, when the arguments are 0 or 1):

- *R-implications*: they are defined by $I(x, y) = sup\{\beta \in [0, 1], \top(x, \beta) \leq y\}$, where $\top$ is a t-norm operator. The two most used R-implications are *Godël* implication ($I_{Gd}(x, y) = 1$ if $x \leq y$, 0 otherwise) and *Goguen* implication ($I_{Go}(x, y) = 1$ if $x \leq y$, y/x otherwise).
- *S-implications*: they are defined by $I(x, y) = \perp(1 - x, y)$, where $\perp$ is a co-norm operator. The two most popular S-implications are *Kleene-Dienes* implication ($I_{Kl}(x, y) = max((1 - x, y))$ and *Lukasiewicz* implication ($I_{Lu}(x, y) = min(1 - x + y, 1)$).

Note that *Lukasiewicz* implication is also an R-implication. For a complete presentation about fuzzy implications, see [13].

2.2 Modeling Preferences

Fuzzy sets provide a suitable tool to express user preferences [12][14]. Fuzzy set-based approach to preferences queries is founded on the use of fuzzy set membership functions that describe the preference profiles of the user on each attribute domain involved in the query.

The user does not specify crisp (Boolean) criteria, but gradual ones like "very cheap", "affordable" and "fairly expensive", whose satisfaction may be regarded as a matter of degree. Individual satisfaction degrees associated with elementary conditions are combined using a panoply of fuzzy set connectives, which may go beyond conjunctive and disjunctive aggregations. Then, the result of a query is no longer a flat set of elements but is a set of discriminated elements according to their global satisfaction w.r.t. the fuzzy conditions appearing in the query. So, a complete pre-order is obtained.

3. SERVICE COMPOSITION-BASED PREFERENCE QUERIES ANSWERING

3.1 Preference Queries

Users express their preference queries over domain ontologies using a slightly modified version of SPARQL query language. Preferences are modeled using fuzzy sets. For instance, query Q_1 given in Section 1 is expressed as follows:

```
SELECT ?n ?pr ?w ?pw ?co
WHERE  {?Au rdf:type AutoMaker  ?Au madeIn 'French'
       ?Au makes ?C  ?C rdf:type Car  ?C hasName ?n
       ?C hasPrice ?pr   ?C hasWarranty ?w
       ?C hasPower ?pw   ?C hasConsumption ?co}
PREFIRING {?pr is 'affordable'  ?w is 'around 18',
           ?pw is 'normal'  ?co is 'medium'}
```

The semantics of all fuzzy terms of Table 1 are available in the URL: `http://vm.liris.cnrs.fr:36880/FuzzyTerms/`.

3.2 Data Services

Assume that data services are partitioned into different service classes. A class $\mathcal{S}_j$ represents services with the same inputs, outputs, and providing the same functionality but different (fuzzy) constraints. A data service S_{ji} of class $\mathcal{S}_j$ is described as a predicate $S_{ji}(\$X_j, ?Y_j)$:-$<\phi_j(X_j, Y_j, Z_j), C_{ji}>$ where:

- X_j and Y_j are the sets of input and output variables of S_{ji}, respectively. Input and output variables are also called distinguished variables. They are prefixed with the symbols "$" and "?", respectively.
- $\phi_j(X_j, Y_j, Z_j)$ represents the functionality of the service. This functionality is described as a semantic relationship between input and output variables. Z_j is the set of existential variables relating X_j and Y_j.
- $C_{ji} = \{C_{ji_1}, ..., C_{ji_n}\}$ is a set of constraints expressed as intervals or fuzzy sets on X_j, Y_j or Z_j variables.

X_j and Y_j variables are defined in the WSDL description of data services. Functionality ϕ_j and constraints C_{ji} of a data service S_{ji} are added to the standard WSDL descriptions in the form of annotations. The annotations are represented in the form of SPARQL queries. For instance, the following SPARQL query illustrates the functionality and service constraints of the data service S_{21}:

```
SELECT ?y ?z ?t
WHERE  {?Au rdf:type AutoMaker  ?Au name $$x  ?Au makes ?C
        ?C rdf:type Car ?C hasName ?y ?C hasPrice ?z
        ?C hasWarranty ?t  ?z is 'cheap'  ?t is 'short'}
```

3.3 Discovering Relevant Services

Let Q be a preference query. We use an RDF query rewriting algorithm [4] to discover the parts of Q that are covered by each service −recall that in the general case services may match only parts (referred to by q_j) of Q. The same part q_j is in general covered by one or more services that constitute a class of relevant services and is designated as class $\mathcal{S}_j$. A service $S_{ji} \in \mathcal{S}_j$ is said to be relevant to a query Q iff the functionality of S_{ji} completely matches the part query q_j and its constraints match completely or partially the preference constraints of q_j.

As preference constraints of a query are expressed in the rich fuzzy sets framework, their matching degrees with data services constraints may differ from one constraints inclusion method (CIM) to another. Each relevant service is then associated with $|M|$ matching degrees (if $M = \{m_1, ..., m_{|M|}\}$ is the set of the used methods). For instance, Table 2 shows the matching degrees between each service S_{ji} from Table 1 and its corresponding component q_j (of the query Q_1). The degrees are computed by applying the following CIMs:
Let $C \equiv x$ is E and $C' \equiv x$ is F be two constraints.

- *Cardinality-based method (CBM)* [30]. $Deg(C \subseteq C') = \frac{|E \cap F|}{|E|} = \frac{\sum_{x \in X} T(\mu_E(x), \mu_F(x))}{\sum_{x \in X} \mu_E(x)}$ where $\top$ is a t-norm operator. The *min* operator is used in our example.
- *Implication-based method (IBM)* [3]. $Deg(C \subseteq C') = min_{x \in X} I(\mu_E(x), \mu_F(x))$ where I stands for a fuzzy implication. The following IBMs are used in our example: Godel (I_{Go}), Lukasiewicz (I_{Lu}) and Kleene-Diennes (I_{Kl}).

The service S_{11} covering the component q_1 does not have a matching degree because there are no constraints imposed by the user on q_1. However, each service covering the component q_2 is associated with four (the number of methods)

degrees. Each matching degree is formulated as a pair of real values within the range $[0, 1]$, where the first and second values are the matching degrees of the constraints *price* and *warranty*, respectively. Similarly, for the matching degrees of the services covering the component q_3, the first and second values represent the inclusion degrees of the constraints *power* and *consumption*, respectively.

Table 2: Matching Degrees between services and fuzzy preference constraints of Q_1

S_{ji}	q_j	CBM	$I_{Go}BM$	$I_{Lu}BM$	$I_{Kl}BM$
S_{11}	q_1	-	-	-	-
S_{21}	q_2	$(1, 0.57)$	$(1, 0)$	$(1, 0)$	$(0.80, 0)$
S_{22}		$(0.89, 1)$	$(0, 1)$	$(0.90, 1)$	$(0.50, 1)$
S_{23}		$(0.20, 0.16)$	$(0, 0)$	$(0, 0)$	$(0, 0)$
S_{24}		$(0.83, 0.88)$	$(0.60, 0.50)$	$(0.60, 0.50)$	$(0.60, 0.50)$
S_{31}	q_3	$(0.50, 0.36)$	$(0, 0)$	$(0, 0)$	$(0, 0)$
S_{32}		$(0.79, 0.75)$	$(0, 0.25)$	$(0.60, 0.50)$	$(0.40, 0.50)$
S_{33}		$(0.21, 0.64)$	$(0, 0)$	$(0, 0)$	$(0, 0)$
S_{34}		$(0.83, 0.85)$	$(0.50, 0.50)$	$(0.50, 0.50)$	$(0.50, 0.50)$

3.4 Problem Formulation

Given a query Q:-$<q_1, \cdots, q_n>$ where each q_j is a subquery (query component). q_j is a tuple $(\overline{q}_j, P_{q_j})$, where $\overline{q}_j$ represents q_j without its preferences P_{q_j}. Given a set of services classes $\mathcal{S} = \{\mathcal{S}_1, ..., \mathcal{S}_n\}$ where a class $\mathcal{S}_j$ regroups the services that are relevant to a query part q_j and a set $M = \{m_1, ..., m_{|M|}\}$ of CIMs to compute the matching degrees between the constraints of relevant services and the user's preference. The problem we are interested in is how to rank data services in each class $\mathcal{S}_j$ to select the most relevant ones and how to rank generated data service compositions to select the top-k ones that answer the query Q.

4. SERVICES/COMPOSITIONS FUZZY DOMINANCE SCORES

4.1 Fuzzy dominance vs Pareto dominance

Services of the same class $\mathcal{S}_j$ have the same functionality, they only differ in terms of constraints, providing thus different matching degrees. Traditional approaches use only one matching criteria to discriminate among relevant services. Most of these approaches aggregate individual matching degrees to compute a global score for each relevant service. One direction is to assign weights to individual matching degrees and, for instance, compute a weighted average of degrees [11]. In so doing, users may not know enough to make trade-offs between different relevancies using numbers (average degrees). Users thus lose the flexibility to select their desired answers by themselves. Computing the skylines from service comes as a natural solution that overcomes this limitation. Skyline computation has received significant consideration in database research [7, 24, 18, 10, 20]. The skyline consists of the set of points which are not dominated by any other point.

Definition 1. (Pareto dominance)
Given two d-dimensional points u and v. We say that u dominates v, denoted by $u \succ v$, iff u is better than or equal to v in all dimensions, and strictly better in at least one dimension, i.e., $\forall i \in [1, d], u_i \geq v_i \wedge \exists k \in [1, d], u_k > v_k$.

Pareto dominance is not always significant to rank-order points that, however, seem comparable from a user point of view. To illustrate this situation, let $u = (u_1, u_2) = (1, 0)$ and $v = (v_1, v_2) = (0.90, 1)$ be two matching degrees. In Pareto order, we have neither $u \succ v$ nor $v \succ u$, i.e. the instances u and v are incomparable. However, one can consider that v is better than u since $v_2 = 1$ is too much higher than $u_2 = 0$, contrariwise $v_1 = 0.90$ is almost close to $u_1 = 1$. This is why it is interesting to fuzzify the dominance relationship to express the extent to which a matching degree (more or less) dominates another one [6]. In line with the general fuzzification dominance approach discussed in [17], we define below a fuzzy dominance relationship that relies on particular membership function of a graded inequality of the type "strongly larger than".

Definition 2. (Fuzzy dominance)
Given two d-dimensional points u and v, we define the fuzzy dominance to express the extent to which u dominates v as:

$$deg(u \succ v) = \frac{\sum_{i=1}^{d} \mu_{\gg}(u_i, v_i)}{d} \quad (1)$$

Where $\mu_{\gg}(u_i, v_i)$ expresses the extent to which u_i is more or less (strongly) greater than v_i. $\mu_{\gg}$ can be seen as a monotone membership function defined as:

$$\mu_{\gg}(x, y) = \left\{ \begin{array}{cc} 0 & if x - y \leq \varepsilon \\ 1 & if x - y \geq \lambda + \varepsilon \\ \frac{x-y-\varepsilon}{\lambda} & otherwise \end{array} \right\} \quad (2)$$

Where $\lambda > 0$, i.e., $\gg$ is more demanding than the idea of "strictly greater". We should also have $\varepsilon \geq 0$ in order to ensure that $\gg$ is a relation that agrees with the idea of 'greater' in the usual sense.

Let us reconsider the previous instances $u = (1, 0)$, $v = (0.90, 1)$. With $\varepsilon = 0$ and $\lambda = 0.2$, we have $deg(u \succ v) = 0.25$ and $deg(v \succ u) = 0.5$. This is more significant than $|u \succ v| = |v \succ u| = 0$ provided by Pareto dominance, where $|u \succ v| = 1$ if $u \succ v$, 0 otherwise. In the following, we use the defined fuzzy dominance relationship to compute scores of services and compositions.

4.2 Associating fuzzy score with a service

It is well known that under a single matching method (mono criteria), the dominance relationship is unambiguous. When multiple CIMs are applied (multi-criteria), resulting in different matching degrees for the same couple of constraints, the dominance relationship becomes uncertain. The model proposed in [21], namely probabilistic skyline overcomes this problem. Contrariwise, Skoutas et al. show in [22, 23] the limitations of the probabilistic skyline to rank Web services and introduce the Pareto dominating score of individual services. We generalize this score to fuzzy dominance and propose the fuzzy dominating score (FDS). An FDS of a service S_{ji} indicates the average extent to which S_{ji} dominates the whole services of its class $\mathcal{S}_j$.

Definition 3. (Fuzzy dominating score of a service)
The fuzzy dominating score (FDS) of a service S_{ji} in its class $\mathcal{S}_j$ is defined as:

$$FDS(S_{ji}) = \frac{1}{(|\mathcal{S}_j| - 1)\,|M|^2} \sum_{h=1}^{|M|} \sum_{\substack{S_{jk} \in \mathcal{S}_j \\ k \neq i}} \sum_{r=1}^{|M|} deg(S_{ji}^h \succ S_{jk}^r) \quad (3)$$

where S^h_{ji} is the matching degree of the service S_{ji} obtained by applying the h^{th} CIM. The term $(|\mathcal{S}_j| - 1)$ is used to normalize the FDS score and make it in the range $[0, 1]$. Table 3 shows the fuzzy dominating scores of the data Web services in our running example.

4.3 Associating fuzzy score with a composition

Different data service compositions can be generated from different $\mathcal{S}_j$ service classes to answer a user query. To rank such generated compositions, we extend the previous FDS definition to service composition and associate each composition with an FDS. The FDS of a composition C is an aggregation of different FDSs of its component services.

Let $C = \{S_{1i_1}, ..., S_{ni_n}\}$ be a composition of n services. Let also $d = d_1 + ... + d_n$ be the number of user preference constraints where d_j is the number of constraints involved in the service S_{ji_j}. The FDS of C is then computed as follows:

$$FDS(C) = \frac{1}{d} \sum_{j=1}^{n} d_j \cdot FDS(S_{ji_j}) \quad (4)$$

It is important to note that not all compositions are valid. A composition C of data Web services is valid if (i) it covers the user query Q, (ii) it contains one and only one service from each service class $\mathcal{S}_j$ and (iii) it is executable. A composition is said to be executable if all input parameters necessary for the invocation of its component services are bound. For more details see [4].

Table 3: Services' scores and services

Services	Class	Score	Top-k
S_{11}	$\mathcal{S}_1$	-	S_{11}
~~S_{21}~~	$\mathcal{S}_2$	0.487	S_{22} S_{24}
S_{22}		0.653	
~~S_{23}~~		0.035	
S_{24}		0.538	
~~S_{31}~~	$\mathcal{S}_3$	0.094	S_{32} S_{34}
S_{32}		0.593	
~~S_{33}~~		0.130	
S_{34}		0.743	

5. TOP-K DATA SERVICE COMPOSITIONS

5.1 An Efficient Generation of Top-k Compositions

A straightforward method to find the top-k data Web service compositions that answer a query is to generate all possible compositions, compute their scores, and return the top-k ones. However, this approach results in a high computational cost, as it needs to generate all possible compositions, whereas, most of them are not in the top-k. In the following, we provide an optimization technique to find the top-k data Web service compositions. This technique allows eliminating relevant services S_{ji} from their classes $\mathcal{S}_j$ before generating the compositions, i.e., services that we are sure that if they are composed with other ones, the obtained compositions are not in the top-k. The idea is: we first compute the score of each service in its class, then only the best services in each class are retained, after that we compose the retained services, finally, we compute the score of the obtained compositions and return the top-k ones. To this end, we introduce the following lemma and theorem.

LEMMA 1. *Let $C = \{S_{1i_1}, \cdots, S_{ni_n}, S\}$ and $C' = \{S_{1i_1}, \cdots, S_{ni_n}, S'\}$ be two similar service compositions that only differ in the services S and S'. Then, the following statement holds: $FDS(S) > FDS(S') \Longrightarrow FDS(C) > FDS(C')$.*

PROOF. Denoting by d' the number of constraints involved in S and S', we have:
$FDS(C) = \frac{1}{d} \sum_{j=1}^{n} d_j \cdot FDS(s_{ji_j}) + \frac{d'}{d} \cdot FDS(S)$ and
$FDS(C') = \frac{1}{d} \sum_{j=1}^{n} d_j \cdot DS_f(s_{ji_j}) + \frac{d'}{d} \cdot FDS(S)$.
Then, $FDS(C) - FDS(C') = \frac{d'}{d}(FDS(S) - FDS(S'))$.
Since $\frac{d'}{d} > 0$ and $FDS(S) - FDS(S) > 0$, we have $FDS(C) > FDS(C')$. □

Lemma 1 indicates that the best services in their classes will generate the best compositions.

THEOREM 1. *Let $C = \{S_{1i_1}, ..., S_{ni_n}\}$ be a service composition and top-$k(\mathcal{S}_j)$ (resp. top-$k(\mathcal{C})$) be the top-k services of the class $\mathcal{S}_j$ (resp. the top-k compositions). Then, $\exists S_{ji_j} \in C; S_{ji_j} \notin$ top-$k(\mathcal{S}_j) \Longrightarrow C \notin$ top-$k(\mathcal{C})$*

PROOF. Assume that $C \in$ top-$k(\mathcal{C}) \wedge \exists S_{ji_j} \in C; S_{ji_j} \notin$ top-$k(\mathcal{S}_j)$. This means that $\exists S'_{ji_1}, \ldots, S'_{ji_k} \in \mathcal{S}_j; FDS(S'_{ji_\ell}) > FDS(S_{ji_j})$. Now, by replacing S_{ji_j} in C with the services $S'_{ji_1}, \ldots, S'_{ji_k}$ we obtain k compositions $C_1, ..., C_k$ such as $FDS(C_i) > FDS(C)$ according to the Lemma 1. This contradicts our hypothesis. Hence, $C \notin$ top-$k(\mathcal{C})$ □

Theorem 1 means that the top-k sets of the different service classes are sufficient to compute the top-k data Web service compositions that answer the considered query.

The fourth column of Table 3 shows the top-k (where $k = 2$) data services in each service class using the FDS scores. Thus, relevant data services that are not in the top-k of their classes are eliminated. They are crossed out in Table 3. The other data services are retained. The top-k data service compositions are generated from the different top-k $\mathcal{S}_j$ classes. Table 4 shows the possible compositions along with their fuzzy dominating scores and the top-k ones of our example.

Table 4: Compositions' scores and top-k ones

Compositions	Score	Top-k
$C_1 = \{S_{11}, S_{22}, S_{32}\}$	0.623	C_2 C_4
$C_2 = \{S_{11}, S_{22}, S_{34}\}$	0.698	
$C_3 = \{S_{11}, S_{24}, S_{32}\}$	0.566	
$C_4 = \{S_{11}, S_{24}, S_{34}\}$	0.640	

5.2 Top-k Data Service Compositions Algorithm

The algorithm, hereafter referred as $TKDSC$, computes the top-k data service compositions according to the fuzzy dominatind scores. The algorithm proceeds as following.

Step.1 *computing the matching degrees (lines 1-13).* Each service class whose services cover a query component is added to the list of relevant classes. If its services touch the query's user preferences, we compute its different matching degrees according to the number of methods.

Step.2 *eliminating less relevant services (lines 14-23).* For each class whose services do not touch the user preferences, we select randomly k services since they are all equal

Algorithm 1 TKDSC

Require: Q a preference query; $\mathcal{S} = \{\mathcal{S}_1, ..., \mathcal{S}_n\}$ a set of service classes; $M = \{m_1, ..., m_{|M|}\}$ a set of methods; $k \in \mathbb{N}$; $\varepsilon > 0$; $\lambda > 0$;
Ensure: the top-k service compositions

```
1: for all S_j in S do
2:   S ← random(S_j, 1);
3:   if ∃q_j ∈ Q; cover(S, q_j) then
4:     R ← R ∪ S_j;
5:     if P_{q_j} ≠ ∅ then
6:       for all S_ji in S_j do
7:         for all m in M do
8:           ComputeMatchingDegree(C_ji, P_{q_j}, m);
9:         end for
10:      end for
11:    end if
12:  end if
13: end for
14: for all S_j in R do
15:   if P_{q_j} = ∅ then
16:     top-k.S_j ← random(S_j, k);
17:   else
18:     for all S_ji in S_j do
19:       ComputeServiceScore(S_ji);
20:     end for
21:     top-k.S_j ← top(k, S_j);
22:   end if
23: end for
24: C ← ComposeServices(top-k.S_{j_1}, ..., top-k.S_{j_m});
25: for all C in C do
26:   ComputeCompositionScore(C);
27: end for
28: return top(k, C);
```

with respect to user preferences. Otherwise (i.e., its services touch the user preferences), we first compute the score of its services and then retain only the top-k ones.

Step.3 *returning top-k compositions (lines 24-28).* First, we compose services from only the retained ones, i.e., the top-k in each class. Then we compute the score of generated compositions and finally we provide the user with the top-k ones.

6. SYSTEM ARCHITECTURE AND EXPERIMENTAL EVALUATION

6.1 System Architecture

Figure 2 presents our implemented top-k data service compositions system. The system consists of the three major modules: *Annotation Module*, *Interactive Query Formulation Module* and *Top-k Service Compositions Module*.

The *Annotation Module* allows service providers to annotate WSDL description files of services with fuzzy sets to represent linguistic terms and with SPARQL queries to represent the functionality and constraints of services. This annotation is implemented by adding a new element called "rdfQuery" to the XML Schema of WSDL as in WSDL-S approach. The annotated WSDL files are then published on a service registry. The ontology manager uses Jena API to manage domain ontology, i.e., to add/delete concepts.

The *Interactive Query Formulation Module* provides users

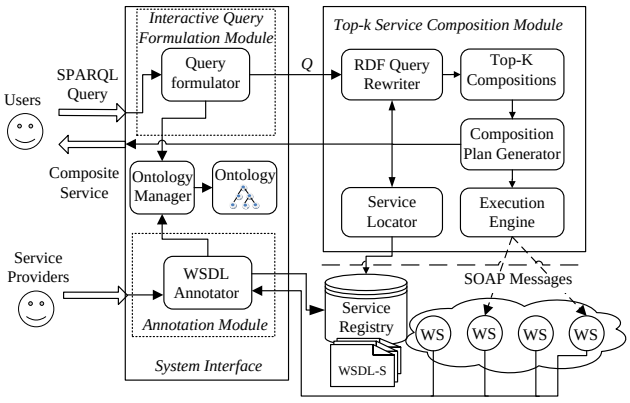

Figure 2: Data Service Composition Architecture

with a GUI implemented with Java Swing to interactively formulate their queries over a domain ontology. Users are not required to know any specific ontology query languages to express their queries.

The *Top-k Service Compositions Module* consists of five components. The *RDF Query Rewriter* implements an efficient RDF query rewriting algorithm (*RDF Query Rewriter*) to identify the relevant services that match (some parts of) a user query. For that purpose, it exploits the functionalities in the service description files. The *Service Locator* feeds the *Query Rewriter* with services that most likely match a given query. The *Top-K Compositions* component computes (i) the matching degrees of relevant services, (ii) the fuzzy dominating scores of relevant services, (iii) the top-k services of each relevant service class and (iv) the fuzzy compositions scores to return the top-k compositions. The top-k compositions are then translated by the *composition plan generator* into execution plans expressed in the XPDL language. They are executed by a workflow execution engine; we use the Sarasvati execution engine from Google.

6.2 Experimental Evaluation

Our objective is to prove the efficiency and the scalability of our proposed top-k data Web service composition. For this purpose, we implemented a Web service generator. The generator takes as input a set of (real-life) model services (each representing a class of services) and their associated fuzzy constraints and produces for each model service a set of synthetic Web services and their associated synthetic fuzzy constraints. In the experiments we evaluated the effects of the following parameters: (i) the number of services per class, (ii) the service classes number, (iii) the number of fuzzy constraints per class, (iv) the number of matching methods and (v) the parameter k. The default values of these parameters are : 400, 4, 4, 4, 5, respectively.

The algorithm (i.e., TKDSC is implemented in Java. The experiments were conducted on a 2.00 GHz Intel dual core CPU and 2 GB of RAM, running Windows. The results of the experiments are presented in Figure 3.

6.2.1 Performance vs. number of services per class

We measured the average execution time required to solve the top-k service compositions problem as the number of services per class increases, varying the number of services per class from 100 to 1000. The results of this experiment

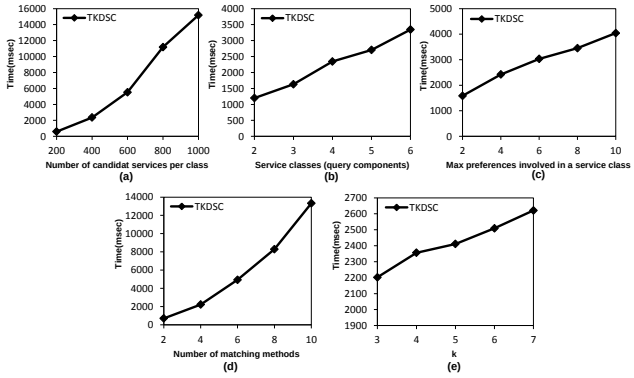

Figure 3: Performance results

are presented in Figure 3 (plot-a). The results show that our framework can handle hundreds of services per class in a reasonable time.

6.2.2 Performance vs number of classes

We measured the average execution time required to solve the top-k service compositions problem as the number of service classes increases. We varied the classes number from 1 to 6. The results of this experiment in Figure 3 (plot-b) show that the execution time is proportional to the number of service classes.

6.2.3 Performance vs number of constraints per service

We varied the fuzzy constraints number from 1 to 10 and measured the average execution time required to compute the top-k service compositions. Figure 3 (plot-c) shows the time required to compute the top-k service compositions.

6.2.4 Performance vs. number of matching methods

We varied the number of matching methods from 1 to 10. We measured the average execution time required to compute the top-k service compositions. The results of this experiment are shown in Figure 3 (plot-d).

6.2.5 Performance vs. k

We measured the average execution time required to compute the top-k service compositions as the value of k increases. We varied the value of k from 3 to 5. The results of this experiment in Figure 3 (plot-e) show that the execution time increases as the value of k increases.

7. RELATED WORK

Preferences in Web service selection/composition have received much attention in the service computing community during the last years. Taking user preferences into account allows to rank candidate services/compositions and return only the best ones to the user. Hereafter, we review some works for ranking and selecting Web services.

ServiceTrust [15] calculates reputations of services from users. It introduces transactional trust to detect QoS abuse, where malicious services gain reputation from small transactions and cheat at large ones. However, *ServiceTrust* models transactions as binary events (success or failure) and combines reports from users without taking their preferences into account. In [26], the authors use a qualitative graphical representation of preference, CP-nets, to deal with services selection in terms of user preferences. This approach can reason about a user's incomplete and constrained preference. In [19], a method to rank semantic web services is proposed. It is based on computing the matching degree between a set of requested NFPs (Non-Functional Properties) and a set of NFPs offered by the discovered Web services. NFPs cover QoS aspects, but also other business-related properties such as pricing and insurance. Semantic annotations are used for describing NFPs and the ranking process is achieved by using some automatic reasoning techniques that exploit the annotations. However, the problem of composition is not addressed in these works.

Agarwal and Lamparter [1] propose an automated Web service selection approach for composition. Web service combinations can be compared and ranked according to user preferences. Preferences are modeled as a set of fuzzy IF-THEN rules. The IF part contains fuzzy descriptions of the various properties of a service (i.e., a concrete Web service composition) and the THEN part is one of the fuzzy characterizations of a special concept called *Rank*. A fuzzy rule describes which combination of attribute values a user is willing to accept to which degree, where attribute values and degrees of acceptance are defined in a fuzzy way. ServiceRank [27] considers the QoS aspects as well as the social perspectives of services. Services that have good QoS and are frequently invoked by others are more trusted by the community and will be assigned high ranks. In [25], the authors propose a system for conducting qualitative Web service selection in the presence of incomplete or conflicting user preferences. The paradigm of CP-nets is used to model user preferences. The system utilizes the history of users to amend the preferences of active users, thus improving the results of service selection.

The work most related to ours is [22, 23], where the authors consider dominance relationships between Web services based on their degrees of match to a given request in order to rank available services. Distinct scores based on the notion of dominance are defined for assessing when a service is objectively interesting. However, that work only considers selection of single services, without dealing with the problem of composition nor the user preferences.

Recent approaches, focus on computing the skyline from Web services. All these approaches focus on selecting Web services based on QoS parameters. The work in [2] focuses on the selection of skyline services for QoS based Web service composition. A method for determining which QoS levels of a service should be improved so that it is not dominated by other services is also discussed. In [29], the authors propose a skyline computation approach for service selection. The resulting skyline, called multi-service skyline, enables services users to optimally and efficiently access sets of service as an integrated service package. In the robust work [28], Yu and Bouguettaya address the problem of uncertainty inherent in QoS and compute the skylines from service providers. A service skyline can be regarded as a set of service providers that are not dominated by others in terms of QoS aspects that interest all users. To this end, a concept called p-dominant service skyline is defined. A provider S belongs to the p-dominant skyline if the probability that S is dominated by any other provider is less than p. The authors provide also a discussion about the interest of p-dominant skyline w.r.t.

the notion of p-skyline proposed in [21]. In [5], we propose a new concept called α-dominant service skyline based on a fuzzy dominance relationship to address the majors issues of the traditional service skyline, i.e., privileging Web services with a bad compromise between QoS attributes and not allowing users to control the size of the returned set of Web services. However, these works do not take user preferences into account and except for [2] the problem of composition is not addressed.

8. CONCLUSION

In this paper, we addressed the problem of top-k retrieval of data Web service compositions to answer fuzzy preference queries under different matching methods. We presented a suitable ranking criteria based on a fuzzification of Pareto dominance and developed a suitable algorithm for computing the top-k data Web service compositions. Our experimental evaluation shows that our approach can retrieve the top-k data Web service compositions in a reasonable time. In the future, we plan to use a user study to evaluate the quality of the results and combine this work with QoS aspect.

9. REFERENCES

[1] S. Agarwal and S. Lamparter. User preference based automated selection of web service compositions. In K. V. A. S. M. Z. C. Bussler, editor, *ICSOC Workshop on Dynamic Web Processes*, pages 1–12, Amsterdam, Netherlands, Dezember 2005. IBM.

[2] M. Alrifai, D. Skoutas, and T. Risse. Selecting skyline services for qos-based web service composition. In *WWW*, pages 11–20, 2010.

[3] W. Bandler and L. Kohout. Fuzzy power sets and fuzzy implication operators. *Fuzzy Sets and Systems*, 9:149–183, 1983.

[4] M. Barhamgi, D. Benslimane, and B. Medjahed. A query rewriting approach for web service composition. *IEEE T. Services Computing*, 3(3):206–222, 2010.

[5] K. Benouaret, D. Benslimane, and A. Hadjali. On the use of fuzzy dominance for computing service skyline based on qos. In *ICWS*, page to appear, 2011.

[6] K. Benouaret, D. Benslimane, and A. Hadjali. Top-k service compositions: A fuzzy set-based approach. In *SAC*, pages 1033–1038, 2011.

[7] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[8] M. J. Carey. Data delivery in a service-oriented world: the bea aqualogic data services platform. In *SIGMOD Conference*, pages 695–705, 2006.

[9] M. J. Carey. Declarative data services: This is your data on soa. In *SOCA*, page 4, 2007.

[10] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, pages 717–816, 2003.

[11] X. Dong, A. Y. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Simlarity search for web services. In *VLDB*, pages 372–383, 2004.

[12] D. Dubois and H. Prade. Using fuzzy sets in database systems: Why and how? In *FQAS*, pages 89–103, 1996.

[13] D. Dubois and H. Prade, editors. *Fundamentals of Fuzzy Sets* . The Handbooks of Fuzzy Sets Series. Kluwer, Boston, Mass., 2000.

[14] A. HadjAli, S. Kaci, and H. Prade. Database preferences queries - a possibilistic logic approach with symbolic priorities. In *FoIKS*, pages 291–310, 2008.

[15] Q. He, J. Yan, H. Jin, and Y. Yang. Servicetrust: Supporting reputation-oriented service selection. In *ICSOC/ServiceWave*, pages 269–284, 2009.

[16] A. Jhingran. Enterprise information mashups: integrating information, simply. In *In Proccedings of the 2006 VLDB*, pages 3–4, 2006.

[17] M. Köppen, R. Vicente-Garcia, and B. Nickolay. Fuzzy-pareto-dominance and its application in evolutionary multi-objective optimization. In *EMO*, pages 399–412, 2005.

[18] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB*, pages 275–286, 2002.

[19] M. Palmonari, M. Comerio, and F. D. Paoli. Effective and flexible nfp-based ranking of web services. In *ICSOC/ServiceWave*, pages 546–560, 2009.

[20] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD Conference*, pages 467–478, 2003.

[21] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, pages 15–26, 2007.

[22] D. Skoutas, D. Sacharidis, A. Simitsis, V. Kantere, and T. K. Sellis. Top- dominant web services under multi-criteria matching. In *EDBT*, pages 898–909, 2009.

[23] D. Skoutas, D. Sacharidis, A. Simitsis, and T. K. Sellis. Ranking and clustering web services using multicriteria dominance relationships. *IEEE T. Services Computing*, 3(3):163–177, 2010.

[24] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB*, pages 301–310, 2001.

[25] H. Wang, S. Shao, X. Zhou, C. Wan, and A. Bouguettaya. Web service selection with incomplete or inconsistent user preferences. In *ICSOC/ServiceWave*, pages 83–98, 2009.

[26] H. Wang, J. Xu, and P. Li. Incomplete preference-driven web service selection. In *IEEE SCC (1)*, pages 75–82, 2008.

[27] Q. Wu, A. Iyengar, R. Subramanian, I. Rouvellou, I. Silva-Lepe, and T. A. Mikalsen. Combining quality of service and social information for ranking services. In *ICSOC/ServiceWave*, pages 561–575, 2009.

[28] Q. Yu and A. Bouguettaya. Computing service skyline from uncertain qows. *IEEE T. Services Computing*, 3(1):16–29, 2010.

[29] Q. Yu and A. Bouguettaya. Computing service skylines over sets of services. In *ICWS*, pages 481–488, 2010.

[30] L. A. Zadeh. A computational approach to fuzzy quantifiers in natural languages. *Computer Mathematics with Applications*, 9:149–183, 1965.

[31] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.