Advanced techniques for Web service query optimization

Karim Benouaret



Numéro d'ordre: 177-2012 Année 2012

Université Claude Bernard Lyon 1

Laboratoire d'InfoRmatique en Image et Systèmes d'information

École Doctorale Informatique et Mathématiques de Lyon

Thèse de l'Université de Lyon

Présentée en vue d'obtenir le grade de Docteur,
spécialité Informatique

par

Karim Benouaret

Advanced Techniques for Web Service Query Optimization

Thèse soutenue le 09.10.2012 devant le jury composé de:

Rapporteurs:	Mourad Chabane Oussalah	–	Professeur à l'Université de Nantes
	Anne Laurent	–	Professeur à l'Université Montpellier 2
Examinateurs:	Claude Godart	–	Professeur à l'Université de Lorraine
	Juliette Dibie-Barthélemy	–	Maitre de Conférences à l'INRA de Paris
Directeur:	Djamal Benslimane	–	Professeur à l'Université Lyon 1
Co-directeur:	Allel Hadjali	–	Professeur à l'ENSMA – Poitiers

To my mother, Zahia.

To my father, Ahmed.

To my sister, Sonia.

To my brother, Idir.

Acknowledgments

It would not have been possible to write this dissertation without the help and support of the kind people around me, to only some of whom it is possible to give particular mention here.

First, I wish to express my deep gratitude to my advisor Prof. Djamal Benslimane, for his excellent guidance, caring, patience, and providing me with an excellent atmosphere for doing research. I thank him for his continuous support and encouragement on both my research and other matters of life during my PhD study. As an advisor, he taught me practices and skills that will benefit my future academic career. I am also grateful to my co-advisor Prof. Allel Hadjali for his consistent supports and encouragements. His advice and cooperation are very helpful, despite the distance. It has been a great fortune for me to work under the supervision of Prof. Djamal Benslimane and Prof. Allel Hadjali.

I am very thankful to Dr. Dimitris Sacharidis. I thank him for the knowledge and skills he imparted through our collaboration. He was always willing to help and give his best suggestions. Working with him is so enjoyable. I also appreciate his help to improve the quality of my dissertation.

Many thanks go to Dr. Mahmoud Barhamgi. I thank him for his collaboration. We spent countless hours together discussing research and other fun part about life.

I would also like to thank my parents, my sister and my brother for their continuous moral support and encouragement with their best wishes. Their love accompanies me wherever I go.

Finally, I would like to thank my friends at LIRIS Laboratory and Lyon 1 University. I would have been lonely without them.

Abstract: As we move from a Web of data to a Web of services, enhancing the capabilities of the current Web search engines with effective and efficient techniques for Web services retrieval and selection becomes an important issue.

In this dissertation, we present a framework that identifies the top-k Web service compositions according to the user fuzzy preferences based on a fuzzification of the Pareto dominance relationship. We also provide a method to improve the diversity of the top-k compositions. An efficient algorithm is proposed for each method. We evaluate our approach through a set of thorough experiments. After that, we consider the problem of Web service selection under multiple users preferences. We introduce a novel concept called majority service skyline for this problem based on the majority rule. This allows users to make a "democratic" decision on which Web services are the most appropriate. We develop a suitable algorithm for computing the majority service skyline. We conduct a set of thorough experiments to evaluate the effectiveness of the majority service skyline and the efficiency of our algorithm. We then propose the notion of α-dominant service skyline based on a fuzzification of Pareto dominance relationship, which allows the inclusion of Web services with a good compromise between QoS parameters, and the exclusion of Web services with a bad compromise between QoS parameters. We develop an efficient algorithm based on R-Tree index structure for computing efficiently the α-dominant service skyline. We evaluate the effectiveness of the α-dominant service skyline and the efficiency of the algorithm through a set of experiments. Finally, we consider the uncertainty of the QoS delivered by Web services. We model each uncertain QoS attribute using a possibility distribution, and we introduce the notion of *pos*-dominant service skyline and the notion of *nec*-dominant service skyline that facilitates users to select their desired Web services with the presence of uncertainty in their QoS. We then develop appropriate algorithms to efficiently compute both the *pos*-dominant service skyline and *nec*-dominant service skyline. We conduct extensive sets of experiments to evaluate the proposed service skyline extensions and algorithms.

Keywords: skyline, top-k, preferences, service selection, QoS, service composition

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Contents

Over the last decade, the Web has undergone a major transformation, changing from a Web of data to a Web of services. This essentially allows organizations across all spectra to offer their services and conduct their daily life. Web services are self-describing, self-contained, modular software applications and are designed to perform a specific task. Typical examples include services returning information to the user, such as news or weather forecast services, or services altering the world state, such as on-line booking or shopping services.

Nowadays, Web services are emerging to provide a systematic and extensible framework for application-to-application interaction built on the top of existing Web protocols and based on open XML standards. Major industry players took a lead to set up crucial standards. This has greatly facilitated the adoption and deployment of Web services [Lan03]. Three key XML-based standards have been defined to support the Web services framework [CDK+02]: *(i)* the Simple Object Access Protocol (SOAP), which enables communication among Web services; *(ii)* the Web Services Description Language (WSDL), which provides a formal, computer-readable description of Web services; and *(iii)* the Universal Description, Discovery, and Integration (UDDI) directory, which is a registry of Web service descriptions.

While individual Web services usually fulfill the users' needs, in some cases, users need to compose different Web services to achieve a more complex task that cannot

be fulfilled by an individual Web service. Web service composition is a powerful solution for building value-added services on top of existing ones [Sin01, MBE03]. Thus, Web service composition is a crucial aspect of Web services technology, which gives us the opportunity to select new Web services and best suits our needs.

1.1 Research Statement

Consequently, it becomes apparent that the Web services paradigm rapidly gains popularity constituting an integral part of many real-world applications. For this purpose, several techniques for discovering Web services have been recently proposed; e.g., keyword search and semantic search paradigms. However, as Web services and service providers proliferate, there will be a large number of candidate − most likely competing − Web services for fulfilling a desired task. According to [AMM08], there has been a more than 130% growth in the number of published Web services in the period from October 2006 to October 2007. In addition, the statistics published by the Web services search engine Seekda![1] indicate an exponential increase in the number of Web services over the last 72 months. Therefore, to select a relevant Web service, users need to go through several trial-run processes. This would be very painstaking, and the selected Web service is not necessarily among the most interesting ones. Hence, enhancing the capabilities of the current Web search engines with effective and efficient techniques for identifying and selecting the most appropriate Web services or Web service compositions becomes an important issue.

1.1.1 Web Service Query Optimization

The purpose of Web service query optimization is to select optimal Web services – among the discovered ones – since it is common that the result of the service discovery contains a large number of Web services. Even for a composite Web service consisting of many atomic Web services, the selection issue still needs to be addressed as multiple Web services may be available for an atomic Web service. User preferences play a key role during the selection process. Taking user preferences into account allows to return Web services that best satisfy the user requirements. In addition, as the number of Web services with similar functionality is expected to

[1] http://webservices.seekda.com/

be very large, it is crucial to select the best Web services – among the functionally similar ones – based on quality of service (QoS), i.e., preferences are expressed on the QoS parameters of Web services (e.g., price, response time, etc.) instead of the data they manipulate.

The objective of this research is to devise advanced techniques for Web service query optimization. We focus on giving users the flexibility to find the most appropriate Web services or Web service compositions. This will serve as a key block for building tomorrow's Web service search engines.

1.1.2 Research Requirements

We summarize the requirements that need to be dealt with when devising advanced techniques for Web service query optimization as follows:

- $\mathcal{R}_1$: *User preferences aware Web service query optimization* – Web service composition is a powerful means to answer users' complex queries. Due to the proliferation of Web services, selecting Web services from the massive candidates plays a crucial role in the Web service composition world since a large number of Web services may be used to answer the same query. It is thus important to set up an effective framework that would identify and retrieve the most relevant Web services, and return the best Web service compositions according to the user preferences.

- $\mathcal{R}_2$: *Web service query optimization for multiple users preferences* – In many practical situations, multiple users with different – possibly conflicting – preferences need to make a group decision. For example, members of a family who want to buy a car, or a group of friends who want to rent an apartment for the holidays. However, this problem is not taken into account by the current Web service optimization approaches. It is thus interesting to devise optimization strategies for finding the most relevant Web services with respect to all users.

- $\mathcal{R}_3$: *QoS aware Web service query optimization* – The exploding number of functionally similar Web services has led to a new challenge of selecting the most relevant services using QoS aspects. Traditionally, the relevance of a Web service is determined by computing an overall score that aggregates individual QoS values, where users are required to assign weights over QoS attributes.

Users thus lose the flexibility to select their desired Web services. Computing the skyline comes as a popular solution that overcomes this limitation. The skyline consists of the set of Web services that are not dominated by any other one. A Web service s_i dominates another Web service s_j if and only if s_i is better than or equal to s_j in all QoS attributes, and strictly better in at least one QoS attribute. However, the skyline often privileges Web services with a bad compromise between different QoS attributes, i.e., Web services with some very good and very bad QoS values, while users prefer Web services with a good compromise between QoS attributes, i.e., Web services that are (moderately) good in all QoS values. Therefore, there is a need to provide a framework that allows users to select Web services with a good compromise between different QoS attributes in a flexible way.

- $\mathcal{R}_4$: *Web service query optimization over uncertain QoS* – Current QoS-based Web service selection approaches assume that the QoS does not change over time. Whereas, the QoS values may not precisely reflect the actual performances of Web services due to the dynamic Web service environment. For example, the response time may vary with the quality of the network. In addition, Web service providers can still not supply according to their betrothed QoS because of intentional deception. Therefore, the QoS delivered by Web services is uncertain. Taking into account the uncertainty of QoS during the selection process is thus an important issue.

1.1.3 Key Contributions

We address the above-mentioned requirements by providing optimization strategies to enable users to select the most appropriate Web services or Web service compositions in a flexible way. More specifically, our major contributions are summarized as follows:

- $\mathcal{C}_1$: *Top-k Web service compositions with fuzzy preferences* – We present an approach to automatically compose Web services while taking into account the user preferences. User preferences are expressed in a fuzzy linguistic way. They are modeled using fuzzy sets then incorporated into the composition query. We use an efficient query rewriting algorithm to determine the relevant

Web services that may be used to answer the composition query. The (fuzzy) constraints of the relevant Web services are then matched to those of the query to determine their matching degrees using a set of matching methods. We rank-order Web services using a methodology based on a fuzzification of Pareto dominance relationship, then compute the top-k Web service compositions. We propose also a method to improve the diversity of returned compositions while maintaining as possible the compositions with the highest scores. As the problem of Web service composition is known to be NP-hard, we develop for each method a suitable algorithm that prunes the search space. We evaluate our approach through a set of thorough experiments.

- $\mathcal{C}_2$: *Majority-rule-based Web service selection* – We introduce a novel concept called majority service skyline based on the majority rule. This allows users to make a "democratic" decision on which Web services are the most appropriate. We then developed an efficient algorithm for computing the majority service skyline. We conduct a set of thorough experiments to evaluate the effectiveness of the majority service skyline and the efficiency of the proposed algorithm.

- $\mathcal{C}_3$: *Computing skyline Web services using fuzzy dominance* – We propose a skyline variant called α-dominant service skyline based on a fuzzification of Pareto dominance relationship. The α-dominant service skyline allows the inclusion of Web services with a good compromise between QoS parameters, and the exclusion of Web services with a bad compromise between QoS parameters. It thus provides users with the most relevant Web services. The α-dominant service skyline also gives users the flexibility to control the size of the returned Web services. We then develop an efficient algorithm based on R-Tree index structure for computing the α-dominant service skyline. We evaluate the effectiveness of the α-dominant service skyline and the efficiency of the algorithm through a set of experiments.

- $\mathcal{C}_4$: *Selecting skyline Web services from uncertain QoS* – We leverage possibility theory, and model each uncertain QoS attribute of a Web service using a possibility distribution. We then introduce the notion of *pos*-dominant service skyline and the notion of *nec*-dominant service skyline that facilitate users to select their desired Web services with the presence of uncertainty in their

QoS. We then develop appropriate algorithms to efficiently compute both the *pos*-dominant service skyline and *nec*-dominant service skyline. We evaluate our approach through a set of experiments.

Table 1.1: Mapping between Research Requirements and our Contributions

Research requirement	Contribution	Chapter
$\mathcal{R}_1$	$\mathcal{C}_1$	Chapter 3
$\mathcal{R}_2$	$\mathcal{C}_2$	Chapter 4
$\mathcal{R}_3$	$\mathcal{C}_3$	Chapter 5
$\mathcal{R}_4$	$\mathcal{C}_4$	Chapter 6

Table 1.1 shows the mapping between the mentioned research requirements and our contributions, and lists the chapters that cover the corresponding contributions.

1.2 Dissertation Organization

The rest of this dissertation is organized as follows.

In Chapter 2, we provide the necessary background, that we feel is needed to understand the content of this dissertation. First, we present the key concepts around the Web service technology. We then concentrate specifically on the area of preferences. Finally, we introduce the reader to fuzzy sets and possibility theory.

In Chapter 3, we present a framework that identifies the top-k Web service compositions according to the user fuzzy preferences. A fuzzy dominance relationship is proposed to better rank the results. We propose also a method to improve the diversity of the top-k compositions. An efficient algorithm is proposed for each method. We also conduct a set of experiments to evaluate the effectiveness of our methods and the scalability of our algorithms.

In Chapter 4, we introduce a novel concept called majority service skyline based on the majority rule to allow users to make a "democratic" decision on which Web services are the most appropriate. We then develop an efficient algorithm to compute the majority service skyline. This chapter also presents a set of experiments to show the effectiveness of the majority service skyline and the efficiency of our algorithm.

In Chapter 5, we present a new skyline variant called α-dominant service skyline based on a fuzzification of Pareto dominance. The α-dominant service skyline provides users with Web service with a good compromise between QoS parameters, and gives them the flexibility to control the size of the returned Web services. An efficient algorithm is developed to compute efficiently the α-dominant service skyline. We also evaluate the effectiveness of the proposed concept and the efficiency of the algorithm.

In Chapter 6, we present an approach to deal with QoS pervaded with uncertainty. We model each uncertain QoS attribute using a possibility distribution, and introduce two skyline extensions called *pos*-dominant service skyline and the *nec*-dominant service skyline. These skyline extensions facilitate users to select their desired Web services with the presence of uncertainty in their QoS. We then develop appropriate algorithms to efficiently compute the skyline extensions. We also evaluate our approach through a set of experiments.

In Chapter 7, we review the related work that are most related to our research. This aims to position our work with respect to existing ones.

In Chapter 8, we provide concluding remarks and discuss some possible directions for future research.

Chapter 2

Background

Contents

In this chapter, we first present the key concepts behind Web service technology in Section 2.1. We then provide some basic notions around preferences in Section 2.2, while, we focus on both fuzzy sets and possibility theory in Section 2.3 and Section 2.4, respectively. Finally, Section 2.5 concludes this chapter.

2.1 Overview of Web Services

Various software architectures and technologies have been proposed over the last years for easing the development and deployment of distributed systems; e.g., middleware for distributed objects [Emm00]. However, the generalization of the Internet and the diversification of networked devices have led to the definition of a new computing paradigm: the Service-Oriented Architecture (SOA), which allows developing software as a service delivered and consumed on demand [PG03, EL04]. The use of Web service technology allows applications at various locations on the World Wide Web to be interconnected and integrated in a loosely-coupled manner as if they were parts of a single, large information technology system.

2.1.1 Web Services

A variety of definitions about Web services are given in the literature. However, that proposed by the Word Wide Web Consortium (W3C[2]) is considered as reference: "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."[3]

This definition highlights the major technological and business benefits of Web services, namely:

- Interoperability – This is the most important benefit of Web services. Web services typically work outside of private networks, offering developers a non-

[2] http://www.w3.org/

[3] http://www.w3.org/TR/ws-arch/

proprietary route to their solutions. Web services developed are likely, therefore, to have a longer life-span, offering better return on investment of the developed Web service. Web services also let developers use their preferred programming languages. In addition, thanks to the use of standards-based communications methods, Web services are virtually platform-independent.

- Usability – Web services allow the business logic of many different systems to be exposed over the Web. This gives your applications the freedom to chose the Web services that they need. Instead of re-inventing the wheel for each client, you need only include additional application-specific business logic on the client-side. This allows you to develop services and/or client-side code using the languages and tools that you want.

- Reusability – Web services provide not a component-based model of application development, but the closest thing possible to zero-coding deployment of such Web services. This makes it easy to reuse Web service components as appropriate in other Web services. It also makes it easy to deploy legacy code as a Web service.

- Deployability – Web services are deployed over standard Internet technologies. This makes it possible to deploy Web services even over the fire wall to servers running on the Internet on the other side of the globe. Also thanks to the use of proven community standards, underlying security is already built-in.

2.1.2 Web Service Model

The Web service model is based upon interactions between three types of participants including service provider, service registry and service client. Interactions involve three basic operations: service publishing, finding and binding. Participants and operations act upon the Web service artifacts encompassing the service implementation and description. Figure 2.1 shows the different participants and the interactions among them.

In a typical scenario, a service provider provides a network-accessible software module, i.e., an implementation of a Web service, defines a service description for the Web service and publishes it to a service registry so that the service client can find it. The service description contains information such as the inputs/outputs of the

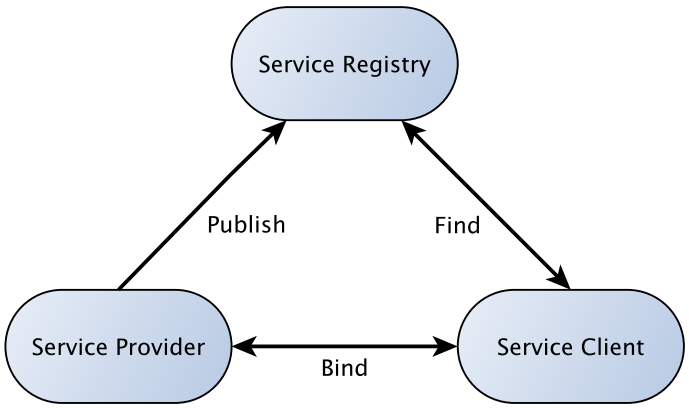


Figure 2.1: The Web Service Model

Web service, the address where the service is located and QoS. The service client queries the service registry for a certain type of service and retrieves the service description. Then it uses the information in the service description to bind with the service provider and invoke the Web service implementation.

2.1.3 Web Service Standards

Standards are key enablers of Web services [CDK+02, VN02]. The service model from above is realized via the following XML-based standards:

- Simple Object Access Protocol (SOAP[4]) – SOAP is a protocol specification for exchanging structured information in the implementation of Web services in computer networks. It relies on XML for its message format, and usually relies on other application layer protocols, most notably HTTP, for message negotiation and transmission.

- Web Services Description Language (WSDL[5]) – WSDL is an XML-based language that is used for describing the functionality offered by a Web service. A WSDL description of a Web service (also referred to as a WSDL file) provides a machine-readable description of how the service can be called, what parameters it expects, and what data structures it returns.

[4] http://www.w3.org/TR/soap12
[5] http://www.w3.org/TR/wsdl

- Universal Description, Discovery and Integration (UDDI[6]) – UDDI is a platform-independent, XML-based registry by which businesses worldwide can list themselves on the Internet, and a mechanism to register and locate web service applications. UDDI is an open industry initiative, sponsored by the Organization for the Advancement of Structured Information Standards (OASIS), for enabling businesses to publish service listings and discover each other, and to define how the services or software applications interact over the Internet.

2.2 Preferences

The handling of user preferences is becoming an increasingly important issue in present-day information systems [Cho03]. Motivations for such a concern are manifold [HKP11]. First, it has appeared to be desirable to offer more expressive query languages which can be more faithful to what a user intends to say. Second, the introduction of preferences in queries provides a basis for rank-ordering the retrieved items, which is especially valuable in case of large sets of items satisfying a query. Third, on the contrary, a classical query may also have an empty set of answers, while a relaxed (and thus less restrictive) version of the query might be matched by items in the database.

2.2.1 Preference Representation

Preference representation approaches can be categorized as follows [SKP11]:

- Formulation – Preferences are formulated *(i)* quantitatively, i.e., specified using functions that associate a numerical score with each tuple. For example, "my interest in sport cars is 0.6, in passenger car is 0.3 and in vans is 0.1", which implies that sport cars are more preferable than passenger cars, which in turn are more preferable than vans; or *(ii)* qualitatively, i.e., defined as binary relations between two tuples. For example, "I like sport cars better than passenger cars or vans", which implies that sport cars are preferred over passenger cars and vans, but passenger cars and vans are indifferent;

- Granularity – Preferences can be expressed at different levels of granularity, i.e., for tuples, sets, relations, attributes, and relationships. For example, "I

[6] http://www.uddi.org/pubs/uddi_v3.htm

want three cars, and prefer one of them to be a sport car" is a preference expressed over a set;

- Context – Preferences can be context-free or can hold under specific conditions. For example, "I like passenger cars when where accompanied with my family for holidays";

- Aspects – Preferences may vary based on their intensity, elasticity, complexity and other aspects. For example, a preference may express a like "I like sport cars" or dislike "I do not like vans".

2.2.2 Preference Aggregation

Different aggregation mechanisms can be applied to combine, infer or override preferences. Preference aggregation mechanisms can be grouped into the following categories [SKP11]:

- Quantitative aggregation – These mechanisms combine preferences by assigning global scores to the tuples, which are thus ordered in a quantitative way. For example, "I interest in sport cars is 0.7 and in petrol engine is 0.6", then a sport car with a petrol engine may have a score of 1.3, i.e., the sum of the two weights or a score of 0.6, i.e., the minimum of the weights, and so on;

- Qualitative aggregation – These mechanisms combine preferences resulting in a relative (i.e., qualitative) ordering of the tuples. The most popular qualitative aggregation is the Pareto preference composition, where the involved preferences are considered equally important. For example, "I like sport cars better than passenger cars, and petrol engine better than diesel engine", then a sport car with a petrol engine is preferred over a sport car with a diesel engine, a passenger car with a petrol engine or a passenger car with a diesel engine, but a sport car with a diesel engine and a passenger car with a petrol engine are indifferent;

- Heterogeneous aggregation – These mechanisms are used to combine preferences of different granularity; e.g., using the Pareto preference composition.

2.2.3 Preference Query Processing

Preferences are used in query processing to provide users with customized results. There are roughly two different lines of work on using preferences in query processing [SKP11]:

- Expanding queries – These methods assume the existence of a number of user preferences and appropriately rewrite regular queries to incorporate them. This process is often referred to as query personalization. For example, determining which preferences are related to a given query, and providing users with a flexible way to express their preferences, then the query is expanded with the selected preferences;

- Employing preference operators – These methods use special database operators to explicitly express preferences within queries. The most popular preference operators are: *(i)* the top-k operator, where the items in the result are ranked according to a user defined scoring function and the results with the k highest scores are returned to the user; and *(ii)* the skyline operator, which comprises those items that are not dominated (in the sense of Pareto) by any other item in the result; an item dominates another item, if the former is as good as or better than the latter with regard to a set of preferences and strictly better in at least one preference.

2.3 Fuzzy Sets

Fuzzy set theory was introduced by Zadeh [Zad65] to model sets whose boundaries are not well defined. Typical examples are those described using adjectives of the natural language, such as *cheap*, *expensive*, etc. For such sets, the transition between full membership and full mismatch is gradual rather than crisp.

2.3.1 Definition

A fuzzy set $\mathcal{F}$ on a referential $\mathcal{X}$ is characterized by a membership function $\mu_{\mathcal{F}} : \mathcal{X} \rightarrow [0, 1]$ where $\mu_{\mathcal{F}}(x)$ denotes the grade of membership of x in $\mathcal{F}$. In particular, $\mu_{\mathcal{F}}(x) = 1$ reflects full membership of x in $\mathcal{F}$, while $\mu_{\mathcal{F}}(x) = 0$ means absolute

non-membership. When $0 < \mu_{\mathcal{F}}(x) < 1$, x has partial membership in $\mathcal{F}$. $\mathcal{F}$ is normalized if $\exists x \in \mathcal{X} : \mu_{\mathcal{F}}(x) = 1$.

2.3.2 Practical Representation

Two crisp sets are of particular interest when defining a fuzzy set $\mathcal{F}$:

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

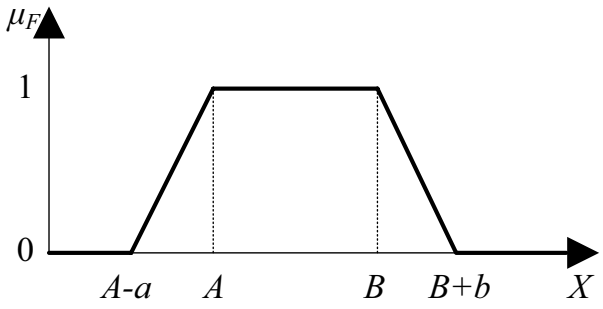


Figure 2.2: Trapezoidal Membership Representation

In practice, the membership function associated with $\mathcal{F}$ has often a trapezoidal shape. Then, $\mathcal{F}$ is expressed by the quadruplet (A, B, a, b) where $C(\mathcal{F}) = [A, B]$ and $S(\mathcal{F}) = [A - a, B + b]$ (cf. Figure 2.2). A regular interval [A, B] can be seen as a fuzzy set represented by the quadruplet $(A, B, 0, 0)$.

2.3.3 Fuzzy Operations

Given two fuzzy sets $\mathcal{F}$ and $\mathcal{G}$ in the universe (i.e., referential) $\mathcal{X}$, the intersection, union, and complement fuzzy operations are defined as follows [DP00]:

- Intersection – The membership function of the intersection of $\mathcal{F}$ and $\mathcal{G}$ is defined by $\mu_{\mathcal{F}\cap\mathcal{G}} = \top(\mu_{\mathcal{F}}(x), \mu_{\mathcal{G}}(x))$ where $\top$ is a t-norm operator that generalizes the conjunction operation (e.g., $\top(x, y) = min(x, y)$ and $\top(x, y) = x \cdot y$);
- Union – The membership function of the union of $\mathcal{F}$ and $\mathcal{G}$ is defined by $\mu_{\mathcal{F}\cup\mathcal{G}} = \bot(\mu_{\mathcal{F}}(x), \mu_{\mathcal{G}}(x))$ where $\bot$ is a co-norm operator that generalizes the disjunction operation (e.g., $\bot(x, y) = max(x, y)$ and $\bot(x, y) = x + y - x \cdot y$);

- Complement – The membership function of the complement of $\mathcal{F}$, denoted by $\mathcal{F}^C$, is defined by $\mu_{\mathcal{F}^C}(x) = 1 - \mu_{\mathcal{F}}(x)$.

As usual, the logical counterparts of the theoretical set operators $\cap$, $\cup$ and complementation correspond respectively to conjunction $\wedge$, disjunction $\vee$ and negation $\neg$. See [DP00] for more details.

2.3.4 Fuzzy Implications

A fuzzy implication is an operator $\rightarrow_f$ defined from $[0,1]^2$ to $[0,1]$ satisfying the following conditions [Yag80]:

- $x \rightarrow_f 1 = 1$;
- $0 \rightarrow_f x = 1$;
- $1 \rightarrow_f x = x$;
- if $y \geq z$ then $x \rightarrow_f y \geq x \rightarrow_f z$, i.e., increasing with respect to the second argument;
- if $x \leq z$ then $x \rightarrow_f y \geq z \rightarrow_f b$, i.e., decreasing with respect to the first argument.

Two families of fuzzy implications are studied in the fuzzy literature due to their semantic properties and the fact that their results are similar with the ones of usual implications, material implications, when the arguments are 0 or 1:

- *R-implications* – These fuzzy implications are defined by $x \rightarrow_f y = sup\{\beta \in [0,1], \top(x,\beta) \leq y\}$, where $\top$ is a t-norm operator. The two most used R-implications are *(i) Godöl* implication: $x \rightarrow_{God} y = 1$ if $x \leq y$, 0 otherwise; and *(ii) Goguen* implication: $x \rightarrow_f I_{Gog} y = 1$ if $x \leq y$, y/x otherwise;
- *S-implications*: These fuzzy implications are defined by $x \rightarrow_f y = \perp(1-x, y)$, where $\perp$ is a co-norm operator. The two most popular S-implications are *(i) Kleene-Dienes* implication: $x \rightarrow_{Kle} y = max((1-x, y)$; and *(ii) Lukasiewicz* implication: $x \rightarrow_{Luk} y = min(1 - x + y, 1)$.

Note that *Lukasiewicz* implication is, also, an R-implication. For a complete presentation on fuzzy implications, the reader is invited to see [DP00].

2.3.5 Fuzzy Inclusion

Given two fuzzy sets $\mathcal{F}$ and $\mathcal{G}$ in the universe $\mathcal{X}$, $\mathcal{F} \subseteq \mathcal{G}$ if and only if $\forall x \in \mathcal{X}, \mu_{\mathcal{F}}(x) \leq \mu_{\mathcal{G}}(x)$. Moreover, if $\mathcal{F}$ is not included in $\mathcal{G}$, there is two main approaches to define an inclusion degree of $\mathcal{F}$ in $\mathcal{G}$ [BBP96]:

- Quantitative method – The inclusion degree of $\mathcal{F}$ in $\mathcal{G}$ is computed in the following way: $Deg(\mathcal{F} \subseteq \mathcal{G}) = \frac{|\mathcal{F} \cap \mathcal{G}|}{|\mathcal{F}|} = \frac{\sum_{x \in X} \top(\mu_{\mathcal{F}}(x), \mu_{\mathcal{G}}(x))}{\sum_{x \in X} \mu_{\mathcal{F}}(x)}$ where $\mid \mathcal{F} \mid$ stands for the cardinality of $\mathcal{F}$ and defined by $|\mathcal{F}| = \sum_{x \in \mathcal{X}} \mu_{\mathcal{F}}(x)$;
- Logic method – The degree of inclusion is given by the following expression: $Deg(\mathcal{F} \subseteq \mathcal{G}) = min_{x \in X}(\mu_{\mathcal{F}}(x) \rightarrow_f \mu_{\mathcal{G}}(x))$ where $\rightarrow_f$ stands for a fuzzy implication.

2.3.6 Modeling Preferences

Fuzzy sets provide a suitable tool to express user preferences. A fuzzy set-based approach to deal with preference queries is founded on the use of the notion of membership functions that describe the preference profiles of user for each attribute domain involved in the query [DP96, HKP08].

The user does not specify crisp (Boolean) criteria, but gradual ones like *affordable*, *very cheap* and *fairly expensive* (for the attribute price), whose satisfaction is 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 a set of discriminated elements according to their global satisfaction with respect to the fuzzy criteria appearing in the query. So, a complete pre-order is obtained. One can limit the number of answers by using a quantitative calibration (e.g., return the top-k answers) or a qualitative calibration (e.g., return the answers that satisfy the query with a degree above a threshold η).

2.4 Possibility Theory

Possibility theory was introduced by Lotfi Zadeh [Zad78] for dealing with some facets of uncertainty due to incomplete state of knowledge where probability theory is inappropriate. Possibility theory offers a qualitative model for uncertainty where

a piece of information is represented by means of a possibility distribution encoding a complete pre-order over the possible situations [DP88]. A possibility distribution is frequently attached to a variable v taking a single value, possibly not well known, on a domain Ω.

2.4.1 Possibility Distribution

A possibility distribution of a variable v, on a domain Ω, is a function π_v from Ω to $[0, 1]$, where $\pi_v(x)$ expresses the degree to which x ($x \in \Omega$) is a possible value for v. The normalization condition imposes that at least one of the values of the domain "x_0" is completely possible for any variable v, i.e., $\pi_v(x_0) = 1$ in case of consistent information. When the domain is discrete, a possibility distribution of any variable v of Ω can be written $\pi_v = \{\pi_v(x_1)/x_1, \pi_v(x_2)/x_2, \ldots, \pi_v(x_m)/x_m\}$ where x_i is a candidate value and $\pi_v(x_i)$ is its possibility degree with respect to the variable v.

2.4.2 Possibility and Necessity

Whereas probability theory uses a single number, the probability, to describe how likely an event is to occur, in possibility theory, an event e is characterized by two measures: its possibility and its necessity. The possibility measure and necessity measure are defined as follows:

- Possibility measure – The possibility measure is a function $\Pi : 2^{\Omega} \rightarrow [0, 1]$ such that:
  - $\Pi(\emptyset) = 0$;
  - $\Pi(\Omega) = 1$;
  - $\Pi(e_1 \cup e_2) = \max(\Pi(e_1), \Pi(e_2))$.
- Necessity measure – The necessity measure is defined by $N(e) = 1 - \Pi(\overline{e})$ where $\overline{e}$ is the event opposite to e. From this formula, it is straightforward to show that:
  - $N(e) \leq \Pi(e)$;
  - $\Pi(e_1 \cap e_2) = \min(N(e_1), N(e_2))$;
  - $\Pi(e) + \Pi(\overline{e}) \geq 1$;

2.4.3 Interpretation

One can distinguish four cases to characterize the uncertainty of an event e:

- $N(e) = 1$ means that e is necessary, i.e., e is certainly true. It implies that $\Pi(e) = 1$;
- $\Pi(e) = 0$ means that e is impossible, i.e., e is certainly false. It implies that $N(e) = 0$;
- $N(e) = 0$ means that e is unnecessary. I would not be surprised at all if e does not occur. It leaves $\Pi(e)$ unconstrained;
- $\Pi(e) = 1$ means that e is possible. I would not be surprised at all if e occurs. It leaves $N(e)$ unconstrained.

2.4.4 Possibility vs Probability

It is worth to note that possibility and probability measures carry two distinct semantics. A probability value provides a frequency of occurrence of an event, which also allows ordering the different events depending on their frequency. A value of possibility is purely ordinal in the sense that it is only intended to order the different choices. For example, assume that the universe $\Omega = \{red, black\}$ represents the results of a casino roulette, the probability p defined by $p(\{red\}) = 0.8$ and $p(\{black\}) = 0.2$ indicates that the frequency of the event "red (resp. black) occurs" is 8 (resp. 2) times out of 10. The event "red occurs" is four times more frequent than the event "black occurs". If we must bet on one of these two colors, red is first class. If the result of the roulette is modeled by the possibilities, it is always possible to classify the two possibilities but both event frequencies are not expressed. Roughly speaking, possibility theory is adapted to the context where frequencies are not available. For instance, the response time of a Web service s_i must be modeled using a possibility distribution because no information is provided about the quality of the network to determine the different frequencies (knowing that the response time vary with the quality of the network).

2.5 Conclusion

In this chapter, we presented the main concepts around Web service technology and preferences. We also introduced the reader to fuzzy sets and possibility theory. Now, the reader should be able to understand our contributions described in the next four chapters as well as the rest of this dissertation.

Chapter 3

Top-k Web Service Compositions with Fuzzy Preferences

Contents

3.1 Introduction

Recent years have witnessed a growing interest in the use of Web services as a reliable means for e-commerce, content publication and management. Thereby, enabling users to perform several operations, like searches, purchases and data uploads. This type of Web services is known as *data-driven Web services* [DSV04] or data services for short, where Web services are typically powered by databases. Moreover, Web users often need to compose different Web services to achieve a more complex task that cannot be fulfilled by an individual Web service. *Data Web Service Composition* is a powerful solution to answer the user's complex queries by combining primitive simple Data Web services to realize value-added services on top of existing ones. Moreover, user preferences play a major role in the customization of the composition process. A more general and crucial approach to represent preferences is based on the fuzzy sets theory [DP00][HKP08]. Fuzzy sets are very appropriate for the interpretation of linguistic terms, which constitute a convenient way for users to express their preferences. For example, when expressing preferences about the price of a car, users often employ linguistic terms like *rather cheap*, *affordable* and *not expensive*.

One of the most challenging problems in data service composition is that due to the proliferation of data services and service providers, a large number of candidate data service compositions that would use different, most likely competing, data services may be used to answer the same query. It is therefore important to set up an effective data service composition framework that would identify and retrieve the most relevant data services and return the top-k data service compositions according to the user preferences.

The following example presents a typical scenario from the e-commerce domain that clearly shows the different challenges involved in finding the top-k data service compositions.

3.1.1 Motivating Example

Consider a set of car trading Web services in Table 3.1 (i.e., typical data services that can be provided by systems like the e-Bay). The symbols "$" and "?" denote inputs and outputs of data services, respectively. Data services providing the same

functionality belong to the same service class. For instance, the data services s_{21}, s_{22}, s_{23} and s_{24} belong to the same class $\mathcal{S}_2$. Each data 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.

Table 3.1: Example of Data Services

Data service	Functionality	Constraints
$s_{11}(\$x, ?y)$	Returns the automakers y in a given country 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 *roughly* 4
$s_{33}(\$x, ?y, ?z)$		y is *powerful*, z is *high*
$s_{34}(\$x, ?y, ?z)$		y is $[60, 110]$, z is $[3.5, 5.5]$

Let us now assume that a user, Bob, wants to buy a car. He sets his preferences and submits the following query $\mathcal{Q}_1$: "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 uses the services described in Table 3.1 to obtain such information. He will have to invoke data service s_{11} to retrieve the French automakers, then invoke one or more of the data 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 data services $s_{31}, s_{32}, s_{33}, s_{34}$ to retrieve the power and the consumption of retrieved cars.

To select the car that better satisfies his requirements, Bob needs to go through a series of trial-run processes. If the number of available services is large, this manual process would be very painstaking and raises the following challenges:

- How to understand the semantics of the published data services to select the relevant ones that can contribute to answering the query at hand;

- How to retain the most relevant data services (several similar data services offer the same functionality but are associated with different constraints) that better satisfy the user's fuzzy preferences (i.e., preferences based on fuzzy terms);

- How to generate the best k data service compositions that satisfy the query.

3.1.2 Contributions

The first challenge is already tackled in [BBM10] by proposing a semantic annotation of data services that describes the services functionality and an efficient RDF-based query rewriting approach that generates automatically the data service compositions for a given query (which does take into account any user preference). In this paper, we focus on the second and third challenges. We leverages the RDF query rewriting algorithm [BBM10] to find the relevant data services that can contribute to the resolution of a given preference query. Since the number of candidate data services for a composition may be still large, performing an exhaustive search, i.e., generate all possible combinations, to find the best data service compositions is not practical as the problem of composition is known to be NP-hard, i.e., any exact solution to this problem has an exponential cost. Therefore, reducing the search space by focusing only on the best data services of each service class is crucial for reducing the computational cost. Our main contributions in this chapter include the following:

- As data services of the same class have the same functionality and only differ in their constraints, the relevance of each service with respect to a given query can be reduced to the relevance of their constraints with respect to the user preferences. For this purpose, we investigate multiple methods for computing the matching degrees between the preferences involved in the query and the data services' constraints;

- We present a method for further reducing the search space by examining only the top-k data services of each service class. In particular, we define a ranking criterion based on a fuzzy dominance relationship in order to select the top-k data services in each service class, we then compose these data services and return only the top-k data service compositions;

- To avoid returning similar data service compositions, i.e., those returning similar informations, we also propose a diversified top-k data service compositions method that aims to both improve the diversity of top-k selection and maintain as possible top-k highest ranked ones;
- We propose a comprehensive architecture of our composition system and evaluate our approach through a set of thorough experiments.

The rest of this chapter is organized as follows. In Section 3.2, we formally define the studied problem. Section 3.3 describes the proposed fuzzy dominance relationship and a ranking approach for data services. Section 3.4 is devoted to both top-k and diversified top-k data service composition methods for answering preference queries. Section 3.5 presents the architecture of our implemented composition system for preference query answering and reports the results of a set of thorough experimental evaluations. Finally, Section 3.6 concludes the chapter.

3.2 Preferences-Based Data Service Composition Model

Assume a preference query $\mathcal{Q}$ and a set $\mathbb{S} = \{\mathcal{S}_1, \cdots, \mathcal{S}_n\}$ of service classes, which classify the universe of available data services according to their functionality. Each service class $\mathcal{S}_i = \{s_{i1}, ..., s_{in_i}\}$, $\mathcal{S}_i \in \mathbb{S}$, consists of all data services that deliver the same functionality but potentially differ in terms of constraints (see Table 3.1). Individual data services of a service class $\mathcal{S}_i$ may handle, i.e., are relevant to answer, only a part (query component) q_i of the query $\mathcal{Q}$ and each has its own constraints that may partially match the user preferences.

3.2.1 Preference Queries

We adopt a declarative approach to Web services composition, i.e., instead of selecting and composing Web services manually, users formulate their composition queries over domain ontologies. We consider conjunctive preference queries expressed over domain ontologies using a slightly modified version of SPARQL[7], the de facto query language for the Semantic Web. Figure 3.1 depicts a portion of the mediated ontology in an e-commerce domain, in particular the automobile domain.

[7] http://sparql.org/

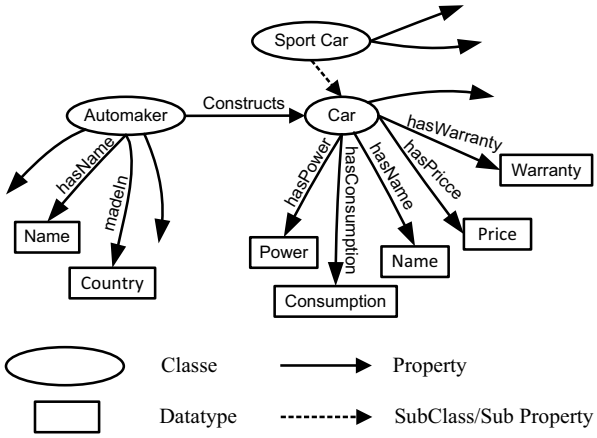


Figure 3.1: Sample of Ontology

Formally, a conjunctive preference query $\mathcal{Q}$ has the form $\mathcal{Q}(X)$:-$<\varphi(X,Y), \mathcal{P}>$, where:

- $\mathcal{Q}(X)$ is the head of $\mathcal{Q}$, has the form of a relational predicate and represents the result of the query.
- $\varphi(X,Y)$ is the body of $\mathcal{Q}$, contains a set of RDF triples where each triple is of the form (`subject.property.object`). X and Y are called distinguished and existential variables, respectively.
- $\mathcal{P} = \{p_1, ..., p_d\}$ is a set of preferences expressed using fuzzy sets on X and Y variables.

Membership functions of fuzzy terms are implemented as Web services and can be shared by users. They are used in the `PREFERRING` clause of the query where the URL of the implementing Web service is mentioned. More details are provided in Section 3.5. The head and body of $\mathcal{Q}$ are defined in `SELECT` and `WHERE` clauses, respectively. For example, query $\mathcal{Q}_1$ given in Section 3.1 is expressed as follows:

```
URL=http://vm.liris.cnrs.fr:36880/MembershipFunctions/
SELECT ?n ?pr ?w ?pw ?co
WHERE {?Au rdf:type AutoMaker  ?Au hasCountry 'France'  ?Au makes ?C
       ?C rdf:type Car  ?C hasName ?n  ?C hasPrice ?pr
       ?C hasWarranty ?w  ?C hasPower ?pw  ?C hasConsumption ?co}
PREFERING {?pr is 'URL/Affordables', ?w is 'URL/around(18)',
```

?pw is 'URL/Normal', ?co is 'URL/Medium'}

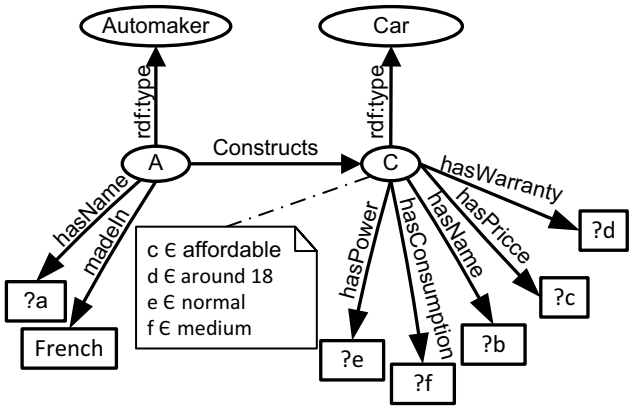


Figure 3.2: Graphical Representation of the Fuzzy Query

For instance, `?w is 'URL/around(18)'` means that the user prefers services that provide cars with a warranty around 18 months. The semantics of *around 18* is given in `URL = http://vm.liris.cnrs.fr:36880/MembershipFunctions/around(18)`. `SELECT` and `WHERE` clauses define the head and body of $\mathcal{Q}$, respectively. `PREFERING` clause indicates the preferences in $\mathcal{Q}$. Figure 3.2 gives the graphical representation of query $\mathcal{Q}_1$. The ovals *Automaker* and *Car* are concepts in the ontology. The arcs (e.g., *Constructs*, *hasPrice*, etc) are properties in the ontology. The ovals A and C are existential variables, whereas a, b, c, d, e and f are distinguished variables.

3.2.2 Data Services

The functionalities of data services, as opposed to traditional Web services that encapsulate software artifacts, can be only captured when representing the semantic relationship between inputs and outputs [BBM10, MBM+07]. Therefore, we modeled data services as RDF Parameterized Views (RPVs) over domain ontologies. Each view captures the semantic relationships between input and output sets of a data service using concepts and relations whose semantics are formally defined in ontologies. Functionalities of data services are provided under some data constraints. For example, z `is` *cheap*, t `is` *short* (for data service s_{21} in Table 3.1).

Formally, a data service s_{ij} of a service class $\mathcal{S}_i$ is described as a predicate $s_{ij}(\$X_i, ?Y_i)$:-$<\phi_i(X_i, Y_i, Z_i), \mathcal{C}_{ij}>$ where:

- X_i and Y_i are the sets of input and output variables of s_{ij}, respectively. Input and output variables are also called distinguished variables. They are prefixed with the symbols "$" and "?", respectively.

- $\phi_i(X_i, Y_i, Z_i)$ represents the functionality of the data service. This functionality is described as a semantic relationship between input and output variables. Z_i is the set of existential variables relating X_i and Y_i.

- $\mathcal{C}_{ij} = \{C_{ij_1}, ..., C_{ij_i}\}$ is a set of constraints expressed as intervals or fuzzy sets on X_i, Y_i or Z_i variables.

Each data service requires a particular set of inputs (parameter values) to retrieve a particular set of outputs; i.e., outputs cannot be retrieved unless inputs are bound. For example, one cannot invoke data service s_{31} without specifying the car for which it need to know the power and the consumption. Inputs and Outputs are prefixed with "$" and "?", respectively in the head of the view ($s_{ij}(\$X_i, ?Y_i)$). X_i and Y_i variables are defined in the WSDL description of data services. Functionality ϕ_i of and constraints $\mathcal{C}_{ij}$ over a data service s_{ij} 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 of and constraints over the data service s_{21} in Table 3.1:

```
URL=http://vm.liris.cnrs.fr:36880/MembershipFunctions/
RDFQuery{
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}}
CONSTRAINTS{?z is 'URL/Cheap', ?t is 'URL/Short'}
```

`SELECT` and `WHERE` clauses define the functionality of s_{21} and `CONSTRAINTS` clause gives the fuzzy constraints of service s_{21} given in Table 3.1. Figure 3.3 gives the graphical representation of the data services given in Table 3.1.

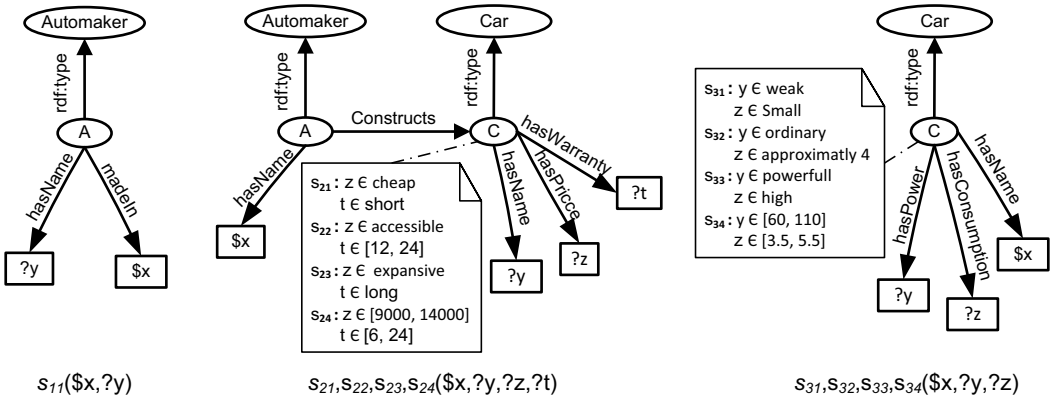


Figure 3.3: Functionality of Data Services

3.2.3 Discovering Relevant Data Services

Given a preference query $\mathcal{Q}$, we use the RDF query rewriting algorithm described in [BBM10] to discover the parts of $\mathcal{Q}$ that are covered by each data service – recall that in the general case data services may cover only parts of $\mathcal{Q}$. For simplicity, assume a set of service classes $\mathbb{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ where each $\mathcal{S}_i$ is a set of data services that provide the same functionality. Each data service s_{ij} of the service class $\mathcal{S}_i$ can cover a part of $\mathcal{Q}$ referred to as q_i. A data service $s_{ij} \in \mathcal{S}_i$ covers a part q_i of $\mathcal{Q}$ if the functionality of s_{ij} completely matches q_i and its constraints match completely or partially the preference constraints involved q_i. Therefore, to differentiate the most relevant data services, we need to compute a matching degree between the preference constraints involved in q_i and the data services' constraints.

To determine the matching degree of a service s_{ij}, traditional approaches assign to each constraint which corresponds to a preference in q_i, a matching degree. Then, this degree can be computed as an aggregation of individual matching degrees (i.e., the matching degree of each constraint). One direction is to assign weights to individual matching degrees [DHM+04]. However, users may not know how to set trade-off between different relevancies using numbers and an imprecise specification of weights could miss their desired services. They thus lose the flexibility to select their desired services. Computing the skyline from services [ASR10, YB10b, YB10a, YB12] comes as a natural solution to overcome this limitation. Skyline computation has received significant consideration in database research; e.g., see [BKS01, TEO01, KRR02, PTFS03, Cho03, GSG05a]. For a d-dimensional dataset, the skyline consists

of the set of points that are not dominated by any other point. A point u dominates another point v if and only if u is at least as good as v in all dimensions and (strictly) better than v in at least one dimension.

However, as shown in [SSS+09, SSSS10] considering a single matching method for evaluating services is a very coarse metric. For this purpose, we investigate multiple methods from the fuzzy set theory to compute the matching degrees between user preferences and data services' constraints, namely, constraints inclusion methods that measure the to what extent the items returned by a given data service satisfy the user preferences.

Let $C \equiv x \text{ is } \mathcal{F}$ and $C' \equiv x \text{ is } \mathcal{G}$ be two fuzzy constraints. From Section 2.3.5 two classes of constraint inclusion methods may be considered:

- Quantitative Method (QM) – The degree of inclusion is given by: $Deg(C \subseteq C') = \frac{\sum_{x \in X} \top(\mu_{\mathcal{F}}(x), \mu_{\mathcal{G}}(x))}{\sum_{x \in X} \mu_{\mathcal{F}}(x)}$. In our example, we use the *"min"* and *"product"* t-norms. However, other t-norms can be used. The methods that rely on $\top$ ="*min*" and $\top$ = *"product"* are denoted by M-QM and P-QM, respectively.

- Logic Method (LM) – The degree of inclusion is given by: $Deg(C \subseteq C') = min_{x \in X}(\mu_{\mathcal{F}}(x) \rightarrow_f \mu_{\mathcal{G}}(x))$. In our example, we make use of two fuzzy implications: Gödel ($a \rightarrow_G b = 1$ if $a \leq b$, 0 otherwise) and Lukasiewicz ($a \rightarrow_L b = 1$ if $a \leq b$, $1 - a + b$ otherwise) implications; the methods based on these two implications are denoted by G-LM and L-LM, respectively. Also, other fuzzy implications like *Goguen* or *Kleene-Dienes* implications can be used.

Each relevant data service is then associated with a set of matching degrees. Table 3.2 shows the matching degrees between each service s_{ij} in Table 3.1 and its corresponding component q_i (of the query $\mathcal{Q}_1$). Service s_{11} covering component q_1 does not have a matching degree since there are no user preferences involved in q_1. However, each data service covering component q_2 is associated with four (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 data services covering component q_3, the first and second values represent the matching degrees of the constraints `power` and `consumption`, respectively.

Table 3.2: Matching Degrees between Data Services' Constraints and Preference Constraints of $\mathcal{Q}_1$

s_{ij}	q_i	M-QM	P-QM	G-LM	L-LM
s_{11}	q_1	–	–	–	–
s_{21}	q_2	$(1, 0.57)$	$(0.98, 057)$	$(1, 0)$	$(0.80, 0)$
s_{22}		$(0.89, 1)$	$(0.77, 1)$	$(0, 1)$	$(0.50, 1)$
s_{23}		$(0.20, 0.16)$	$(0.13, 0.13)$	$(0, 0)$	$(0, 0)$
s_{24}		$(0.83, 0.88)$	$(0.83, 0.88)$	$(0.60, 0.50)$	$(0.60, 0.50)$
s_{31}	q_3	$(0.50, 0.36)$	$(0.46, 0.32)$	$(0, 0)$	$(0, 0)$
s_{32}		$(0.79, 0.75)$	$(0.69, 0.72)$	$(0, 0.25)$	$(0.40, 0.50)$
s_{33}		$(0.21, 0.64)$	$(0.17, 0.61)$	$(0, 0)$	$(0, 0)$
s_{34}		$(0.83, 0.85)$	$(0.83, 0.85)$	$(0.50, 0.50)$	$(0.50, 0.50)$

3.2.4 Problem Statement

Given a preference query $\mathcal{Q}$:-$<q_1, ..., q_n>$ where each part (query component) q_i is a tuple $(\overline{q}_i, \mathcal{P}_{q_i})$; $\overline{q}_i$ represents q_i without its preferences $\mathcal{P}_{q_i}$. Given a set of services classes $\mathbb{S} = \{\mathcal{S}_1, \cdots, \mathcal{S}_n\}$ where a class $\mathcal{S}_i$ regroups data services that are relevant to a query part q_i, and a set $\mathbb{M} = \{M_1, \cdots, M_m\}$ of matching methods to compute the matching degrees between the constraints on relevant data services and the user's preference. The problem to address is how to rank data services in each class $\mathcal{S}_i$ to select the most relevant ones and how to rank generated data service compositions to select the top-k ones that can answer the preference query $\mathcal{Q}$.

3.3 Fuzzy Dominance and Fuzzy Scores

In this section, we introduce the notion of fuzzy dominance relationship considered between data services. To further motivate why the fuzzy dominance is needed, we first investigate the difference between fuzzy dominance and Pareto dominance. We then define the scores associated with both the data services and the data service compositions based-on the fuzzy dominance relationship.

It is well known that under a single matching degree method (mono criteria), the dominance relationship is unambiguous. When multiple methods are applied, result-

ing in different matching degrees for the same constraints, the dominance relationship becomes uncertain. The model proposed in [PJLY07], namely probabilistic skyline overcomes this problem. Contrariwise, Skoutas et al. show in [SSS+09, SSSS10] the limitations of the probabilistic skyline to rank services and introduce the Pareto dominating score of individual services. There is, however, still some problems when applying the Pareto dominance as shown bellow.

3.3.1 Fuzzy Dominance vs Pareto Dominance

We start by defining formally the Pareto dominance, then discuss the reasons that motivate to make it fuzzy.

Definition 3.1 *(Pareto Dominance)*
Given two d-dimensional points u and v, we say that u dominates v, denoted by $u \succ v$, if and only if u is at least as good as v in all dimensions and (strictly) better than v in at least one dimension, i.e., $\forall \imath \in [1, d], u_\imath \geq v_\imath \wedge \exists \jmath \in [1, d], u_\jmath > v_\jmath$.

One can see that Pareto dominance does not allow discrimination between points with a large variance, i.e., points that are very good in some dimensions and very bad in other ones (e.g., $(1, 0)$ and $(0.80, 0)$ in Table 3.2) and good points, i.e., points that are (moderately) good in all dimensions (e.g., $(0.89, 1)$ and $(0.77, 1)$ in Table 3.2). To further illustrate this situation, let $u = (u_1, u_2) = (1, 0)$ and $v = (v_1, v_2) = (0.90, 1)$ be two matching degrees (or two points in general). 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 Pareto dominance relationship to express the extent to which a matching degrees vector (more or less) dominates another one. We define below a fuzzy dominance relationship that relies on particular monotone comparison function expressing a graded inequality of the type "strongly greater than", as the higher the value, the better is the matching degree.

Definition 3.2 *(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:

$$deg(u \succ v) = \frac{\sum_{\imath=1}^{d} \mu_{\gg}(u_\imath, v_\imath)}{d} \tag{3.1}$$

Where $\mu_{\gg}$ is a membership function of the fuzzy relation $\gg$ that expresses the extent to which u_i is more or less (strongly) greater than v_i. The membership function $\mu_{\gg}$ can be defined in an absolute way (i.e., in terms of $x - y$) as follows:

$$\mu_{\gg}(x,y) = \begin{cases} 0 & if\ x - y \leq \varepsilon \\ 1 & if\ x - y \geq \lambda + \varepsilon \\ \frac{x-y-\varepsilon}{\lambda} & otherwise \end{cases} \tag{3.2}$$

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.

Figure 3.4 gives the graphical representation of $\mu_{\gg}$ in terms of $x-y$ where H is a fuzzy parameter associated with the relation $\gg$ such that $\mu_{\gg}(x,y) = \mu_H(x-y)$. One can easily check that the trapezoidal membership function of H is $(\lambda+\varepsilon, +\infty, \lambda, 0)$.

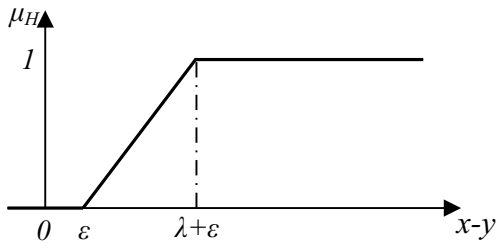


Figure 3.4: Graded Inequality Representation in terms of $x - y$

One can explain the semantics of $\mu_{\gg}$ in the following way:

- If $x - y$ is less than ε, then x is not at all strongly greater than y;
- If $x - y$ is larger than $\lambda + \varepsilon$, then x is all much greater than y;
- If $x - y$ is between ε and $\lambda + \varepsilon$, then x is much greater than y to some extent.

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 and v are incomparable provided by Pareto dominance.In the following sections, we will use the defined fuzzy dominance to compute scores of data services and their compositions.

3.3.2 Associating Fuzzy Score with a Data Service

We generalize the (Pareto) dominating score defined in [SSS+09, SSSS10] to fuzzy dominance and propose the fuzzy dominating score (DS_f) of a data service. The DS_f of a data service s_{ij} indicates the average extent to which s_{ij} dominates the whole data services of its class $\mathcal{S}_i$.

Definition 3.3 *(Fuzzy Dominating Score of a Data Service)*
The fuzzy dominating score (DS_f) of a data service s_{ij} in its class $\mathcal{S}_i$ is defined as:

$$DS_f(s_{ij}) = \frac{1}{(|\mathcal{S}_i| - 1)m^2} \sum_{\imath=1}^{m} \sum_{\substack{s_{ik} \in \mathcal{S}_i \\ k \neq j}} \sum_{\jmath=1}^{m} deg(s_{ij}^{\imath} \succ s_{ik}^{\jmath}) \tag{3.3}$$

where $s_{ij}^{\imath}$ is the matching degree of the data service s_{ij} obtained by applying the $\imath^{th}$ matching method and m stands for the number of matching methods applied. The term $(|\mathcal{S}_j| - 1)$ is used to normalize the fuzzy dominating score and make it in the range $[0, 1]$.

Table 3.3 shows the fuzzy dominating scores of the data services of our running example.

Table 3.3: Services' Scores and Top-k Data Services

Data service	Service class	Score	Top-k
s_{11}	$\mathcal{S}_1$	–	s_{11}
~~s_{21}~~	$\mathcal{S}_2$	0.527	s_{22} s_{24}
s_{22}		0.657	
~~s_{23}~~		0.027	
s_{24}		0.533	
~~s_{31}~~	$\mathcal{S}_3$	0.083	s_{32} s_{34}
s_{32}		0.573	
~~s_{33}~~		0.187	
s_{34}		0.717	

3.3.3 Associating Fuzzy Score with a Data Service Composition

Different data service compositions can be generated from service classes $\mathcal{S}_i$ to answer a user query. To rank such generated compositions, we extend the previous defined score, i.e., the fuzzy dominating score (DS_f) to data service composition and associate each composition with a DS_f. The fuzzy dominating score of a data service composition $\mathcal{CS}$ is an aggregation of different DS_f scores of its component data services. It indicates the average number of possible compositions that $\mathcal{CS}$ more or less dominates.

Definition 3.4 *(Fuzzy Dominating Score of a Data Service Composition)*
Let $\mathcal{CS} = \{s_{1j_1}, ..., s_{nj_n}\}$ be a composition of n services and $d = d_1 + ... + d_n$ be the number of preference constraints in $\mathcal{Q}$, where d_i is the number of constraints (resp. preferences) involved in the service s_{ij_i} (resp. in the query component q_i). The DS_f of $\mathcal{CS}$ is then computed as follows:

$$DS_f(\mathcal{CS}) = \frac{1}{d} \sum_{i=1}^{n} d_i \cdot DS_f(s_{ij_i}) \tag{3.4}$$

It is important to note that not all compositions are valid. A composition $\mathcal{CS}$ of data services is valid if (i) it covers the user query $\mathcal{Q}$; (ii) it contains one and only one data service from each service class $\mathcal{S}_i$ and (iii) it is executable. A composition is said to be executable if all input parameters necessary for the invocation of its component data services are bound or can be made bound by the invocation of primitive data services whose input parameters are bound. For example, the composition $\{s_{11}(\$x, ?y)$, $s_{21}(\$x, ?y, ?z, ?t)$, $s_{31}(\$x, ?y, ?z)\}$) is executable since the inputs parameters of its component data services are all bound (the value of the variable x is supplied by the user). More details are provided in [BBM10].

3.4 Top-k Data Service Compositions

3.4.1 Efficient Generation of Top-k Data Service Compositions

The problem of top-k data service compositions entails computing the scores of each data service composition and returning the top-k highest ranked ones. A straightforward method to find the top-k data service compositions that answer a query is to generate all possible compositions, compute their scores, and return the

top-k ones. Clearly, 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 service compositions. This technique allows eliminating data services from their classes before generating the compositions, i.e., data services that we are sure that if they are composed with others, the obtained compositions are not in the top-k. The basic idea is to compute the score of each data service in its class, then only the best ones in each class are retained. The retained data services are then composed, and the scores of obtained compositions are computed; the top-k ones can be then returned to users. To this end, we introduce the following Lemma and Theorem.

Lemma 3.1

Let $\mathcal{CS} = \{s_{1j_1}, ..., s_{nj_n}, s\}$ *and* $\mathcal{CS}' = \{s_{1j_1}, ..., s_{nj_n}, s'\}$ *be two similar data service compositions that only differ in the data services* s *and* s'*. Then, the following statement holds:* $DS_f(s) > DS_f(s') \Rightarrow DS_f(\mathcal{CS}) > DS_f(\mathcal{CS}')$.

Proof

Denoting by d' the number of constraints contained in s and s', we have: $DS_f(\mathcal{CS}) = \frac{1}{d}\sum_{i=1}^{n} d_i \cdot DS_f(s_{ij_i}) + \frac{d'}{d} \cdot DS_f(s)$ and $DS_f(\mathcal{CS}') = \frac{1}{d}\sum_{i=1}^{n} d_j \cdot DS_f(s_{ij_i}) + \frac{d'}{d} \cdot DS_f(s')$. Then, $DS_f(\mathcal{CS}) - DS_f(\mathcal{CS}') = \frac{d'}{d}(DS_f(s) - DS_f(s'))$. Since $\frac{d'}{d} > 0$ and $DS_f(s) - DS_f(s') > 0$ (as $DS_f(s) > DS_f(s')$), we have $DS_f(\mathcal{CS}) - DS_f(\mathcal{CS}') > 0$. Hence, $DS_f(\mathcal{CS}) > DS_f(\mathcal{CS}')$. □

Lemma 3.1 indicates that the best data services in their classes will generate the best data service compositions.

Theorem 3.1

Let $\mathcal{CS} = \{s_{1j_1}, ..., s_{nj_n}\}$ *be a composition of* n *data services. Let top-k.*$\mathcal{S}_i$ *and top-k.*$\mathbb{CS}$ *be the top-k data services of the service class* $\mathcal{S}_i$ *and the top-k data service compositions, respectively. Then,* $\exists s_{ij_i} \in \mathcal{CS}, s_{ij_i} \notin$ *top-k.*$\mathcal{S}_i \Rightarrow \mathcal{CS} \notin$ *top-k.*$\mathbb{CS}$.

Proof

Assume that $\mathcal{CS} \in$ *top-k.*$\mathbb{CS}$ and $\exists s_{ij_i} \in \mathcal{CS}, s_{ij_i} \notin$ *top-k.*$\mathcal{S}_i$. This means that $\exists s'_{ij_1}, ..., s'_{ij_k} \in \mathcal{S}_i$ such as $DS_f(s'_{ij_\ell}) > DS_f(s_{ij_i})$. By replacing s_{ij_i} in $\mathcal{CS}$ with the data services $s'_{ij_1}, ..., s'_{ij_k}$, we obtain k data service compositions $\mathcal{CS}_1, ..., \mathcal{CS}_k$ such

as $DS_f(\mathcal{CS}_i) > DS_f(\mathcal{CS})$ according to Lemma 3.1. This contradicts our hypothesis. Hence, $\mathcal{CS} \notin$ *top-k*.$\mathbb{CS}$. □

From Theorem 3.1, we can see that the top-k sets of the different service classes are sufficient to compute the top-k data service compositions that answer the considered query.

The fourth column of Table 3.3 shows the top-k ($k = 2$) data services in each service class according the fuzzy dominating scores. Thus, relevant data services that are not in the top-k of their classes are eliminated. They are crossed out in Table 3.3. The other data services are retained. The top-k data service compositions are generated from different top-k.$\mathcal{S}_i$ classes. Table 3.4 shows the possible compositions along with their fuzzy dominating scores, as well as the top-k compositions (i.e., $\mathcal{CS}_2$, $\mathcal{CS}_4$) of our running example.

Table 3.4: Compositions' Scores and Top-k Ones

Composition	Composition score	Top-k
$\mathcal{CS}_1 = \{s_{11}, s_{22}, s_{32}\}$	0.615	
$\mathcal{CS}_2 = \{s_{11}, s_{22}, s_{34}\}$	0.687	$\mathcal{CS}_2$
$\mathcal{CS}_3 = \{s_{11}, s_{24}, s_{32}\}$	0.553	$\mathcal{CS}_4$
$\mathcal{CS}_4 = \{s_{11}, s_{24}, s_{34}\}$	0.625	

3.4.2 Top-k Service Compositions Algorithm

The algorithm, hereafter referred to as TKSC, computes the top-k data service compositions according to the fuzzy scores (see Algorithm 3.1). The algorithm proceeds as the following steps.

Step 1 (lines 2-9): *Find the relevant data services and compute their matching degrees* – Each service class $\mathcal{S}_i$ whose data services cover a query component, q_i, is added to the list of relevant classes $\mathcal{R}$. If its data services touch the query's user preferences, i.e., there is one or more preference constraint involved in the query part covered by the data services of $\mathcal{S}_i$, then compute its different matching degrees according to the number of methods;

Algorithm 3.1: TKSC

Input: $\mathcal{Q}$ preference query; $\mathbb{S} = \{\mathcal{S}_1, \cdots, \mathcal{S}_n\}$ set of service classes;
$\mathbb{M} = \{\mathcal{M}_1, \cdots, \mathcal{M}_m\}$ set of matching methods; $k \in \mathbb{N}$; $\varepsilon \geq 0$; $\lambda > 0$;

Output: the top k compositions

```
1  begin
2      foreach S_i in 𝕊 do
3          s ← random(S_i, 1);
4          if ∃q_i ∈ 𝒬; cover(s, q_i) then
5              ℛ ← ℛ ∪ S_i;
6              if 𝒫_{q_i} ≠ ∅ then
7                  foreach s_ij in S_i do
8                      foreach M_ℓ in 𝕄 do
9                          ComputeMatchingDegree(C_ij, 𝒫_{q_i}, 𝕄);
10     foreach S_i in ℛ do
11         if 𝒫_{q_i} = ∅ then
12             top-k.S_i ← random(S_i, k);
13         else
14             foreach s_ij in S_i do
15                 ComputeServiceScore(s_ij);
16             top-k.S_i ← top(k, S_i);
17     ℂ𝕊 ← ComposeServices(top-k.S_1, ··· , top-k.S_n);
18     foreach 𝒞𝒮 in ℂ𝕊 do
19         ComputeCompositionScore(𝒞𝒮);
20     return top(k, ℂ𝕊);
```

Step 2 (lines 10-16): *eliminate less relevant data services* – For each relevant service class $\mathcal{S}_i \in \mathcal{R}$ whose data services do not touch the user preferences, select randomly k services since they are all equal with respect to user preferences. Otherwise, i.e., its data services touch the user preferences, compute the score of its data services, and retain only the top-k ones;

Step 3 (lines 17-20): *return top-k compositions* – Compose the retained services, i.e., the top-k in each relevant service class, then, compute the scores of generated compositions. Finally, provide the user with the top-k ones.

3.4.3 Diversity-aware Top-k Data Service Compositions

Different similar data services could exist in each class $\mathcal{S}_i$ leading to similar data services compositions. A little variety in the top-k data services compositions list will probably lead to the user frustration. For this reason, it is crucial to provide users with the data service compositions that are still relevant to their preferences but less similar to each other, i.e., as diverse as possible. Diversification is thus needed to improve user satisfaction. Diversification allows to find compositions that cover many aspects of users information needs. Consider, for instance, a user who wants to buy a car and submits the query $\mathcal{Q}_1$ given in Section 3.1.1. A diverse result, i.e., a result that contains various prices and warranties with different horsepower and other technical characteristics, is intuitively more informative than a result that contains a homogeneous result containing only cars with similar features.

The diversity problem has attracted a lot of attention in the context of recommender systems, information retrieval and case-based reasoning systems. Some research works highlight that the diversity can be considered as important as similarity to the target query [McS02, ZMKL05]. Two main definitions of a set diversity are introduced: (i) average dissimilarity of all pairs of elements and (ii) average rarity of the elements in the set. Different similarity/dissimilarity and rarity measures were defined and used in different heuristic algorithms for computing the diversified set that maximizes the diversity without loss of similarity; e.g., see [DP10].

In the context of our top-k data service compositions approach, we challenge and tackle the lack of top-k data service compositions variety by proposing a method for maximizing the diversity of data service compositions while maintaining an acceptable satisfaction level (expressed in terms of fuzzy scores) of data service compositions. We propose to diversify the top-k data service compositions by firstly diversifying the top-k data services of each class $\mathcal{S}_i$, and then by diversifying the data service compositions themselves. The diversity of the top-k data services of a class $\mathcal{S}_i$ means that the data services should be dissimilar each other.

A principled way to improving diversity of the top-k data services of a class $\mathcal{S}_i$, while at the same time maintaining satisfaction of data services, is to explicitly use both diversity and satisfaction of data services during the top-k data services selection. To this end, we make use of the following quality metric that combines diversity and satisfaction:

$$Quality(s_{ij}) = DS_f(s_{ij}) \times RelDiv(s_{ij}, dtopk.\mathcal{S}_i) \tag{3.5}$$

The quality of a data service s_{ij} in its class $\mathcal{S}_i$ is proportional to its satisfaction, and to its relative diversity to those diversified top-k data services so far selected $dtopk.\mathcal{S}_i$. Initially, $dtopk.\mathcal{S}_i$ is an empty set, and its first element will be necessary one of the data services s_{ij} with higher DS_f. The relative diversity of a data service s_{ij} to the current set $dtopk.\mathcal{S}_i$ is defined as the average dissimilarity between data service s_{ij} and the so far selected data services [McS02] as described in the following equation:

$$RelDiv(s_{ij}, dtopk.\mathcal{S}_i) = \begin{cases} 1 & if\ dtopk.\mathcal{S}_i = \emptyset \\ \frac{\sum_{S_{i\ell} \in dtopk.\mathcal{S}_i} Dist(s_{ij}, s_{i\ell})}{|dtopk.\mathcal{S}_i|} & otherwise \end{cases} \tag{3.6}$$

The relative diversity of a data service s_{ij} to an initial empty set, i.e., $|dtopk.\mathcal{S}_i| = 0$, is set to 1. The quantity $Dist(s_{ij}, s_{i\ell})$ represents the distance (i.e., dissimilarity) measure between the two data services s_{ij} and $s_{j\ell}$. Recall that data services of the same class have the same functionality and only differ in their constraints, therefore the data services dissimilarity can be reduced to the dissimilarity of their constraints to quantify the extent to which two data services have similar constraints on their variables (i.e., they provide the same information about the same variable).

Given two data services $s_{ij}, s_{i\ell}$ having constraints $\mathcal{C}_{ij} = \{x_1 \text{ is } \mathcal{F}_1, ..., x_{d_i} \text{ is } \mathcal{F}_{d_i}\}$ and $\mathcal{C}_{i\ell} = \{x_1 \text{ is } \mathcal{G}_1, ..., x_{d_i} \text{ is } \mathcal{G}_{d_i}\}$, respectively. The distance between s_{ij} and $s_{i\ell}$ can be measured by $Dist(s_{ij}, s_{i\ell}) = max_{\imath \in \{1,...,d_i\}} Dist(\mathcal{F}_\imath, \mathcal{G}_\imath)$, where $Dist(\mathcal{F}_\imath, \mathcal{G}_\imath) = max_{x \in X_\imath} |\mu_{\mathcal{F}_\imath}(x) - \mu_{\mathcal{G}_\imath}(x)|$ represents the distance between the fuzzy sets $\mathcal{F}_\imath$ and $\mathcal{G}_\imath$ [DP00]. Of course, the distance between two fuzzy sets can be measured by others distance metrics. We provide the effects of the distance metric in Section 6.

3.4.3.1 Diversified Top-k Data Services Computing

The above quality measure guides the construction of the diversified top-k data services of each relevant service class $\mathcal{S}_i$. This construction is achieved in an incremental way as described in Algorithm 3.2; refereed to as DTKS. During each step, the remaining data services of a class $\mathcal{S}_i$ are rank-ordered according to their quality and the data service with the highest quality is added to $dtopk.\mathcal{S}_i$. The first data

service of the diversified top-k of a service class $\mathcal{S}_i$ to be selected is always the one with the highest DS_f. The initial service class $\mathcal{S}_i$ can be bounded to a smaller size equivalent to $k \cdot \eta$ ($\eta > 1$) to decrease the search space especially when $\mathcal{S}_i$ is too large. It is worth to note that for the service classes whose data services do not meet the user preferences, we just select randomly one data service, as they are all strictly similar.

Algorithm 3.2: DTKS

Input: $k \in \mathbb{N}$; $\eta \in \mathbb{N}$; $\mathcal{S}_i$ service class;

Output: $dtopk.\mathcal{S}_i$ diversified top-k data services of the class $\mathcal{S}_i$;

```
1 begin
2     S'_j ← top(k · η,S_i);
3     dtopk.S_i ← ∅;
4     for i=1 to k do
5         ComputeQuality(S'_i);
6         dtopk.S_i ← dtopk.S_i ∪ {MaxQuality(S'_i)};
7         S'_i ← S'_i − {MaxQuality(S'_i)};
8     return dtopkS_i;
```

3.4.3.2 Diversified Top-k Data Service Compositions Computing

The top-k data service compositions set is made more diverse (by applying a diversification on its component compositions) while maintaining acceptable compositions scores. The quality of a data service composition $\mathcal{CS}$ is an aggregation of qualities of its component services. Let $\mathcal{CS} = \{s_{1j_1}, ..., s_{nj_n}\}$ be a composition of n data services and $d = d_1 + ... + d_n$ be the number of user preferences involved in the query, where d_i is the number of constraints involved in the service s_{ij_i}. The quality of the composition $\mathcal{CS}$ is then computed using a weighted average as follows:

$$Quality(\mathcal{CS}) = \frac{1}{d} \sum_{i=1}^{n} d_i \cdot Quality(s_{ij_i}) \tag{3.7}$$

The diversified top-k data service compositions algorithm referred as DTKSC is obtained from TKSC (the top-k data service compositions algorithm) by applying the following modifications:

- Line 17 – For relevant service classes whose data services do not meet user preference, select randomly one data service instead of k data services as motioned above. So line 17 writes: $top\text{-}k.\mathcal{S}_i \leftarrow random(\mathcal{S}_i,1)$;
- Line 22 – Instead of taking the top-k data services in each class based on their scores, take them based on their qualities, i.e., take the diversified top-k ones, by applying Algorithm 2, so line 22 writes: $top\text{-}k.\mathcal{S}_i \leftarrow \text{DTKS}(k, \eta, \mathcal{S}_i)$;
- Line 27 – Compute the quality of the data service compositions instead of their scores. This line writes: ComputeCompositionQuality($\mathcal{CS}$);
- Line 29 – Instead of returning the top-k data service compositions, i.e., the top-k with the highest scores, return the diversified top-k ones, i.e., the ones having the best qualities. So line 29 writes: **return** Diversifiedtop($k, \mathbb{CS}$).

3.5 System Architecture and Experimental Evaluation

3.5.1 System Architecture

In this section, we outline the basic components of our implemented system, describe their roles and how they interact with each other. A high-level overview of our system is presented in Figure 3.5.

The *Fuzzy Membership Functions Manager* is used to manage fuzzy linguistic terms. It enables users and service providers to define their desired fuzzy terms along with the associated fuzzy membership functions. The defined terms are stored in a local fuzzy terms knowledge base which can be shared by users, and are linked to their implementing Web services. Examples of fuzzy terms along with their implementing services can be found on http://vm.liris.cnrs.fr:36880/FuzzyTerms. Users and service providers can directly test the proposed membership functions on that link and use the associated fuzzy terms. For each fuzzy term we provide a shape that gives a graphical representation of the associated membership function, a form that helps users to compute the degree to which a given value is in the fuzzy set of the considered fuzzy term, and a WSDL description of the Web service that implements the membership function.

The *Service Annotator* allows service providers to (i) define the functionalities of their data services in the form of RDF parameterized views (RPVs) [BBM10],

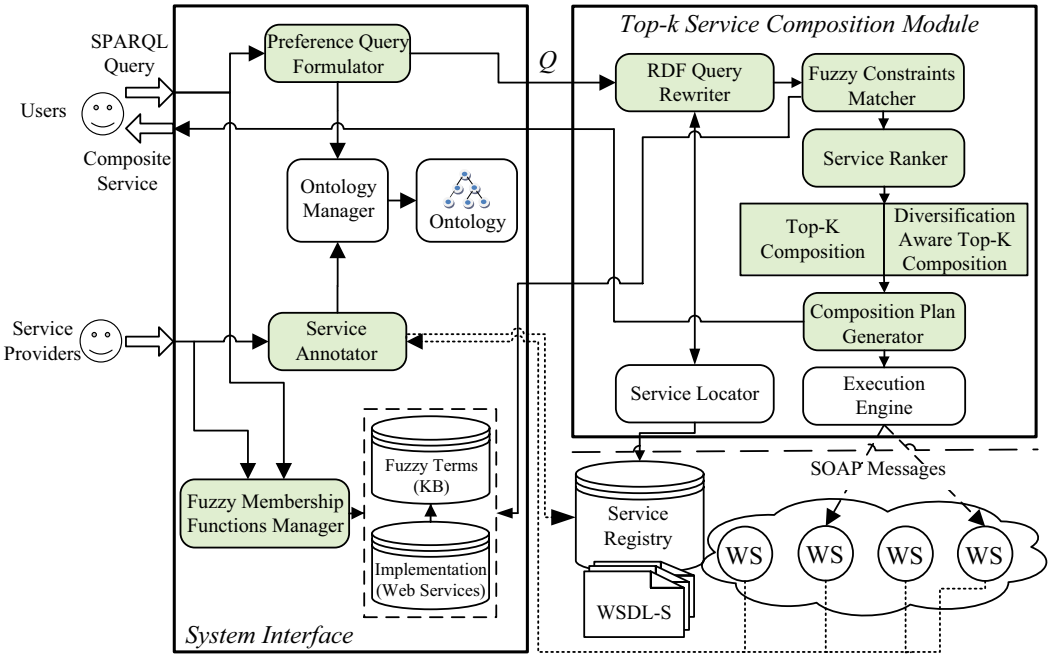


Figure 3.5: Data Service Composition Architecture

and specify the defined views with the desired fuzzy terms to represent the services' constraints, and (*ii*) annotate the services description files (e.g., WSDL files) with the defined views. This annotation is implemented by adding a new XML element called "rdfQuery" to the "Operation" elements in the XML Schema of WSDL as in the WSDL-S approach. The annotated WSDL files are then published to a service registry. The ontology manager uses Jena API to manage domain ontology (i.e., to add/delete concepts).

The *Preference Query Formulator* provides users with a GUI implemented with Java Swing to interactively formulate their queries over a domain ontology. Users are not required to have knowledge about SPARQL (or any specific ontology query languages) to express their queries, they are assisted interactively in formulating their queries and specifying the desired fuzzy terms.

The *Top-k Service Composition Module* consists of five components. The *RDF Query Rewriter* implements an RDF query rewriting algorithm [BBM10] to identify the relevant data services that match (some parts of) a user query. For that purpose, it exploits the annotations that were added to the service descrition files (e.g., WSDls). The *Service Locator* feeds the *Query Rewriter* with data services that

most likely match a given query. The *Top-k Composition* component computes (*i*) the matching degrees of relevant data services, (*ii*) the fuzzy dominating scores of relevant data services, (*iii*) the top-k data services of each relevant service class, and (*iv*) the fuzzy compositions scores to return the top-k data service compositions.

The *diversification-aware Top-k Compositions* component implements the proposed quality metrics to compute the diversified top-k data service compositions. The (diversified) top-k data service compositions are then translated by the *composition plan generator* into execution plans expressed in the XML Process Definition Language (XPDL)[8]. They are executed by a workflow execution engine. In our implementation, we use the Sarasvati[9] execution engine from Google.

3.5.2 Experimental Evaluation

This section presents an extensive experimental evaluation of our approach, focusing on : (*i*) the efficiency of our algorithms in terms of execution time, (*ii*) the effects of the used distance measure on the retrieved diversified top-k data service compositions, (*iii*) the effects of ε and λ on the top-k data service compositions/diversified top-k data service compositions and the benefits in terms of diversity, resulting from the use of the diversity aspect, and (*iv*) the effectiveness of the use of the fuzzy dominating score for ranking data services.

3.5.2.1 Experiment Setting

Due to the limited availability of real data services, we implemented a Web service generator. The generator takes as input a set of (real-life) model data services (each representing a class of services) and their associated fuzzy constraints and produces for each model service a set of synthetic data services and their associated synthetic fuzzy constraints. The generated data services satisfy some fuzzy constraints on the attributes of the implemented model service. The generation of the synthetic data services is controlled by the following parameters: (*i*) the number of candidate data services per service class, (*ii*) the number of service classes, (*iii*) the number of max preferences in a service class, (*iv*) the number of matching methods and (*v*) the values of the parameters k, ε and λ. The default values of these parameters are

[8] http://www.xpdl.org/

[9] http://code.google.com/p/sarasvati/

: 400, 4, 4, 4, 5, 0.02 and 0.2, respectively.

The service generator and the algorithms, i.e., TKSC and DTKSC, were implemented in Java, and the experiments were conducted on a Pentium D 2:4GHz with 2GB of RAM, running Windows.

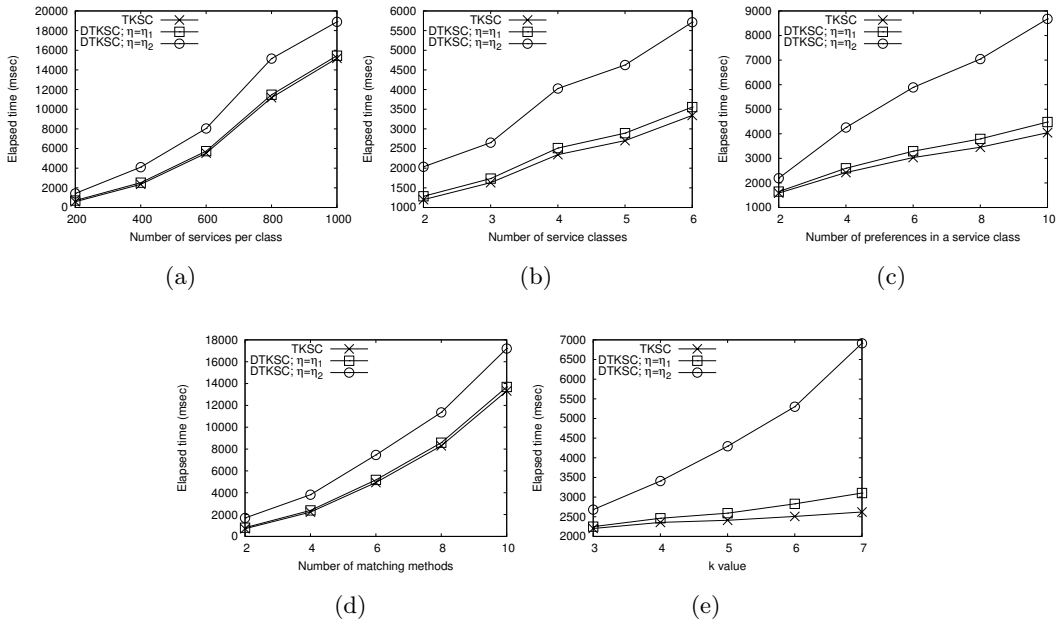


Figure 3.6: Performance Results; case of $\eta_1 = \left\lfloor \sqrt{\frac{|\mathcal{S}_j|}{k}} \right\rfloor$ and $\eta_2 = \left\lfloor \frac{|\mathcal{S}_j|}{k} \right\rfloor$

3.5.2.2 Performance vs Number of Services per Class

We measured the average execution time required to solve the composition problem as the number of data services per class increases. We varied the number of data services per class from 200 to 1000. The results of this experiment are presented in Figure 3.6a. The results show that our framework can handle hundreds of services in a reasonable time. The results also show that computing the diversified top-k composition introduces an insignificant cost when the factor η is small (e.g., $\eta = \eta_1$); this cost increases as η increases (e.g., $\eta = \eta_2$) since the search space for the diversified services in each class becomes larger.

3.5.2.3 Performance vs Number of Classes

We measured the average execution time required to solve the composition problem as the number of service classes increases. We varied the classes number from 1 to 6. The results of this experiment in Figure 3.6b show that the execution time is proportional to the classes number.

3.5.2.4 Performance vs Number of Constraints per Service

We varied number of fuzzy constraints from 2 to 10. Figure 3.6c shows the time required to compute the top-k/diversified top-k data service compositions. The results show that when the factor η is small (e.g., $\eta = \eta_1$) the cost incurred in computing the diversified top-k data service compositions is insignificant as the constraints number increases.

3.5.2.5 Performance vs Number of Matching Methods

We varied the number of matching methods from 1 to 10. The results of this experiment are shown in Figure 3.6d. Once again the cost incurred in computing the diversified top-k compositions remains insignificant as the methods number increases if the factor η has a reasonable value (e.g., $\eta = \eta_1$).

3.5.2.6 Performance vs k

The results in Figure 3.6e show that as k increases, the cost incurred in computing the diversified top-k data service compositions increases slightly relative to the time needed to compute the top-k data service compositions.

3.5.2.7 The Effect of the used Distance Measure

To compute the diversified top-k data service compositions we implemented all of the three distance measures:

$$M(\mathcal{F},\mathcal{G}) = \begin{cases} 0 & if \mathcal{F} = \mathcal{G} = \emptyset \\ 1 - \frac{\sum_{x \in X} min(\mu_{\mathcal{F}}(x), \mu_{\mathcal{G}}(x))}{\sum_{x \in X} max(\mu_{\mathcal{F}}(x), \mu_{\mathcal{G}}(x))} & otherwise \end{cases} \tag{3.8}$$

$$L(\mathcal{F},\mathcal{G}) = max_{x \in X} \, |\mu_{\mathcal{F}}(x) - \mu_{\mathcal{G}}(x)| \tag{3.9}$$

$$N(\mathcal{F},\mathcal{G}) = \begin{cases} 0 & if \mathcal{F} = \mathcal{G} = \emptyset \\ \frac{\sum_{x\in X}|\mu_{\mathcal{F}}(x)-\mu_{\mathcal{G}}(x)|}{\sum_{x\in X}(\mu_{\mathcal{F}}(x)+\mu_{\mathcal{G}}(x))} & otherwise \end{cases} \tag{3.10}$$

The membership functions used in computing the distance measures were discretized with a step of the order $(B + b - A + a)/1000$ (see Figure 2.2).

Table 3.5: The Effects of the used Distance Measure

Diversified Top-k Compositions				
Composite Services	Score	Quality		
		M	L	N
$\mathcal{CS}_1$: $\{s_{1356}, s_{2372}, s_{3285}, s_{4214}, s_{5183}\}$	0.6919484	0.6919484	0.6919484	0.6919484
$\mathcal{CS}_2$: $\{s_{1356}, s_{2372}, s_{3283}, s_{4214}, s_{5183}\}$	0.68804884	0.6744621	0.6615082	0.6780993
$\mathcal{CS}_3$: $\{s_{1356}, s_{2372}, s_{3360}, s_{4214}, s_{5183}\}$	0.69165516	0.6713853	0.6594182	0.6809209

Changing the used distance measure may change the quality of a composition, leading thus to its exclusion or inclusion to the diversified top-k compositions. Table 3.5 shows the diversified top-3 compositions of a given query along with their qualities when applying each of the previously seen distance measures. The composition $\mathcal{CS}_2$, for example, has a quality higher than that of $\mathcal{CS}_3$ if the distance measures M and L were applied; however its quality is lower than that of $\mathcal{CS}_3$ if the distance measure N was applied, thus leading to its exclusion if k was 2.

Table 3.6: Effects of ε and λ on the Top-k Compositions

(ε, λ)	Top-k Compositions		
	Component Services	Score	Diversity
$(0.002, 0.05)$	$\{s_{1318}, s_{2292}, s_{3154}, s_{4154}\}$	0.74703556	0.6121456
	$\{s_{1318}, s_{259}, s_{3154}, s_{4154}\}$	0.7441032	
	$\{s_{1318}, s_{2152}, s_{3154}, s_{4154}\}$	0.7441032	
$(0.02, 0.2)$	$\{s_{1318}, s_{2292}, s_{3154}, s_{4154}\}$	0.6563174	0.59373885
	$\{s_{1318}, s_{2132}, s_{3154}, s_{4154}\}$	0.655371	
	$\{s_{1318}, s_{259}, s_{3154}, s_{4154}\}$	0.65328693	
$(0.1, 0.3)$	$\{s_{1318}, s_{2292}, s_{3154}, s_{4154}\}$	0.53315574	0.62760955
	$\{s_{1318}, s_{2132}, s_{3154}, s_{4134}\}$	0.5312762	
	$\{s_{1318}, s_{2292}, s_{3154}, s_{4154}\}$	0.53008974	

Table 3.7: Effects of ε and λ on the Diversified Top-k Compositions

(ε, λ)	Diversified Top-k Compositions			
	Component Services	Quality	Score	Diversity
$(0.002, 0.05)$	$\{s_{1318}, s_{2292}, s_{3154}, s_{4154}\}$	0.74703556	0.74703556	0.6995363
	$\{s_{1318}, s_{2292}, s_{3154}, s_{4134}\}$	0.6972428	0.7426259	
	$\{s_{1318}, s_{2134}, s_{3154}, s_{4154}\}$	0.6972428	0.7426259	
$(0.02, 0.2)$	$\{s_{1318}, s_{2292}, s_{3154}, s_{4154}\}$	0.6563174	0.6563174	0.6995363
	$\{s_{1318}, s_{2292}, s_{3154}, s_{4134}\}$	0.612067	0.6519956	
	$\{s_{1318}, s_{2134}, s_{3154}, s_{4154}\}$	0.6098658	0.6515922	
$(0.1, 0.3)$	$\{s_{1318}, s_{2292}, s_{3154}, s_{4154}\}$	0.53315574	0.53315574	0.71135545
	$\{s_{1318}, s_{2292}, s_{3154}, s_{4134}\}$	0.49845165	0.5312762	
	$\{s_{1318}, s_{2134}, s_{3154}, s_{4155}\}$	0.49460968	0.5256555	

3.5.2.8 The Effects of ε and λ

Changing the used values of the parameters ε and λ change the scores and the qualities for both the top-k/diversified top-k data service compositions. This may consequently lead to the inclusion or to the exclusion of a composition from top-k/diversified top-k data service compositions. Table 3.6 and Table 3.7 show the top-k/diversified top-k data service compositions for different values of ε and λ; the higher the values of these parameters are, the higher the global diversity of the diversified top-k compositions is. The global diversity of the diversified top-k compositions set described in Equation 3.11 is the average of the diversities between each couple of compositions in the compositions set. Note that the global diversity of the diversified top-k compositions is always higher than that of the top-k compositions and the applicability of $DTKSC$ produce an average gain of $9,22\%$.

$$div(\textit{top-}k) = \frac{\sum_{i=1}^{k} \sum_{j=i+1}^{k} Dist(\mathcal{CS}_i, \mathcal{CS}_j)}{(k^2 - k)/2} \tag{3.11}$$

3.5.2.9 Effectiveness of the Fuzzy Dominating Score

To evaluate the quality of results returned by applying our approach, we have focussed on one service class $\mathcal{S}_0$ containing a small set of 100 data services. We have considered 3 matching methods M_1, M_2 and M_3 and 2 preferences involved in this class of services. For comparison, we also computed the top-k data service in this

service class by applying the Pareto dominating score proposed in [SSS+09, SSSS10].

Table 3.8: Top-5 Data Services using Pareto Dominating Score and Fuzzy Dominating Score

Data service	Matching degrees			Rank	
	M_1	M_2	M_3	DS	DS_f
s_{04}	$(0.6443, 0.7146)$	$(0.5761, 0.8961)$	$(0.7063, 0.8739)$	3	2
s_{09}	$(0.8010, 0.6494)$	$(0.6462, 0.8378)$	$(0.7112, 0.9996)$	2	1
s_{22}	$(0.0454, 0.4498)$	$(0.7529, 0.9747)$	$(0.8894, 0.8827)$	4	–
s_{057}	$(0.8508, 0.9447)$	$(0.3884, 0.1678)$	$(0.9576, 0.9885)$	1	5
s_{072}	$(0.8809, 0.9661)$	$(0.3884, 0.1678)$	$(0.9934, 0.3117)$	5	3
s_{093}	$(0.8508, 0.9447)$	$(0.8963, 0.8598)$	$(0.7112, 0.9996)$	–	4

Table 3.8 lists the top-5 data services using Pareto dominating score (DS) and fuzzy dominating score (DS_f). Table 3.8 shows that almost all the top-5 with respect to DS are also in the top-5 with respect to DS_f except for s_{22} witch is replaced by s_{093}. This is because s_{22} is very bad according to M_1, in particular for the first constraint. In addition, Table 3.8 shows that the rank of the data services s_{04}, s_{09}, s_{057} and s_{072} is different in the two top-5 sets. s_{057}, the best data service with respect to DS is ranked last (i.e., fifth) with respect to DS_f. On the other side, the data services s_{04}, s_{09} and s_{072} are in the top-3 (with respect to. DS_f). This is because s_{057} is very bad according to M_2, in particular, for the second constraint. However, s_{04}, s_{09} and s_{072} are good or moderately good according to all matching methods. This is consistent with our motivation to fuzzify the Pareto dominance relationship illustrated in Section 3.3.1.

3.6 Conclusion

In this paper, we have proposed an approach to compute the top-k data-driven Web service compositions for the purpose of answering fuzzy preference queries. We have introduced the concept of fuzzy dominance relationship, and proposed the fuzzy dominating score to measure to what extent a data service dominates another one. This new score allowed us to rank-order candidate services in their respective classes and to compute the top-k compositions. An algorithm is developed for this purpose. We have also proposed a new quality metric to assess the diversity of a composition

relative to a set of compositions and an algorithm to select the diversified top-k compositions based on the proposed quality metric. We have evaluated thoroughly our proposed composition algorithms on a large set of data-driven Web services and reported their performances.

Chapter 4

Majority-Rule-Based Web Service Selection

Contents

4.1 Introduction

Web data services, as a key technology for the development, deployment and management of Web services-based access to information systems, promise to enable maximal mashup, reuse, and sharing of structured data (e.g., relational tables), semi-structured information (e.g., XML documents) and unstructured information (e.g., commercial data from online business sources). Thereby, enabling users to perform several operations, e.g., data analysis, searches, purchases.

Consequently, it becomes apparent that the Web services paradigm rapidly gains popularity constituting an integral part of many real-world applications. For these reasons, several techniques for discovering Web services have been recently proposed. However, as Web data services (or services for short) and service providers proliferate, there will be a large number of candidate, most likely competing, services for fulfilling a desired task. Thus, *service selection* is becoming important for helping users to identify desirable services. User preferences play a key role during the selection process [WXL08]. However, in many practical situations, the responsibility to decide which is the appropriate service is shared among multiple parties, e.g., among the department heads of a university.

The following running example illustrates such a scenario, where a university decides to obtain a software license of a cloud-based data analytics service.

4.1.1 Motivating Example

Consider a set of cloud-based data analytics Web services, and assume that several departments within a university wish to buy a license for one of them. The services are described by their annual *Cost*, the number of allowed simultaneous *Processes*, the level of data *Redundancy*, and the number of computing *Nodes*.

Table 4.1: User Preferences

User	Budget	Processes	Redundancy	Nodes
u_1	$[7000, 10000]$	$[5, 10]$	–	–
u_2	–	–	$[3, 5]$	–
u_3	–	$[8, 12]$	–	$[80, 100]$

The users, in this case the department heads, have different preferences with respect to the service descriptions, as depicted in Table 4.1. User u_1, has a budget of $[7000, 10000]$ and expects to run simultaneously $[5, 10]$ processes; user u_2 cares much about data redundancy and expects a redundancy level of $[3, 5]$; user u_3 expects to run simultaneously $[8, 12]$ processes requiring $[80, 100]$ computing nodes.

The service selection process follows two phases. In the first, given the user's preferences on service description attributes, the degrees of match between a requested and an available service (see e.g., [PKPS02, LH03, DHM+04]) are computed. In this

work, we assume the Jaccard coefficient for matching service descriptions. If I_1, I_2 are two intervals, their Jaccard coefficient is $J(I_1, I_2) = \frac{|I_1 \cap I_2|}{|I_1 \cup I_2|}$, where $|I|$ measures the length of the interval [DH73].

Table 4.2: Discovered Services

Service	Cost	Processes	Redundancy	Nodes
s_1	$[7000, 11000]$	$[7, 12]$	$[3.5, 5.5]$	$[60, 110]$
s_2	$[5000, 10000]$	$[5, 11]$	$[4, 6]$	$[70, 115]$
s_3	$[6000, 12000]$	$[1, 10]$	$[4, 6]$	$[70, 110]$
s_4	$[8000, 12000]$	$[2, 12]$	$[3.5, 5]$	$[75, 130]$
s_5	$[9000, 15000]$	$[9, 12]$	$[4, 7]$	$[90, 130]$

Returning to our example, consider that the set of relevant Web services are the ones depicted on Table 4.2. Each service is shown along with its description attributes. For instance service s_1 offers license plans that cost $[7000, 11000]$ per year, allows $[7, 12]$ simultaneous processes, offers a redundancy level of $[3.5, 5.5]$, and allocates $[60, 110]$ computing nodes.

Based on the set of relevant service in Table 4.2 and the user requirements in Table 4.1, the service selection process computes the matching degrees between each user's specified preference and the corresponding service characteristic. Table 4.3 shows the matching degrees of discovered service with respect to users preferences. For instance, the matching degree of service s_1 with respect to the cost and processes requirements of user u_1 are $\frac{|[7000,10000]|}{|[7000,11000]|} = 0.75$ and $\frac{|[7,10]|}{|[5,12]|} = 0.43$, respectively.

Table 4.3: Matching Degrees of Services with respect to Users Preferences

Service	u_1: (Cost, Processes)	u_2: Redundancy	u_3: (Processes, Nodes)
s_1	$(0.75, 0.43)$	0.62	$(0.83, 0.41)$
s_2	$(0.67, 0.86)$	0.35	$(0.57, 0.47)$
s_3	$(0.57, 0.54)$	0.35	$(0.23, 0.51)$
s_4	$(0.50, 0.54)$	0.76	$(0.45, 0.38)$
s_5	$(0.57, 0.25)$	0.27	$(0.80, 0.27)$

The second phase of service selection is to identify the most interesting services

with respect to users preferences. Most of service selection approaches focus on computing a score for each service as an aggregate of its individual matching degrees. Various approaches for aggregating the matching degrees exist. A common direction is to assign weights over different preference attributes; e.g., [LASG07]. However, when multiple users are involved, it would be difficult to make tradeoffs between different weights. The natural option is to use the skyline operator [YB10a, ASR10, YB10b, YB12] to determine an objectively good set of services. We refer to this set as the *unanimous service skyline*, and it contains all services which are not unanimously dominated. A service *unanimously dominates* another, if the former is higher than or equal to the latter in all users' preferences and (strictly) higher in at least one.

In our example, service s_1 unanimously dominates service s_5, as s_1's matching degrees are higher. On the other hand, no other service is unanimously dominated. Hence, the *unanimous service skyline* comprises services s_1, s_2, s_3 and s_4.

Computing the unanimous service skyline frees users from assigning relative importance over different preference attributes. However, a major drawback is that, when multiple parties are involved, the number of services in the skyline becomes very large and no longer offers any interesting insights. The reason is that as the number of users and preferences increase, for any services s_i, s_j, it is more likely that s_i and s_j are incomparable, i.e., better than each other in different matching degrees. It is thus crucial to further reduce the size of the service skyline.

4.1.2 Contributions

The core of the above drawback is in the definition of dominance, which requires a unanimous verdict. To mitigate this, we choose to follow the majority rule. Informally, a service *majority-dominates* another, if the former is higher than or equal to the latter in the majority of users' preferences and higher in at least one (in this majority of users' preferences). Then, we naturally define the *majority service skyline*, as the services which are not majority-dominated.

To compute the majority service skyline, we make the observation that conventional skyline computation algorithms, with the exception of [CJT+06a], cannot be adapted, due to the intransitivity of the majority-dominance relationship. Therefore, an extension of the algorithms in [CJT+06a] can be used to compute the

majority service skyline. However, we propose a novel algorithm for the service selection problem and show that it most cases it outperforms the extended algorithms. Our main contributions in this Chapter are summarized as follows:

- We introduce a new concept for service selection when multiple preferences are involved, which is based on the majority rule, and is called the *majority service skyline*;
- We extend existing algorithms and propose a novel algorithm to efficiently compute the majority service skyline;
- We evaluate both the effectiveness of the proposed concept and the efficiency of our algorithm through a comprehensive experimental study.

The rest of the Chapter is structured as follows. Section 4.2 introduces the problem of majority service skyline. Section 4.3 describes the majority service skyline computation algorithm. Section 4.4 presents our experimental study. Finally, Section 4.5 concludes the Chapter.

4.2 Problem Definition

In this section, we provide the basic notions used throughout this paper, and formalize the notion of *majority service skyline*.

We assume a set of users $\mathcal{U} = \{u_1, u_2, \ldots, u_m\}$, and a set of discovered services $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$. We use $s_i.u_k$ to denote the matching degrees of service s_i with respect to user u_k. For instance, the matching vector of service s_1 with respect to user u_1 is $s_1.u_1 = (0.75, 0.43)$.

Definition 4.1 *(Weak Dominance)*
Given a user u_k, we say that service s_i weakly dominates service s_j with respect to u_k, denoted as $s_i.u_k \succeq s_j.u_k$, if and only if s_i is better than or equal to s_j on all specified preference attributes.

Definition 4.2 *(Dominance)*
Given a user u_k, we say that service s_i dominates service s_j with respect to u_k, denoted as $s_i.u_k \succ s_j.u_k$, if and only if s_i is better than or equal to s_j on all specified preference attributes, and better on at least one.

Definition 4.3 *(Unanimous Dominance)*
Given a set of users $\mathcal{U}$, we say that service s_i unanimous-dominates service s_j, denoted as $s_i \succ_U s_j$, if and only if s_i weakly dominates s_j with respect to all users, i.e., $\forall u_k \in \mathcal{U} : s_i.u_k \succeq s_j.u_k$, and there exists one user, say u_ℓ, for which s_i dominates s_j, i.e., $\exists u_\ell \in \mathcal{U} : s_i.u_\ell \succ s_j.u_\ell$.

Definition 4.4 *(Unanimous Service Skyline)*
Given a set of a set of users $\mathcal{U}$ and discovered services $\mathcal{S}$, the unanimous service skyline of $\mathcal{S}$ with respect to $\mathcal{U}$ denoted as $USS(\mathcal{S},\mathcal{U})$ comprises the set of services in $\mathcal{S}$ that are not unanimous-dominated by any other service in $\mathcal{S}$, i.e., $USS(\mathcal{S},\mathcal{U}) = \{s_i \in \mathcal{S} \mid \nexists s_j \in \mathcal{S} : s_j \succ_U s_i\}$.

In the following, we introduce the concept of majority rule in the service selection process and alter the definitions of dominance and skyline.

Definition 4.5 *(Majority Dominance)*
Given a set of users $\mathcal{U}$, we say that service s_i majority-dominates service s_j, denoted as $s_i \succ_M s_j$, if and only if (i) there exists a subset $\mathcal{U}' \subseteq \mathcal{U}$ containing more than half of the users such that s_i weakly dominates s_j with respect to all users in this subset, i.e., $|\mathcal{U}'| > \lfloor |\mathcal{U}|/2 \rfloor$ and $\forall u_k \in \mathcal{U}' : s_i.u_k \succeq s_j.u_k$, and (ii) there exists one user, say u_ℓ, for which s_i dominates s_j, i.e., $\exists u_\ell \in \mathcal{U} : s_i.u_\ell \succ s_j.u_\ell$.

Definition 4.6 *(Majority Service Skyline)*
Given a set of a set of users $\mathcal{U}$ and discovered services $\mathcal{S}$, the majority service skyline of $\mathcal{S}$ with respect to $\mathcal{U}$ denoted as $MSS(\mathcal{S},\mathcal{U})$ comprises the set of services in $\mathcal{S}$ that are not majority-dominated by any other service in $\mathcal{S}$, i.e., $MSS(\mathcal{S},\mathcal{U}) = \{s_i \in \mathcal{S} \mid \nexists s_j \in \mathcal{S} : s_j \succ_M s_i\}$.

Returning to our running example, s_2 majority-dominates services s_3, s_4 and s_5, while, services s_1 and s_2 are not majority-dominated by any other service. Thus, services s_1 and s_2 form the majority service skyline. Recall that the unanimous service skyline comprises services s_1, s_2, s_3 and s_4. Observe that the MSS has smaller cardinality than the USS. Thus, users can make a good, quick, selection.

We now provide the formal definition for the service selection problem for multiple users.

Problem statement: Given a set of users $\mathcal{U}$ and a set of discovered services $\mathcal{S}$, compute the *majority service skyline*.

4.3 Computing the Majority Service Skyline

In this section, we first introduce some important observations regarding the problem at hand. We then develop an algorithm based on these observations for efficiently computing the majority service skyline.

4.3.1 Observations

Next, we make some observations regarding the majority dominance relationship.

Lemma 4.1

If service s_i unanimous-dominates service s_j, then s_i majority-dominates s_j, i.e., $s_i \succ_U s_j \Rightarrow s_i \succ_M s_j$.

Proof

Proof follows from Definition 4.3 and Definition 4.5, setting $\mathcal{U}' = \mathcal{U}$. □

Theorem 4.1

The majority service skyline is a subset of the unanimous service skyline, i.e., $MSS(\mathcal{S},\mathcal{U}) \subseteq USS(\mathcal{S},\mathcal{U})$.

Proof

Assume that there exists a service s_i, such that $s_i \in MSS(\mathcal{S},\mathcal{U})$ and $s_i \notin USS(\mathcal{S},\mathcal{U})$. Since $s_i \notin USS(\mathcal{S},\mathcal{U})$, there must exist a service s_j, such that $s_j \succ_U s_i$. Thus, by Lemma 4.1, we have $s_j \succ_M s_i$. Which leads to a contradiction, as $s_i \in MSS(\mathcal{S},\mathcal{U})$. □

Moreover, observe that the majority dominance relationship does not maintain the transitive property of the unanimus dominance relationship, as discovered services can exhibit a *cyclic majority dominance relationship*.

Theorem 4.2

It is possible to have a set of users $\mathcal{U} = \{u_1, u_2, \ldots, u_m\}$ and a set of discovered services $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ such that s_1 majority-dominates s_2, s_2 majority-dominates

Table 4.4: Example of Cyclic Majority Dominance

Service	u_1: (Cost, Processes)	u_2: Redundancy	u_3: (Processes, Nodes)
s_a	$(0.76, 0.69)$	0.74	$(0.58, 0.80)$
s_b	$(0.56, 0.64)$	0.70	$(0.78, 0.86)$
s_c	$(0.80, 0.88)$	0.68	$(0.72, 0.76)$
s_d	$(0.78, 0.86)$	0.61	$(0.75, 0.89)$

$s_3, \ldots, s_{n-1}$ majority-dominates s_n and s_n majority-dominates s_1, i.e., forming a cyclic majority dominance relationship.

Proof

The example in Table 4.4, where $s_a \succ_M s_b$, $s_b \succ_M s_c$, $s_c \succ_M s_d$, and $s_d \succ_M s_a$, proves the claim. □

The above theorem shows that the majority dominance relationship shares the cyclic property of the k-dominance relationship introduced in [CJT+06a]. Therefore, a service cannot be discarded even if it is majority-dominated because it might be needed for excluding other services. This justifies why the existing algorithms for computing the skyline are not applicable for computing the majority service skyline. However, the one scan algorithm (OSA) and two scan algorithm (TSA) of [CJT+06a], can be adapted to compute the majority service skyline, by exchanging k-dominance checks for majority dominance checks as defined in Section 4.2. In the following, we denote as OSA and TSA the adaptations of the algorithms in [CJT+06a] to computing the majority service skyline.

4.3.2 Majority Service Skyline Algorithm

Next, we introduce the *Majority Service Skyline Algorithm* (MSA), which improves on OSA by employing the following properties.

Lemma 4.2

If service s_i unanimous-dominates service s_j and s_j majority-dominates service s_k, then s_i majority-dominates s_k, i.e., $s_i \succ_U s_j \wedge s_j \succ_M s_k \Rightarrow s_i \succ_M s_k$.

Proof

As s_i unanimous-dominates s_j means that s_i weakly dominates s_j with respect to all

users, and there exists a user for which s_i dominates s_j; and s_j majority-dominates s_k means that s_j weakly dominates s_k with respect to more than half of users, and there exists a user for which s_j dominates s_k, we have s_i weakly dominates s_k with respect to more than half of users, and there exists a user for which s_i dominates s_k since the dominance relationship is transitive. Hence, s_i majority-dominates s_k. □

Lemma 4.3

Let $f : \mathcal{S} \rightarrow \mathbb{R}^+$ be a monotone function aggregating the matching degrees of service s_i for all users. If s_i unanimous-dominates service s_j, then $f(s_i) > f(s_j)$, i.e., $s_i \succ s_j \Rightarrow f(s_i) > f(s_j)$.

Proof

The fact that s_i unanimous-dominates s_j means that s_i is better than or equal to s_j with respect to all preference attributes of all users. This implies that a monotone aggregate function over the matching degrees of s_i has a greater value than that function over the matching degrees of s_j. Hence, $f(s_i) > f(s_j)$. □

From Lemma 4.1 and Lemma 4.2, we can see that it is sufficient to compare each service against the unanimous skyline services to detect if it is part (or not) of the majority service skyline. This essentially reduces the number of comparisons. Specifically, *if a service s_i is unanimous-dominated, then discard it as (i) it is not part of the majority service skyline (Lemma 4.1), and (ii) it is unnecessary for eliminating other services (Lemma 4.2).*

Lemma 4.3 also helps reduce unnecessary comparisons. In fact, to exploit this property, we sort the services in non-ascending order of the sum of their matching degrees. Then, given a service s_i, searching for services by which s_i is unanimous-dominated can be limited to the part of the service before s_i. This is the idea behind the SFS algorithm [Cho03], which in this context we apply it for cyclic dominance relationships.

The MSA algorithm leverages the observations made above to compute efficiently the majority service skyline. Based on Lemma 4.1 and Lemma 4.2, MSA maintains two sets $\mathcal{R}$ and $\mathcal{T}$, containing respectively the set of intermediate majority skyline services and the set of intermediate unanimous skyline services that are not in $\mathcal{R}$. Thus, $\mathcal{R} \cup \mathcal{T}$ constitutes the intermediate unanimous skyline.

Algorithm 4.1: MSA

Input: *set of users* $\mathcal{U}$*; set of discovered services* $\mathcal{S}$*;*

Output: *majority service skyline* $\mathcal{R}$*;*

```
1  begin
2      sort S in a non-ascending order of the sum of services' matching degrees;
3      R ← ∅; T ← ∅;
4      while S is not empty do
5          extract the top service s_i from S;
6          if s_i is unanimous-dominated by any service in R ∪ T then
7              discard s_i;
8          else
9              if s_i majority-dominates any service s_j in R then
10                 remove s_j from R to T;
11             if s_i is majority-dominated by any service in R ∪ T then
12                 insert s_i into T;
13             else
14                 insert s_i into R;
15     return R;
```

The details of MSA depicted in Algorithm 4.1 are as follows. First, services in $\mathcal{S}$ are sorted in a non-ascending order of the sum of their matching degrees, and both sets $\mathcal{R}$ and $\mathcal{T}$ are initialized to empty sets. Then, the top service (i.e., the service with the maximum sum of matching degrees), say s_i, is extracted from $\mathcal{S}$. Service s_i is compared against services in $\mathcal{R} \cup \mathcal{T}$, i.e., the set of services that may unanimous-dominate s_i (as the other services cannot dominate s_i from Lemma 4.3). If s_i is unanimous-dominated, then it is removed from $\mathcal{S}$ as it is not part of the majority service skyline (Lemma 4.1) and it is unnecessary for eliminating other services (Lemma 4.2). Otherwise, i.e., when s_i is not unanimous-dominated by any service in $\mathcal{R} \cup \mathcal{T}$, if s_i majority-dominates any service s_j in $\mathcal{R}$ (i.e., s_j is not part of the majority service skyline), then s_j is removed from $\mathcal{R}$ to $\mathcal{T}$, as it is a unanimous skyline service, thus useful for eliminating other services. For the same reason, if s_i is *majority-dominated* by any service in $\mathcal{R} \cup \mathcal{T}$, it is inserted into $\mathcal{T}$ as it is not part of the majority service skyline. Else, s_i is an intermediate majority skyline service

and is thus inserted into $\mathcal{R}$. Once all services in $\mathcal{S}$ have been examined, i.e., $\mathcal{S}$ is empty, services in $\mathcal{R}$ form the majority service skyline, and $\mathcal{R}$ is returned.

Applying MSA on our example, services s_1 and s_2 will be inserted into $\mathcal{R}$, while, services s_3 and s_4 will be inserted into $\mathcal{T}$ since they are both *majority-dominated* by service s_1, but they are unanimous skyline services. On the other hand, service s_5 is discarded as it is dominated by service s_1. Thus, the algorithm correctly returns services s_1 and s_2 as the majority service skyline.

4.4 Experimental Evaluation

In this section, we present an extensive experimental evaluation of our approach. Our objective is to prove the effectiveness of the majority service skyline and the efficiency of the proposed algorithm. More specifically, we focus on two issues: *(i)* the size of the majority service skyline (denoted as MSS). To demonstrate that the majority service skyline further reduces the size of the (traditional) service skyline, we also compute the size of the unanimous service skyline (denoted as USS) to compare how their sizes varies; and *(ii)* the performance of our algorithm in terms of elapsed time for computing the majority service skyline. For comparison purposes, we also implemented the adaptations of OSA and TSA [CJT+06a] for computing the majority service skyline.

4.4.1 Experimental Setup

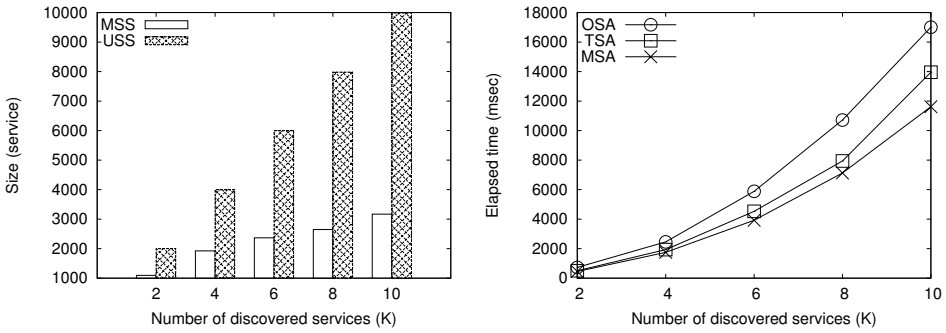
It is worth noting that, due to the limited availability of real-world service data, most existing skyline-based service selection approaches, e.g., [YB10a, YB10b, YB12], use synthetic datasets for their evaluation. For ease of comparison, we also follow this direction. The service generator we use takes as input a (real-world) model service and its associated constraints, representing the requested service and the multiple users preferences, and produce a set of synthetic services, as well as their associated constraints, representing the set of discovered services. The Jaccard coefficient is used for computing the matching degrees between discovered service' constraints and users preferences. The generation of the sets of synthetic services is controlled by the parameters in Table 4.5, which displays the parameters under investigation, their corresponding ranges and their default values. In each experimental setup, we

investigate the effect of one parameter, while setting the remaining ones to their default values.

Table 4.5: Parameters and Examined Values

Parameter	Symbol	Range	Default
Number of discovered services	n	$[2, 10]$K	5K
Number of users	m	$[3, 7]$	5
Number of preferences per user	d	$[3, 7]$	5

The service generator and the algorithms, i.e., MSA, OSA and TSA were implemented in Java, and all experiments were conducted on a 2.3 GHz Intel Core i5 with 8GB of RAM, running Mac OS X.



(a) Size

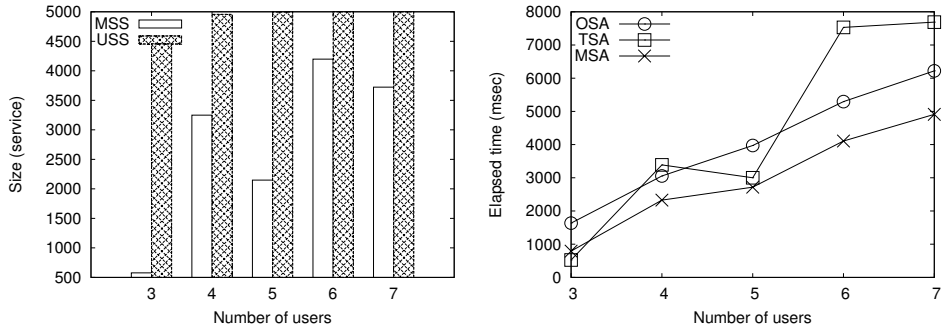
(b) Elapsed time

Figure 4.1: Effects of n.

4.4.2 Effect of Number of Discovered Services

Figure 4.1 depicts the effect of n. As shown in Figure 4.1a, the size of the majority service skyline increases slightly with n. This is because as n varies, it is becoming more difficult to find services which are majority-dominated. Figure 4.1a shows also that the size of the majority service skyline is very smaller then that of the skyline, which is almost equal to the number of discovered services, as the skyline cannot discard all inappropriate services, while the majority service skyline includes only the most interesting ones. On the other hand, Figure 4.1b shows that the execution time of the algorithms increases with n. However, MSA consistently outperforms

OSA and TSA.



(a) Size (b) Elapsed time

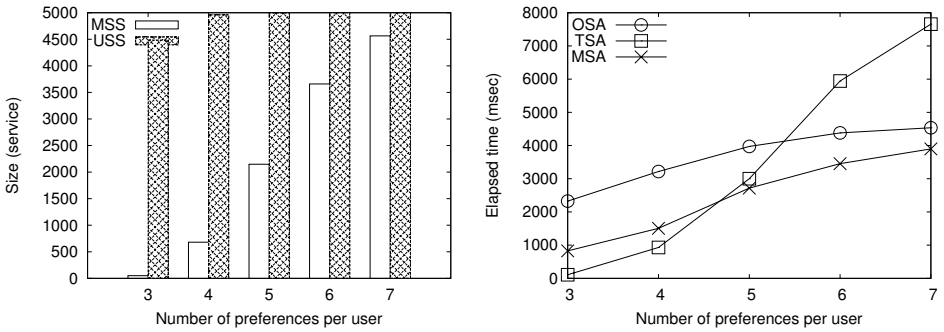
Figure 4.2: Effects of m.

4.4.3 Effect of Number of Users

Figure 4.2 shows the effect of m. Figure 4.2a shows a fluctuation in the size of the majority service skyline. The fluctuation is related to the definition of the majority dominance relationship (Definition 4.5). Indeed, we can distinguish two trend. One for the even values of m, and the second for the odd values of m; each trend increases with the increase of m. This is because, if we have an odd value of m, say m_o, and an even value of m, say m_e, such that $m_o = m_e + 1$, then the percentage of most of users for m_e is greater than that of m_o. For example, for $m = 4$, the percentage is $\frac{3}{4} = 0.75\%$, and for $m = 5$ the percentage is $\frac{3}{5} = 0.60\%$. When this percentage is large, small number of services is discarded, and vice versa. Also, note that the size of the majority service skyline is very smaller then that of the unanimous service skyline, which approximates the number of discovered services for $m \geq 4$. As shown in Figure 4.2b, when m increases, the performance of TSA deteriorates due to the second scan performed. However, the execution time of OSA and MSA increases slightly with m. Still, MSA is better.

4.4.4 Effect of Number of Preferences per User

Figure 4.3 shows the effect of d. As depicted in Figure 4.3a, the size of the majority service skyline increases significantly with the increase of d. This is because as d increases, a service has increased probability not to be dominated in all preference



(a) Size (b) Elapsed time

Figure 4.3: Effects of d.

attributes with respect to a given user. However, the size of the majority service skyline remains smaller than that of the unanimous service skyline, which approximates the number of discovered services for $d \geq 4$. As shown in Figure 4.3b, TSA is better than OSA and MSA for $d \leq 4$ since the size of the majority service skyline is small, thus a large number of services can be eliminated in the first scan. However, TSA does not scale with d as the size of the majority service skyline becomes large, thus the second scan is very time consuming. The execution time of OSA and MSA, on the other hand, increases slightly with d. Also, observe that MSA consistently performs better than OSA.

4.5 Conclusion

In this chapter, we dealt with the problem of preference-based Web service selection under multiple users preferences. We introduce a novel concept called majority service skyline for this problem based on the majority rule. This allows users to make a "democratic" decision on which Web services are the most appropriate. We develop a suitable algorithm for computing the majority service skyline. Our experimental evaluation demonstrates the effectiveness of the introduced concept and the efficiency of the proposed algorithm.

Chapter 5

Computing Skyline Web Services using Fuzzy Dominance

Contents

5.1 Introduction

Recently, there has been a large flow of Web services deployed over the Web, due to the acceptance of Service Oriented Architecture (SOA) as the solution to interoperation, reuse and globalization [PvdH07, YLBM08]. As outlined in Section 1.1 the statistics published by the Web services search engine seekda! indicates an exponential increase in the number of accessible Web services over the last 67 months,

and according to [AMM08], there has been more than 130% growth in the number of published Web services between October 2006 and October 2007.

As the Web is populated with a large number of Web services, there may be multiple service providers competing to offer the same functionality, but with different QoS (quality of service) such as latency, price and reputation. QoS is thus a crucial criterion to select among functionally similar Web services.

The following example illustrates a typical scenario related to our discussion, where users want to search an hotel and make an on-line reservation.

5.1.1 Motivating Example

Consider the common example in the literature concerning a set of Web services that provide hotel search and on-line reservation. For each Web service, Figure 5.1 sets its execution time and its price, where values of both QoS attributes are normalized in the range $[0, 1]$; to allow for an uniform measurement of service qualities independent of units.

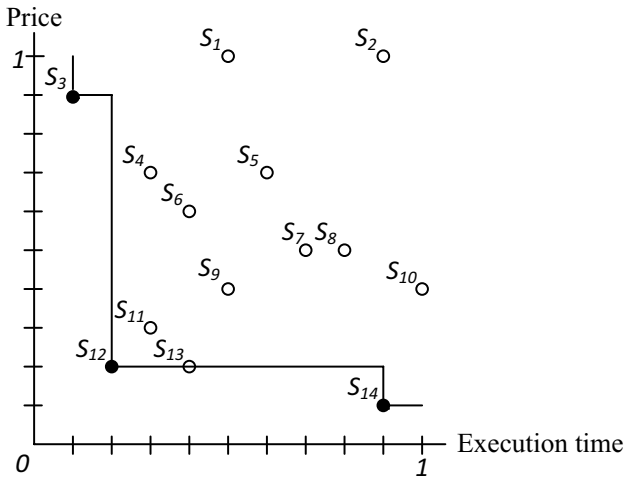


Figure 5.1: Example of Functionally Similar Web Services

To find satisfactory hotels, users need to go through several trial-run processes. This would be very painstaking as the number of competing providers is expected to be very large, and the selected Web services are not necessarily among the most interesting ones. Therefore, optimization strategies are required for finding the best services with respect to a set of QoS aspects desired by the users.

Currently, most approaches that deal with Web service selection based on QoS, compute a global QoS value for each service as an aggregate of the individual QoS values. Various approaches for combining QoS exist. One direction is to assign weights to individual QoS attributes. However, users may not know enough how to make tradeoffs between different quality aspects using numbers. They thus lose the flexibility to select their desired Web services.

Computing the service skyline [ASR10, YB10a, YB10b, YB12] comes as a popular solution that overcomes this limitation. The service skyline consists of a set of services which are not dominated by any other one. A service S_i dominates another service S_j if and only if S_i is at least as good as S_j in all QoS attributes and (strictly) better than S_j in at least one QoS attribute. For instance, the Web service S_3 in Figure 5.1 is better than the Web services S_1 and S_2 since it is faster and cheaper. The skyline of the set of services in Figure 5.1 comprises services S_3, S_{12} and S_{14} since they are not dominated by any other service.

However, there are some inherent issues of applying the service skyline approach. The first issue is related to the nature of retrieved services in the service skyline that privileges services with some very good and very bad QoS values like the services S_3 and S_{14} in Figure 5.1. Such services are referred to as services with a *bad compromise* (between QoS attributes). Whereas, users usually prefer services that are (moderately) good in all QoS attributes like the services S_{12} and S_{11} in Figure 5.1, where the last one is unfortunately not returned by the service skyline approach. The services of this type are referred to as services with a *good compromise* (between QoS attributes). Clearly, the Web service S_{12} is better than S_{11}. Furthermore, S_{12} may be currently unavailable or temporarily deprived of a functionality (e.g., online reservation). Users thus have to choose between the services S_3 and S_{14} while several services like S_{11} and S_{13} may be more appropriate. The second issue concerns the fact that the service skyline approach does not allow users to control the size of the returned set of services. With the presence of a possibly large number of skyline services, the full service skyline may be less informative. Thus, it may be hard for users to make a good, quick selection by scanning the entire skyline that consists of too many services. Also, with the presence of a small number of skyline services, users may lose interesting dominated services; knowing that some interesting services may be unavailable.

5.1.2 Contributions

In this chapter, we address the above mentioned issues by considering a fuzzy dominance relationship between Web services based on their QoS attributes. Our main contributions can be summarized as follows:

- We introduce a novel concept, called α-*dominant* service skyline, to tackle the problem of QoS-based web service selection;
- We develop a suitable algorithm, which leverages pruning techniques to efficiently compute the α-*dominant* service skyline;
- We evaluate both the effectiveness of the proposed concept and the efficiency of the algorithm through a set of experiments.

The rest of this chapter is structured as follows. Section 5.2 provides the formal definition and analysis of the α-*dominant* service skyline. Section 5.3 describes the α-*dominant* service skyline computation algorithm. Section 5.4 presents our experimental study. Finally, Section 5.5 concludes the chapter.

5.2 Definitions and Analysis

In this section, we introduce our terminology and notation. Then, we formalize our concept, called α-*dominant* service skyline, based on a dominance relationship defined in a fuzzy way. To motivate and justify our formulation, we also discuss some related notions, namely Pareto-dominance and service skyline, showing that our concept is more adequate for Web service selection.

Given a set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of functionally similar Web services and a set $Q = \{q_1, \ldots, q_d\}$ of QoS attributes. Each service S_i is characterized by a vector $Q(S_i) = (q_1(S_i), \cdots, q_d(S_i))$ where $q_\imath(S_i)$ denotes the value of the $\imath$-th QoS attribute of S_i. We consider quantitative QoS attributes (e.g., execution time, price, reputation, etc). To allow for an uniform measurement of service qualities independent of units, we normalize the different QoS values in the range $[0, 1]$, such that the lower the value, the higher the quality, as follows:

- For negative QoS parameters, i.e., the higher the value, the lower the quality (e.g., response time, latency, etc): $N_{q_k(s_i)} = \frac{q_k(s_i) - min_{q_k}}{max_{q_k} - min_{q_k}}$;

- For positive QoS parameters, i.e., the higher the value, the higher the quality (e.g., availability, reliability, etc.): $N_{q_k(s_i)} = \frac{max_{q_k} - q_k(s_i)}{max_{q_k} - min_{q_k}}$.

Where $N_{q_k(s_i)}$ is the normalized QoS value of the service s_i on the QoS parameter q_k and min_{q_k} (resp. max_{q_k}) is the minimum (resp. maximum) value of the QoS parameter q_k.

5.2.1 Fuzzy Dominance vs Pareto Dominance

We start by defining the Pareto dominance, then discuss the reasons that motivate to make it fuzzy.

Definition 5.1 *(Pareto Dominance)*
Given two services $S_i, S_j \in \mathcal{S}$, we say that S_i dominates S_j, denoted by $S_i \prec S_j$, if and only if S_i is better than or equal to S_j in all attributes in Q and better in at least one attribute in Q, i.e., $\forall \imath \in [1,d] : q_\imath(S_i) \leq q_\imath(S_j) \wedge \exists \jmath \in [1,d] : q_\jmath(S_i) < q_\jmath(S_j)$.

Pareto dominance does not allow for discriminating between Web services with bad compromise and those with good compromise. To illustrate this issue, let $Q(S_3) = (q_1(S_3), q_2(S_3)) = (0.1, 0.9)$ and $Q(S_{12}) = (q_1(S_{12}), q_2(S_{12})) = (0.2, 0.2)$ be the QoS vectors of S_3 and S_{12}, respectively. i.e., q_1 and q_2 represent respectively the execution time and the price (see Figure 5.1). With Pareto order, we have neither $S_3 \prec S_{12}$ nor $S_{12} \prec S_3$, i.e., the services S_3 and S_{12} are incomparable. However, one can consider that S_{12} is better than S_3 since $q_2(S_{12}) = 0.2$ is too much preferred than $q_2(S_3) = 0.9$, contrariwise $q_1(S_3) = 0.1$ is almost close to $q_1(S_{12}) = 0.2$. For this purpose, it is interesting to fuzzify the Pareto dominance in order to express the extent to which a Web service (more or less) dominates another one.

We define below a fuzzy dominance relationship that relies on particular comparison function expressing a graded inequality of the type *strongly smaller* than.

Definition 5.2 *(Fuzzy Dominance)*
Given two services $S_i, S_j \in \mathcal{S}$, we define the fuzzy dominance to express the degree to which S_i dominates S_j as:

$$deg_{\mu_{\varepsilon,\lambda}}(S_i \prec S_j) = \frac{\sum_{\imath=1}^{d} \mu_{\varepsilon,\lambda}(q_\imath(S_i), q_\imath(S_j))}{d} \tag{5.1}$$

Where $\mu_{\varepsilon,\lambda}$ is a monotone comparison function that expresses the extent to which $q_i(S_i)$ is more or less (strongly) smaller than $q_i(S_j)$. The function $\mu_{\varepsilon,\lambda}$ can be defined in an absolute way as follows:

$$\mu_{\varepsilon,\lambda}(x,y) = \begin{cases} 0 & if\ y-x \leq \varepsilon \\ 1 & if\ y-x \geq \lambda+\varepsilon \\ \frac{y-x-\varepsilon}{\lambda} & otherwise \end{cases} \tag{5.2}$$

Where $\lambda > 0$, i.e., $\mu_{\varepsilon,\lambda}$ is more demanding than the idea of *strictly smaller*. We should also have $\varepsilon \geq 0$ in order to ensure that $\mu_{\varepsilon,\lambda}$ agrees with the idea of *smaller* in the usual sense. Figure 5.2, shows the graphical representation of the function $\mu_{\varepsilon,\lambda}$ in terms of the difference $y - x$.

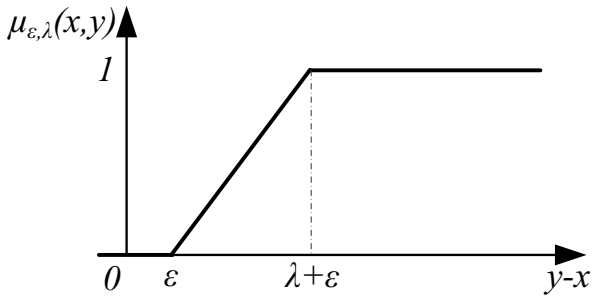


Figure 5.2: Graphical Representation of $\mu_{\varepsilon,\lambda}$ w.r.t. $y - x$

One can interpret the semantics of the function $\mu_{\varepsilon,\lambda}$ as follows:

- if $y - x$ is less than ε, then x is not at all strongly smaller than y;
- if $y - x$ is larger than $\lambda + \varepsilon$, then x is all much smaller than y;
- if $y - x$ is between ε and $\lambda + \varepsilon$, than x is much smaller than y to some extent.

Let us now reconsider the previous Web services S_3 and S_{12}, with $\varepsilon = 0.1$ and $\lambda = 0.2$, we have $deg_{\mu_{0.1,0.2}}(S_3 \prec S_{12}) = 0$ and $deg_{\mu_{0.1,0.2}}(S_{12} \prec S_3) = 0.5$. This is more significant than S_3 and S_{12} are incomparable – provided by Pareto dominance. As can be seen, the fuzzy dominance relationship introduced favors Web services with a good compromise.

5.2.2 α-Dominant Service Skyline vs Service Skyline

We first formally define the service skyline and the α-*dominant* service skyline, we then investigate the difference between them showing that the latter is more robust.

Definition 5.3 *(Service Skyline)*
The service skyline of $\mathcal{S}$, denoted by $sky^{\mathcal{S}}$ comprises the set of services in $\mathcal{S}$ that are not dominated by any other service, i.e., $sky^{\mathcal{S}} = \{S_i \in \mathcal{S} \mid \nexists S_j \in \mathcal{S} : S_j \prec S_i\}$.

In contrast to the service skyline which relies on Pareto dominance, the α-*dominant* service skyline leverages a notion called α-*dominance*. Below, we define both of the α-*dominance* and the α-*dominant* service skyline.

Definition 5.4 *(α-Dominance)*
Given two services $S_i, S_j \in \mathcal{S}$ and $\alpha \in [0,1]$, we say that S_i α-dominates S_j (or S_i dominates S_j at a degree α) in the context of $\mu_{\varepsilon,\lambda}$, denoted by $S_i \prec^{\alpha}_{\mu_{\varepsilon,\lambda}} S_j$, if and only if $deg_{\mu_{\varepsilon,\lambda}}(S_i \prec S_j) \geq \alpha$.

For instances, in the context of $\mu_{0.1,0.2}$ service S_{12} 0.7-*dominates* services S_5 and S_6. In the same context, S_{12} 0.8-*dominates* S_5, but does not 0.8-*dominates* S_6 as $deg_{\mu_{0.1,0.2}}(S_{12} \prec S_6) = 0.75 < 0.8$.

Definition 5.5 *(α-Dominant Service Skyline)*
The α-dominant service skyline of $\mathcal{S}$ with respect to $\mu_{\varepsilon,\lambda}$, denoted by α-$sky^{\mathcal{S}}_{\mu_{\varepsilon,\lambda}}$, comprises the set of services in $\mathcal{S}$ that are not α-dominated by any other service in the context of $\mu_{\varepsilon,\lambda}$, i.e., α-$sky^{\mathcal{S}}_{\mu_{\varepsilon,\lambda}} = \{S_i \in \mathcal{S} \mid \nexists S_j \in \mathcal{S} : S_j \prec^{\alpha}_{\mu_{\varepsilon,\lambda}} S_i\}$.

For example, with $\varepsilon = 0.1, \lambda = 0.2$ and $\alpha = 0.7$ we have $0.7\text{-}sky^{\mathcal{S}}_{\mu_{0.1,0.2}} = \{S_3, S_4, S_{11}, S_{12}, S_{13}, S_{14}\}$.

One can observe that, in contrast to the service skyline, the α-*dominant* service skyline privileges Web services with a good compromise. Now if the Web service S_{12} fails, users can choose between a good deal of Web services with a good compromise (e.g., S_{11} and S_{13}).

Further, the following theorem provides another key property of the α-*dominant* service skyline: a*ll Web services selected by the service skyline can be also selected by the α-dominant service skyline.*

Theorem 5.1

If $\alpha > \frac{d-1}{d}$*, then the service skyline is a subset of the* α*-dominant service skyline for any comparison function* $\mu_{\varepsilon,\lambda}$*, i.e.,* $\alpha > \frac{d-1}{d} \Rightarrow sky^{\mathcal{S}} \subseteq \alpha\text{-}sky^{\mathcal{S}}_{\mu_{\varepsilon,\lambda}}(\forall \varepsilon \geq 0, \forall \lambda > 0)$.

Proof

Assume that $\alpha > \frac{d-1}{d}$, and prove that for any comparison function $\mu_{\varepsilon,\lambda}$ $sky^{\mathcal{S}} \subseteq \alpha\text{-}sky^{\mathcal{S}}_{\mu_{\varepsilon,\lambda}}$. Let $S_i \in sky^{\mathcal{S}}$. According to Definition 5.3, $\nexists S_j \in \mathcal{S} : S_j \prec S_i$, i.e., $\forall S_j \in \mathcal{S}, \exists k \in [1,d] : q_k(S_i) < q_k(S_j)$. Therefore, for any comparison function $\mu_{\varepsilon,\lambda}$ we will have: $\forall S_j \in \mathcal{S}, \exists k \in [1,d] : \mu_{\varepsilon,\lambda}(q_k(S_j), q_k(S_i)) = 0$. Thus, $\forall S_j \in \mathcal{S} : deg_{\mu_{\varepsilon,\lambda}}(S_j \prec S_i) \leq \frac{d-1}{d}$ since S_i is better at least on the dimension k. Then $\forall S_j \in \mathcal{S}, deg_{\mu_{\varepsilon,\lambda}}(S_j \prec S_i) < \alpha$, since $\alpha > \frac{d-1}{d}$. This means that S_i is not α-*dominated* by any other service S_j in $\mathcal{S}$, i.e., $\nexists S_j \in \mathcal{S} : S_j \prec^{\alpha}_{\mu_{\varepsilon,\lambda}} S_i$. Thus, $S_i \in \alpha\text{-}sky^{\mathcal{S}}_{\mu_{\varepsilon,\lambda}}$. Hence, $sky^{\mathcal{S}} \subseteq \alpha\text{-}sky^{\mathcal{S}}_{\mu_{\varepsilon,\lambda}}(\forall \varepsilon \geq 0, \forall \lambda > 0)$. □

Theorem 5.1 shows that the α-*dominant* service skyline is appropriate for all types of users. In other words, if a user prefers a Web service with a bad compromise (e.g., he/she prefers a fast service although it is very expensive), the α-*dominant* service skyline can include such kind of services.

In addition to the above observations, the α-*dominant* service skyline allows users to control the size of the returned services by making changes on the parameter α, and possibly on ε and λ (whereas the service skyline's size does not change for the same set of services and the same query). For example, if users find that the size of $0.7\text{-}sky^{\mathcal{S}}_{\mu_{0.1,0.2}}$ is quite large (resp. small), they can reduce (resp. expand) it by decreasing (resp. increasing) the value of α (e.g., $\alpha = 0.2$ or $\alpha = 0.8$). They will thus have $0.2\text{-}sky^{\mathcal{S}}_{\mu_{0.1,0.2}} = \{S_{11}, S_{12}\}$ or $0.8\text{-}sky^{\mathcal{S}}_{\mu_{0.1,0.2}} = \{S_3, S_4, S_6, S_9, S_{10}, S_{11}, S_{12}, S_{13}, S_{14}\}$. Roughly speaking, the α-*dominant* service skyline allows for taking the feedback of users into account. We show formally this behavior below:

Lemma 5.1

If $\alpha' < \alpha$*, then the* α'*-dominant service skyline with respect to* $\mu_{\varepsilon,\lambda}$ *is a subset of the* α*-dominant service skyline with respect to* $\mu_{\varepsilon,\lambda}$*, i.e.,* $\alpha' < \alpha \Rightarrow \alpha'\text{-}sky^{\mathcal{S}}_{\mu_{\varepsilon,\lambda}} \subseteq \alpha\text{-}sky^{\mathcal{S}}_{\mu_{\varepsilon,\lambda}}$.

Proof Let $\mu_{\varepsilon,\lambda}$ be a comparison function. Assume that $\alpha' < \alpha$ and prove that $\alpha'\text{-}sky^{\mathcal{S}}_{\mu_{\varepsilon,\lambda}} \subseteq \alpha\text{-}sky^{\mathcal{S}}_{\mu_{\varepsilon,\lambda}}$. Let $S_i \in \alpha'\text{-}sky^{\mathcal{S}}_{\mu_{\varepsilon,\lambda}}$. This means that S_i is not α'-*dominated* by any other service S_j in $\mathcal{S}$, i.e., $\nexists S_j \in \mathcal{S} : S_j \prec^{\alpha'}_{\mu_{\varepsilon,\lambda}} S_i$. Thus, there is not a service S_j in $\mathcal{S}$ such as $deg_{\mu_{\varepsilon,\lambda}}(S_j \prec S_i) \geq \alpha'$, i.e., $\forall S_j \in \mathcal{S} : deg_{\mu_{\varepsilon,\lambda}}(S_j \prec S_i) < \alpha'$. Then, $\forall S_j \in \mathcal{S} : deg_{\mu_{\varepsilon,\lambda}}(S_j \prec S_i) < \alpha$, since $\alpha' < \alpha$. Therefore, $\nexists S_j \in \mathcal{S} : S_j \prec^{\alpha}_{\mu_{\varepsilon,\lambda}} S_i$. This means that S_i is not α-*dominated* , thus $S_i \in \alpha\text{-}sky^{\mathcal{S}}_{\mu_{\varepsilon,\lambda}}$. Hence, $\alpha'\text{-}sky^{\mathcal{S}}_{\mu_{\varepsilon,\lambda}} \subseteq \alpha\text{-}sky^{\mathcal{S}}_{\mu_{\varepsilon,\lambda}}$. □

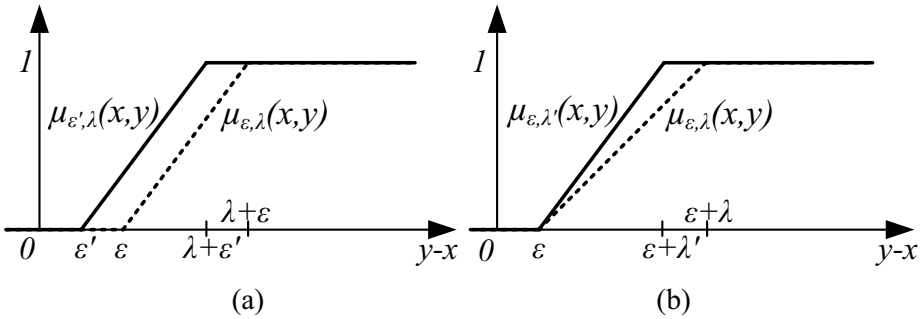


Figure 5.3: Effects of ε and λ

Property 5.1

If $\varepsilon' \leq \varepsilon$ and $\lambda' \leq \lambda$, then for any $(x,y) \in [0,1]$, $\mu_{\varepsilon',\lambda'}(x,y) \geq \mu_{\varepsilon,\lambda}(x,y)$, i.e., $(\varepsilon' \leq \varepsilon \wedge \lambda' \leq \lambda) \Rightarrow \forall (x,y) \in [0,1] : \mu_{\varepsilon',\lambda'}(x,y) \geq \mu_{\varepsilon,\lambda}(x,y)$.

Proof

$[\varepsilon' \leq \varepsilon \wedge \lambda' = \lambda]$: see Figure 5.3 (plot-a). $\ldots(\star)$

$[\varepsilon' = \varepsilon \wedge \lambda' \leq \lambda]$: see Figure 5.3 (plot-b). $\ldots(*)$

$[\varepsilon' \leq \varepsilon \wedge \lambda' \leq \lambda]$: it is straightforward from $(\star)$ and $(*)$. □

Lemma 5.2

If $\mu_{\varepsilon',\lambda'} \geq \mu_{\varepsilon,\lambda}$, then for any $\alpha \in [0,1]$ the α-dominant service skyline with respect to $\mu_{\varepsilon',\lambda'}$ is a subset of the α-dominant service skyline with respect to $\mu_{\varepsilon,\lambda}$, i.e., $\mu_{\varepsilon',\lambda'} \geq \mu_{\varepsilon,\lambda} \Rightarrow \alpha\text{-}sky^{\mathcal{S}}_{\mu_{\varepsilon',\lambda'}} \subseteq \alpha\text{-}sky^{\mathcal{S}}_{\mu_{\varepsilon,\lambda}}$.

Proof

Let α be a dominance degree. Assume that $\mu_{\varepsilon',\lambda'} \geq \mu_{\varepsilon,\lambda}$ and prove that $\alpha\text{-}sky^{\mathcal{S}}_{\mu_{\varepsilon',\lambda'}} \subseteq \alpha\text{-}sky^{\mathcal{S}}_{\mu_{\varepsilon,\lambda}}$. Let $S_i \in \alpha\text{-}sky^{\mathcal{S}}_{\mu_{\varepsilon_1,\lambda}}$. This means that S_i is not α-*dominated* by any

other service S_j in $\mathcal{S}$ in the context of $\mu_{\varepsilon',\lambda'}$, i.e., $\nexists S_j \in \mathcal{S} : S_j \prec^{\alpha}_{\mu_{\varepsilon',\lambda'}} S_i$. Thus, there is not a service S_j in $\mathcal{S}$ such as $deg_{\mu_{\varepsilon',\lambda'}}(S_j \prec S_i) \geq \alpha'$, i.e., $\forall S_j \in \mathcal{S} : deg_{\mu_{\varepsilon',\lambda'}}(S_j \prec S_i) < \alpha'$. Then, $\forall S_j \in \mathcal{S} : \frac{\sum_{i=1}^{d} \mu_{\varepsilon',\lambda'}(q_i(S_j),q_i(S_i))}{d} < \alpha$. Thus, $\forall S_j \in \mathcal{S} : \frac{\sum_{i=1}^{d} \mu_{\varepsilon,\lambda}(q_i(S_j),q_i(S_i))}{d} < \alpha$ (since $\mu_{\varepsilon',\lambda'} \geq \mu_{\varepsilon,\lambda}$). Then, $\forall S_j \in \mathcal{S} : deg_{\mu_{\varepsilon,\lambda}}(S_j \prec S_i) < \alpha$. Thus, $\nexists S_j \in \mathcal{S} : S_j \prec^{\alpha}_{\mu_{\varepsilon,\lambda}} S_i$. This means that S_i is not α-*dominated* in the context of $\mu_{\varepsilon,\lambda}$, therefore $S_i \in \alpha\text{-}sky^{\mathcal{S}}_{\mu_{\varepsilon,\lambda}}$. Hence, $\alpha\text{-}sky^{\mathcal{S}}_{\mu_{\varepsilon',\lambda'}} \subseteq \alpha\text{-}sky^{\mathcal{S}}_{\mu_{\varepsilon,\lambda}}$. □

Lemma 5.1 and Lemma 5.2 provide appropriate tools in order to adapt (by contracting or expanding) the size of the retrieved services to users needs.

We now provide the formal definition for the service selection problem using fuzzy dominance relationship.

Problem statement: Given a set of functional similar services $\mathcal{S} = \{S_1, \ldots, S_n\}$, a set of QoS attributes $Q = \{q_1, \ldots, q_d\}$, a comparison function $\mu_{\varepsilon,\lambda}$ and a dominance degree α. Return the α-dominant service skyline.

5.3 Computing the α-Dominant Service Skyline

Index structures are frequently used to reduce search space in large databases. To this end, in our study we make use of R-trees structures [Gut84] due to their popularity and effectiveness in skyline computation. For the sake of illustration, let us use the Web services given in Figure 5.1. These services can be organized in the R-tree of Figure 5.4, with node capacity = 3. An intermediate entry e_i corresponds to the minimum bounding rectangle (MBR) of a node N_i at the lower level, while a leaf entry corresponds to a Web service. Distances are computed according to L1 norm, i.e., the *mindist* of a point equals the sum of its coordinates and the *mindist* of a MBR (i.e., intermediate entry) equals the *mindist* of its lower-left corner point.

5.3.1 Efficient Computation of the α-*Dominant* Service Skyline

Intuitively, a straightforward approach to compute the α-*dominant* service skyline is to compare each Web service S_i with every other one. If S_i is not α-*dominated*, then it belongs to the α-*dominant* service skyline. However, this approach results in a high computational cost, as it needs to compare each Web service with every others. It is thus crucial to quickly eliminate Web services that are α-*dominated*.

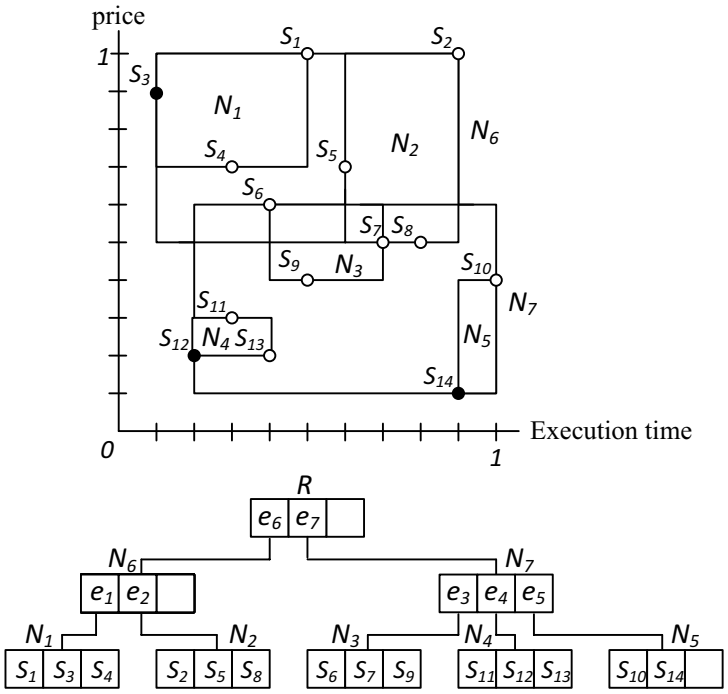


Figure 5.4: An Example of R-tree

It is worth to note that contrary to Pareto dominance the α-*dominance* relationship is not asymmetric, i.e., it is possible to have two Web services S_i and S_i such that S_i α-*dominates* S_j and S_j α-*dominates* S_i, for instance in the context of $\mu_{0.1,0.2}$, S_{13} 0.5-*dominates* S_3 and also S_3 0.5-*dominates* S_{13}. Therefore, the pruning process is far from being straightforward since it can lead to erroneous results. For example, if S_3 is pruned as it is α-*dominated* by S_{13} and there is no other service that α-*dominates* S_{13} (the case of our running example, for $\varepsilon = 0.1$, $\lambda = 0.2$ and $\alpha = 0.5$), then S_{13} will be included in the α-*dominant* service skyline, whereas it is α-*dominated* by S_3. This justifies why the current R-tree-based skyline algorithms are not suitable for computing the α-*dominant* service skyline.

In the following, we provide optimization techniques to address the above mentioned issues. The idea is to prune services that are both α-*dominated* and not needed for pruning other services and to minimize the number of comparisons. The optimization techniques follow an important concept called α-*Pareto-dominance*:

Definition 5.6 *(α-Pareto-Dominance)*
Given two services $S_i, S_j \in \mathcal{S}$, we say that S_i α-Pareto-dominates S_j in the context

of $\mu_{\varepsilon,\lambda}$*, denoted by* $S_i \lhd^{\alpha}_{\mu_{\varepsilon,\lambda}} S_j$*, if and only if* $S_i \prec S_j \wedge S_i \prec^{\alpha}_{\mu_{\varepsilon,\lambda}} S_j$*.*

Lemma 5.3

Given two services $S_i, S_j \in \mathcal{S}$*, if* S_i *dominates* S_j*, then for any* $S_k \in \mathcal{S}$ *and for any comparison function* $\mu_{\varepsilon,\lambda}$*:* $deg_{\mu_{\varepsilon,\lambda}}(S_i \prec S_k) \geq deg_{\mu_{\varepsilon,\lambda}}(S_j \prec S_k)$*, i.e.,* $S_i \prec S_j \Rightarrow \forall S_k \in \mathcal{S} : deg_{\mu_{\varepsilon,\lambda}}(S_i \prec S_k) \geq deg_{\mu_{\varepsilon,\lambda}}(S_j \prec S_k)$ $(\forall \varepsilon \geq 0, \forall \lambda > 0)$*.*

Proof

$S_i \prec S_j \Leftrightarrow \forall \imath \in [1,d] : q_\imath(S_i) \leq q_\imath(S_j) \wedge \exists \jmath \in [1,d] : q_\jmath(S_i) < q_\jmath(S_j)$. Taking only the implication $\Rightarrow$, we will have, $S_i \prec S_j \Rightarrow \forall \imath \in [1,d] : q_\imath(S_i) \leq q_\imath(S_j)$. Thus for any service S_k, $\forall \imath \in [1,d] : q_\imath(S_i) - q_\imath(S_k) \leq q_\imath(S_j) - q_\imath(S_k)$. Then, $\forall \imath \in [1,d] : q_\imath(S_k) - q_\imath(S_i) \geq q_\imath(S_k) - q_\imath(S_j)$. Therefore, for any comparison function $\mu_{\varepsilon,\lambda}$, we will have, $\forall \imath \in [1,d] : \mu_{\varepsilon,\lambda}(q_\imath(S_i), q_\imath(S_k)) \geq \mu_{\varepsilon,\lambda}(q_\imath(S_j), q_\imath(S_k))$. It follows that $\frac{\sum_{\imath=1}^{d} \mu_{\varepsilon,\lambda}(q_\imath(S_i), q_\imath(S_k))}{d} \geq \frac{\sum_{\imath=1}^{d} \mu_{\varepsilon,\lambda}(q_\imath(S_j), q_\imath(S_k))}{d}$. Hence, $deg_{\mu_{\varepsilon,\lambda}}(S_i \prec S_k) \geq deg_{\mu_{\varepsilon,\lambda}}(S_j \prec S_k)$. □

Lemma 5.4

For $\alpha > \frac{d-1}{d}$*, if a Web service* S_i *is not* α*-Pareto-dominated by any Web service in* $\mathcal{S}$ *in the context of a given comparison function* $\mu_{\varepsilon,\lambda}$*, then* $S_i \in sky^{\mathcal{S}}_{\mu_{\varepsilon,\lambda}}$

Proof

Assume that S_i is not α-Pareto-dominated. This means that $\forall S_j \in \mathcal{S} : S_j \not\prec S_i \vee S_j \not\prec^{\alpha}_{\mu_{\varepsilon,\lambda}} S_i$. $[S_j \not\prec^{\alpha}_{\mu_{\varepsilon,\lambda}} S_i]$: the proof is obvious as S_i is not α-dominated. $[S_j \not\prec S_i]$: by adopting the same formality of the proof of theorem 1, we will have $\nexists S_j \in \mathcal{S} : S_j \prec^{\alpha}_{\mu_{\varepsilon,\lambda}} S_i$. Hence, $S_i \in sky^{\mathcal{S}}_{\mu_{\varepsilon,\lambda}}$. □

Lemma 5.3 shows that the skyline services are sufficient to decide if a service is part (or not) of the α-*dominant* service skyline. This essentially reduces the number of comparisons. Also, the combination of Definition 5.6 and Lemma 5.3 specifies a key property that can be used to prune services: *if a service* S_j *is* α*-Pareto-dominated then prune it as (i) it is not part of the* α*-dominant service skyline (it is* α*-dominated); and (ii) it is unnecessary for comparisons (it is dominated).* In addition, Lemma 5.4 helps to avoid any comparison after pruning all α-*dominated* services, in the case where $\alpha > \frac{d-1}{d}$.

5.3.2 α-*Dominant* Service Skyline Algorithm

The algorithm, hereafter referred to as α-DSSA (see Algorithm 5.1), leverages the techniques presented above to compute the α-*dominant* service skyline, avoiding an exhaustive comparison of each service with all other ones. More specifically, it proceeds in two steps. The goal of the first step is to prune the α-*Pareto-dominated* Web services. The remaining Web services will go into the second step, where the α-*dominant* ones will be selected.

Step 1 (lines 1-17): *Finding candidate services* – It stars from the root node of the R-tree R and inserts all its entries into the heap H, sorted in ascending order according to their *mindist*. The top entry e (i.e., the entry with the minimum *mindist*) is extracted. If e is α-*Pareto-dominated* by any service in *Sky* then discard it, since all the services obtained from it (i.e., e) are α-*dominated* and not useful to prune other entries. Otherwise (i.e., e is not α-*Pareto-dominated*), if e is an intermediate entry, insert all its child entries that are not α-*Pareto-dominated* by any service in *Sky* into the heap. Else (i.e., e is a service), there are two cases: (i) If e is not dominated by any service in *Sky* (i.e., e is a skyline service), it is inserted into *Sky* as it may be part of the α-*dominant* service skyline and it is necessary for pruning other entries; (ii) If e is dominated by any service in *Sky*, it is inserted into *Dom* as it may be part of the α-*dominant* service skyline but it is not necessary for pruning other entries. This step proceeds in the same manner until the heap becomes empty;

Step 2 (lines 18-26): *Computing and returning the α-dominant service skyline* – If $\alpha > \frac{d-1}{d}$, then the α-*dominant* service skyline comprises all services of *Sky* and *Dom*, according to Lemma 5. Otherwise (i.e., $\alpha \leq \frac{d-1}{d}$), the algorithm proceeds by refining the lists *Sky* and *Dom*, keeping only the services that are not α-*dominated* by any services in *Sky*, as they are also not α-*dominated* by any services in *Dom*. Finally it provides the user with the α-*dominant* service skyline.

Note that according to Lemma 5.1 and Lemma 5.2, once we compute the α-*dominant* service skyline with respect to $\mu_{\varepsilon,\lambda}$, the α'-*dominant* service skyline with

Algorithm 5.1: α-DSSA

Input: *R-tree R; dominance degree* α*; comparison function* $\mu_{\varepsilon,\lambda}$*;*

Output: *the* α*-dominant service skyline DSky;*

```
1  begin
2      H ← ∅; Sky ← ∅; Dom ← ∅;
3      insert the root entries of R into H;
4      while H ≠ ∅ do
5          extract the top entry e from H;
6          if e is α-Pareto-dominated by some service in Sky then
7              discard e;
8          else
9              if e is an intermediate entry then
10                 foreach child e_i of e do
11                     if e_i is not α-Pareto-dominated by some service in Sky then
12                         insert e_i into H;
13             else
14                 if e_i is not dominated by some service in Sky then
15                     insert e_i into Sky;
16                 else
17                     insert e_i into Dom;
18     if α > (d−1)/d then
19         DSky ← Sky ∪ Dom;
20     else
21         foreach S_i in Sky do
22             if S_i is not α-dominated by some service in Sky then
23                 insert S_i into DSky;
24             if S_i α-dominate a service S_j in Dom then
25                 discard S_j;
26         DSky ← DSky ∪ Dom;
27     return DSky;
```

respect to $\mu_{\varepsilon,\lambda}$ ($\alpha' < \alpha'$) and the α-*dominant* service skyline with respect to $\mu_{\varepsilon',\lambda'}$ ($\mu_{\varepsilon',\lambda'} \geq \mu_{\varepsilon,\lambda}$) can be computed by only performing a simple search on the Web services returned by the α-*dominant* service skyline with respect to $\mu_{\varepsilon,\lambda}$ instead of

rerun the algorithm.

5.4 Experimental Evaluation

In this section, we present an extensive experimental study of our approach. More specifically, we conduct two sets of experiments. First we focus on the size of the α-*dominant* service skyline. We also compute the service skyline (refereed to as TSS) to compare how the size of the the α-*dominant* service skyline (refereed to as α-DSS) varies from that of the traditional service skyline. In the second set of experiments, we study the computational cost of the proposed algorithm. In order to prove the efficiency and the scalability of our algorithm (α-DSSA), we developed also a base line algorithm (referred to as BLA) for comparison purpose.

The algorithms (i.e., α-DSSA and BLA) were implemented in Java. Datasets are indexed with an R-tree. The experiments were conducted on a 2.00 GHz Intel dual core CPU and 2 GB of RAM, running Windows.

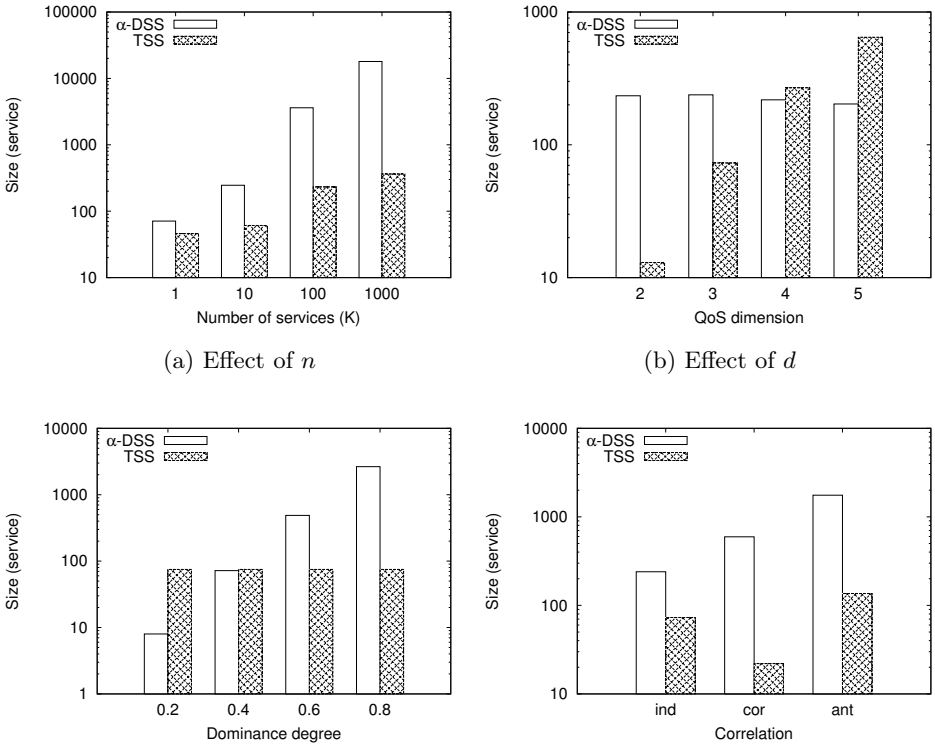
Table 5.1: Parameters and Examined Values

Parameter	Symbol	Values
Number of services	n	1K, **10K**, 100K, 1M
QoS dimensions	d	$[2, 5]$d, **3d**
Dominance degree	α	$[0.2, 0.8]$, **0.5**
Parameter correlation	*corr*	**ind**, cor, ant

5.4.1 Experimental Setup

In our experimental study we focus on synthetically generated datasets due to the limited availability of real data. The QoS values of services are generated in three different ways: independent (ind), where QoS values are assigned independently to each QoS attribute; correlated (cor), where the QoS values of a service are positively correlated, i.e., a good value in some QoS attribute increases the possibility of a good value in the others; anti-correlated (ant), where the QoS values are negatively correlated, i.e., good values (or bad values) in all QoS attributes are less likely to occur. The involved parameters and their examined values are summarized in

Table 5.1. In all experimental setups, we investigate the effects of one parameter, while we set the remaining ones to their default values, shown in bold in Table 5.1; we used $\varepsilon = 0.05$ and $\lambda = 0.2$.



(a) Effect of n

(b) Effect of d

(c) Effect of α

(d) Effect of *corr*

Figure 5.5: Effects of Parameters on the Size of the α-*dominant* Service Skyline an that of Traditional Service Skyline

5.4.2 Size of the α-*Dominant* Service Skyline

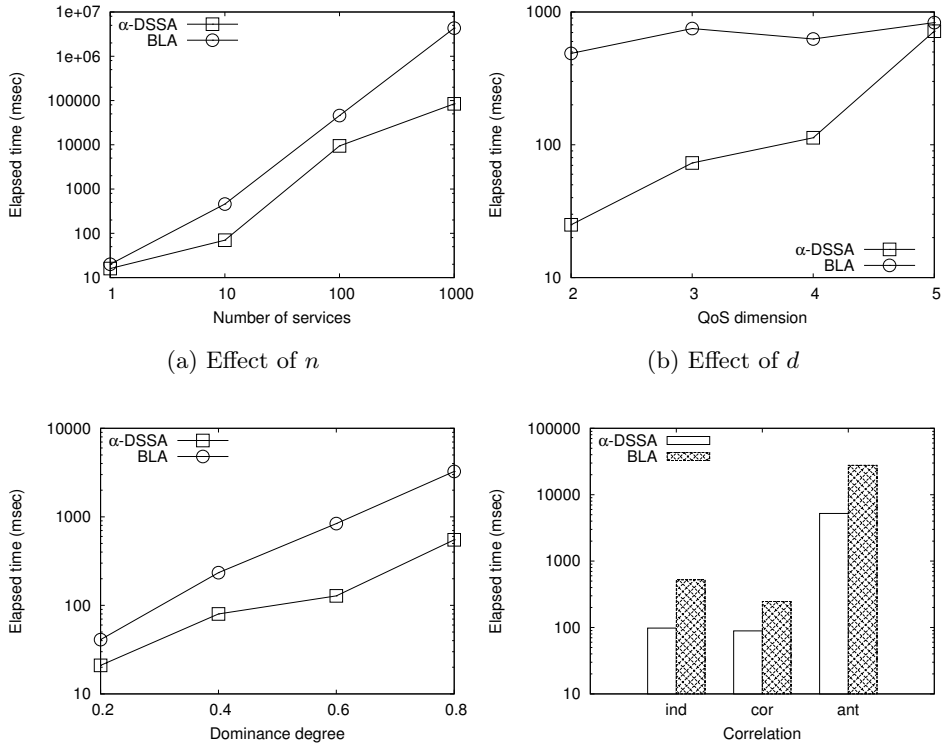
Figure 5.5a shows that the size of the α-*dominant* service skyline follows a similar trends as the traditional skyline with the increase of n. With α set to 0.5, the size of the α-*dominant* service skyline is larger than that of the traditional service skyline. In addition, the difference between the two sizes is proportional to n, since the number of services with a good compromise increases as n increases.

In contrast to the traditional service skyline whose size increases significantly as d increases, d has no obvious effect on the size of the α-*dominant* service skyline as shown in Figure 5.5b (the sizes can be regarded as in the same scale varying d).

Since the number of services with a good compromise is approximately the same when d varies.

The dominance degree α has a significant effect on the size of the α-*dominant* service skyline as shown in Figure 5.5c (the size of the traditional service skyline does not change as it is not related to α). This is because the increase (resp. decrease) of α leads to the inclusion (resp. exclusion) of services with a bad compromise.

Figure 5.5d shows that the α-*dominant* service skyline and the traditional service skyline exhibit different behaviors w.r.t. corr parameter. Furthermore, the size of the α-*dominant* service skyline is larger than that of the traditional service skyline especially for the correlated and anti-correlated datasets. On the correlated datasets many services with a good compromise occur. They are thus included in the α-*dominant* service skyline. On the anti-correlated datasets, services with a good compromise less likely to occur, thus there are not enough services which α-dominate others.



(a) Effect of n

(b) Effect of d

(c) Effect of *alpha*

(d) Effect of *corr*

Figure 5.6: Effects of Parameters on the time of α-DSSA and BLA

5.4.3 Performance and Scalability

Figure 5.6a shows that α-DSSA is consistently more efficient than BLA and BLA does not terminate successfully for $n > 100K$. The difference between α-DSSA and BLA increases significantly as n increases. This is due to the high number of pruned services by α-DSSA.

α-DSSA is consistently more efficient than BLA as shown in Figure 5.6b. The difference between α-DSSA and BLA decreases as d increases. This is because the size of the skyline increases significantly as d increases, thus the number of pruned services by α-DSSA decreases. In other words, the number of services selected to the second step of α-DSSA increases.

Figure 5.6c shows that α-DSSA is faster than BLA in a consistent manner and its performance advantage over BLA becomes more obvious with increasing α. Since the size of the α-*dominant* service skyline increases significantly as α increases. Then, the number of comparison involved in BLA is significant.

α-DSSA is more efficient than BLA on all distributions as shown in Figure 5.6d. In addition, α-DSSA is lower on anti-correlated datasets than correlated and independent datasets. This is because the size of the skyline is quite large on on anti-correlated datasets, thus the number of pruned services decreases. Then, the number of services selected to the second step of α-DSSA increases.

5.5 Conclusion

In this chapter, we have addressed the problem of QoS-based Web service selection. We have introduced a new concept, called α-*dominant* service skyline, to overcome the major issues of the current approaches: (*i*) requiring users to assign weights to QoS attributes, (*ii*) privileging the services with a bad compromise between different QoS attributes and (*iii*) not allowing users to control the size of the returned set of services. Further, we have developed a suitable algorithm for computing the α-*dominant* service skyline using pruning techniques. Our experimental evaluation demonstrates the effectiveness of the α-*dominant* service skyline and the efficiency of the proposed algorithm.

Chapter 6

Selecting Skyline Web Services from Uncertain QoS

Contents

6.1 Introduction

The development of enabling technologies for Web services is expected to change the way of conducting business on the Web. As Web services and service providers proliferate, more and more functionally similar Web services are deployed over the Web. Thus, there could be multiple Web services competing with each other to offer the same functionality, but with different QoS (quality of service) such as latency, price and response time. Moreover, QoS has been considered as a significant criterion for selecting among functionally similar Web services. Many QoS-base Web service selection approaches have been proposed. However, these approaches are not

sufficient in a dynamic Web service environment where the delivered QoS by a Web service is inherently uncertain.

6.1.1 Motivation and Challenges

Consider that a user wants to do an online payment on a given online shopping Web site. Typically, multiple Web services may be available providing this functionality (e.g., PayPal, WebMoney, etc.) but with different QoS values. Thus, finding the perfect Web service, which is the best in all QoS attributes, is ideal for the user. Unfortunately, such a Web service is seldom found. Moreover, computing the skyline from Web services based on QoS comes as a popular solution for selecting among functionally similar Web services [ASR10, YB10a]. A Web service s_i belongs to the service service skyline if there is not another Web service s_j such that s_j is better than s_i in all QoS attributes. In particular, the service skyline overcomes the major limitation of traditional approaches that require users to assign weights over different QoS attributes. However, current approaches that focus on computing the service skyline assume that the QoS does not change over time. Specifically, the QoS values are usually obtained from Web service descriptions. Whereas, these QoS values may not precisely reflect the actual performances of Web services because the performance of a Web service may vary due to the dynamic Web service environment. For instance, the response time may vary with the quality of the network. Therefore, the actual QoS delivered by a Web service is uncertain. Thus, computing the service skyline from uncertain QoS becomes important and challenging.

In summary, given a set of functionally similar Web services, the presence of uncertainty in their QoS raises the following challenges:

- Which is the more convenient way to model uncertain QoS?
- How can we capture the dominance relationship between Web services when their different QoS values are uncertain? And what should be the service skyline on those Web services?
- Can we provide optimization techniques to compute the service skyline from uncertain QoS efficiently?

6.1.2 Contributions

In this chapter, we tackle the above-mentioned challenges with the following main contributions:

- We leverage possibility theory, and model each QoS attribute of Web services as a possibility distribution;
- Given two Web services, we calculate the possibility and the necessity that each Web service dominates the other. Then, based on this dominance relationships, we propose the notion of *pos*-dominant service skyline and the notion of *nec*-dominant service skyline;
- We develop suitable algorithms for computing efficiently both the *pos*-dominant service skyline and the *nec*-dominant service skyline;
- We perform an extensive experimental evaluation verifying the effectiveness and the efficiency of the proposed service skyline extensions and algorithms.

The rest of this chapter is organized as follows. In Section 6.2, we formally define the key concepts, including the dominance relationship on uncertain QoS and the service skyline extensions, while in Section 6.3 we present our algorithms. An experimental evaluation is reported in Section 6.4. We conclude in Section 6.5.

6.2 Service Skyline on Uncertain QoS

In this section, we present a set of key concepts used throughout this chapter and two service skyline extensions on uncertain QoS. For reference, Table 6.1 contains the frequently used notation and its meaning.

Assume a set of functionally similar Web services $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ and a set of QoS attributes $\mathcal{Q} = \{q_1, q_2, \ldots, q_d\}$. Motivated by the example of Section 2.4.4 we model each Web service s_i as a set of possibility distributions $\{\pi_{s_i.q_1}, \pi_{s_i.q_2}, \ldots, \pi_{s_i.q_d}\}$ where each possibility distribution $\pi_{s_i.q_k}$ comprises all possible QoS values of $s_i.q_k$ and their possibility degrees. Note that the issue of measuring the QoS values and their possibility degrees is out of the scope of our current study. Therefore, we assume that such ill-known QoS values are provided by service providers; e.g., measured by experts based on historical, Web service environment, etc.

Table 6.1: The Summary of Notation

Notation	Definition
$\mathcal{S}$	a set of functionally similar Web services
$\mathcal{Q}$	a set of QoS attributes
s_i	a Web service
q_k	a QoS attribute
$s_i.q_k$	the k^{th} QoS value of s_i
$\pi_{s_i.q_k}$	the possibility distribution of $s_i.q_k$
$\Pi(s_i \prec s_j)$	the possibility that s_i dominates s_j
$\Pi(s_i.q_k \prec s_j.q_k)$	the possibility that $s_i.q_k$ dominates $s_j.q_k$
$s_i \prec^{\Pi}_{pos} s_j$	s_i *pos*-dominates s_j
$sky_{\Pi}(pos)$	*pos*-dominant service skyline
$N(s_i \prec s_j)$	the necessity that s_i dominates s_j
$N(s_i.q_k \prec s_j.q_k)$	the necessity that $s_i.q_k$ dominates $s_j.q_k$
$s_i \prec^{N}_{nec} s_j$	s_i *nec*-dominates s_j
$sky_N(nec)$	*nec*-dominant service skyline
$s_i.q_k^-$	minimum of completely possible values of $s_i.q_k$
$s_i.q_k^+$	maximum of completely possible values of $s_i.q_k$

The following example shows how to model QoS using possibility distributions.

6.2.1 Example

Consider two Web services s_1 and s_2 that offer online payment functionality. The provider of each Web service can estimate the QoS delivered to users on different QoS attributes and provide the QoS as a set of possibility distributions. Table 6.2 gives these possibility distributions with a focus on the price and the response time. For example, the price of Web service s_1 may occur with three possible values 1, 2 or 3. The possibility degrees of these values are 0.5, 0.7 and 1, respectively. Similarly, the price of Web service s_2 may occur with four possible values 1, 2, 3 or 4, and the possibility degrees of these values are 0.7, 1, 0.8 and 0.3, respectively. We use this example throughout the rest of the chapter.

Table 6.2: Example of Web Services with Uncertain QoS

Web service	QoS (price and response time)
s_1	$\pi_{s_1.price} = \{0.5/1, 0.7/2, 1/3\}$ $\pi_{s_1.responseTime} = \{0.3/18, 1/24, 1/26, 0.6/28\}$
s_2	$\pi_{s_2.price} = \{0.7/1, 1/2, 0.8/3, 0.3/4\}$ $\pi_{s_2.responseTime} = \{0.2/9, 0.7/20, 1/25, 0.6/30\}$

6.2.2 Service Skyline Extensions

Now let us extend the dominance relationship to the case of uncertain QoS. Let $s_i, s_j \in \mathcal{S}$, the possibility and the necessity that s_i dominates s_j are given by:

$$\Pi(s_i \prec s_j) = \min_{q_k \in \mathcal{Q}} \Pi(s_i.q_k \prec s_j.q_k) \tag{6.1}$$

$$N(s_i \prec s_j) = \min_{q_k \in \mathcal{Q}} N(s_i.q_k \prec s_j.q_k) \tag{6.2}$$

Where $\Pi(s_i.q_k \prec s_j.q_k)$ and $N(s_i.q_k \prec s_j.q_k)$ are the possibility degree and the necessity degree of the event "$s_i.q_k$ is better than (dominates) $s_j.q_k$", respectively and defined by:

$$\Pi(s_i.q_k \prec s_j.q_k) = \begin{cases} 0 & if\ \forall x \in \pi_{s_i.q_k}, \forall y \in \pi_{s_j.q_k} : x \geq y \\ \max_{x<y} \min(\pi_{s_i.q_k}(x), \pi_{s_j.q_k}(y)) & otherwise \end{cases} \tag{6.3}$$

$$N(s_i.q_k \prec s_j.q_k) = \begin{cases} 1 & if\ \forall\ x \in \pi_{s_i.q_k}, \forall\ y \in \pi_{s_j.q_k} : x < y \\ 1 - \max_{x \geq y} \min(\pi_{s_i.q_k}(x), \pi_{s_j.q_k}(y)) & otherwise \end{cases} \tag{6.4}$$

For example, the possibility and the necessity that service s_1 is better than service s_2 with respect to price are $\Pi(s_1.price \prec s_2.price) = 0.7$ and $N(s_1.price \prec s_2.price) = 0$, while those that s_1 is better than s_2 with respect to response time are $\Pi(s_1.responseTime \prec s_2.responseTime) = 1$ and $N(s_1.responseTime \prec s_2.responseTime) = 0.3$. Then, the overall possibility and necessity that s_1 dominates s_2 are $\Pi(s_1 \prec s_2) = \min(0.7, 1) = 0.7$ and $N(s_1 \prec s_2) = \min(0, 0.3) = 0$.

Moreover, a Web service s_i is said to *pos*-dominates (resp. *nec*-dominates) another Web service s_j if and only if $\Pi(s_i \prec s_j) \geq pos$ (resp. $N(s_i \prec s_j) \geq nec$). For example, if $pos = 0.6$ and $nec = 0.3$, we have s_1 *pos*-dominates s_2 as $\Pi(s_1 \prec s_2) = 0.7 \geq 0.6$, while, s_1 does not *nec*-dominates s_2 as $N(s_1 \prec s_2) = 0 < 0.3$.

We can now use these dominance relationships to define two service skyline extensions. More specifically, possibility-based service skyline and necessity-based service skyline. For a possibility (resp. necessity) threshold $pos \in [0, 1]$ (resp. $nec \in [0, 1]$), the *pos*-dominant service skyline (resp. *nec*-dominant service skyline) is the set of Web services that are not *pos*-dominated (resp. *nec*-dominated) by any other Web service. Formally:

$$sky_{\Pi}(pos) = \{s_i \in \mathcal{S} | \nexists s_j \in \mathcal{S} : s_j \prec^{\Pi}_{pos} s_i\} \tag{6.5}$$

$$sky_{N}(nec) = \{s_i \in \mathcal{S} | \nexists s_j \in \mathcal{S} : s_j \prec^{N}_{nec} s_i\} \tag{6.6}$$

Next, we illustrate some important properties of both the *pos*-dominant service skyline and the *nec*-dominant service skyline.

Theorem 6.1

If $pos = nec$ then the pos-dominant service skyline is a subset of the nec-dominant service skyline, i.e., $pos = nec \Rightarrow sky_{\Pi}(pos) \subseteq sky_{N}(nec)$.

Proof

Assume that there exists a Web service s_i, such that $s_i \in sky_{\Pi}(pos)$ and $s_i \notin sky_{N}(nec)$. Since $s_i \notin sky_{N}(nec)$, there must exist a Web service s_j, such that $s_j \prec^{N}_{nec} s_i$. Thus, we have $N(s_j \prec s_i) \geq nec = pos$. On the other hand, since $\Pi(s_j \prec s_i) \geq N(s_j \prec s_i)$ (see Section 2.4.2), we have $\Pi(s_j \prec s_i) \geq N(s_j \prec s_i) = nec = pos$. Thus, $s_j \prec^{\Pi}_{pos} s_i$ as $\Pi(s_j \prec s_i) \geq pos$. Which leads to a contradiction as $s_i \in sky_{\Pi}(pos)$. □

Lemma 6.1

If $pos < pos'$, then the pos-dominant service skyline is a subset of the pos'-dominant service skyline, i.e., $pos < pos' \Rightarrow sky_{\Pi}(pos) \subseteq sky_{\Pi}(pos')$.

Proof

Assume that there exists a Web service s_i, such that $s_i \in sky_{\Pi}(pos)$ and $s_i \notin$

$sky_{\Pi}(pos')$. Since $s_i \notin sky_{\Pi}(pos')$, there must exist a Web service s_j, such that $s_j \prec^{\Pi}_{pos'} s_i$. Thus, we have $\Pi(s_j \prec s_i) \geq pos'$. As $pos < pos'$, $\Pi(s_j \prec s_i) \geq pos$. Thus, $s_j \prec^{\Pi}_{pos} s_i$. Which leads to a contradiction as $s_i \in sky_{\Pi}(pos)$. □

Lemma 6.2

If $nec < nec'$, then the nec-dominant service skyline is a subset of the nec'-dominant service skyline, i.e., $nec < nec' \Rightarrow sky_N(nec) \subseteq sky_N(nec')$.

Proof

In a similar way as Lemma 6.1. □

Theorem 6.1 indicates that the size of the *pos*-dominant service skyline is smaller than or equal to the size of the *nec*-dominant service skyline for the same threshold. On the other hand, Lemma 6.1 shows that the size of the *pos*-dominant service skyline is smaller than or equals to the size of the pos'-dominant service skyline if $pos < pos'$, and the size of the *nec*-dominant service skyline is smaller than or equals to the size of the nec'-dominant service skyline if $nec < nec'$.

Roughly speaking, from Theorem 6.1, Lemma 6.1 and Lemma 6.2, we can see that the users have the flexibility to control the size of the returned services. Specifically, by varying the thresholds *pos* and *nec*.

6.3 Computing the Service Skyline Extensions

In this section, we first describe a general algorithm for computing both the *pos*-dominant service skyline and the *nec*-dominant service skyline. We then devise efficient algorithms for minimizing the number of dominance tests.

TSA (shown in Algorithm 6.1) is similar in spirit to the two scan algorithm [CJT+06a], computes the *pos*-dominant service skyline (*nec*-dominant service skyline) by scanning $\mathcal{S}$ twice. TSA proceeds as follows:

Step 1 (lines 1-10): *First scan* – In the first scan of $\mathcal{S}$, a set of candidate Web services, *Sky*, is computed by comparing each Web service s_i in $\mathcal{S}$ against the computed Web service in *Sky*. If a Web service s_j is *pos*-dominated (resp. *nec*-dominated) by s_i, then s_j is removed from *Sky*. At the end

Algorithm 6.1: TSA

Input: *a set of functionally similar Web service* $\mathcal{S}$*;*
possibility threshold pos; // necessity threshold *nes*

Output: *the pos-dominant skyline Sky;* // the *nec*-dominant skyline *Sky*

```
1  begin
2      Sky ← ∅; Boolean isSkyline;
3      foreach s_i ∈ S do
4          isSkyline ← true;
5          foreach s_j ∈ Sky do
6              if posDominates(s_j, s_i, pos) then        // necDominates(s_j, s_i, nec)
7                  isSkyline ← false;
8              if posDominates(s_i, s_j, pos) then        // necDominates(s_i, s_j, nec)
9                  remove s_j from Sky;
10         if isSkyline then
11             insert s_i into Sky;
12     foreach s_i ∈ S − Sky do
13         foreach s_j ∈ Sky do
14             if posDominates(s_i, s_j, pos) then        // necDominates(s_i, s_j, nec)
15                 remove s_j from Sky;
16     return Sky;
```

of the comparison against *Sky*, s_i is added into *Sky* if it is not *pos*-dominated (resp. *nec*-dominated) by any Web service in *Sky*. After the first scan of $\mathcal{S}$, *Sky* contains a set of candidate Web services;

Step 2 (lines 11-14): *Second scan* – To keep only the *pos*-dominant (resp. *nec*-dominant) Web services in *Sky*, a second scan of $\mathcal{S}$ is necessary. To determine whether a Web service s_j in *Sky* is indeed a *pos*-dominant (resp. *nec*-dominant) Web service, it is sufficient to compare s_j against each Web service s_i in $\mathcal{S} - Sky$.

Even if TSA can return the service skyline extensions, it results in a high computational cost, as the dominance tests (posDominate and necDominate in Algorithm 6.1) are very time-consuming. Specifically, to check if a Web service

s_i *pos*-dominates (resp. *nec*-dominates) another a Web service s_j, a straightforward method is to compare for each QoS attribute q_k in $\mathcal{Q}$, each possible value of $s_i.q_k$ with all possible values of $s_j.q_k$. Then, the minimum $\Pi(s_i.q_k \prec s_j.q_k)$ (resp. $N(s_i.q_k \prec s_j.q_k)$) is compared with the possibility (resp. necessity) threshold *pos* (resp. *nec*) to check if $\Pi(s_i.q_k \prec s_j.q_k) \geq pos$ (resp. $N(s_i.q_k \prec s_j.q_k) \geq nec$).

Minimizing the number of dominance tests, is thus important to improve the performance of TSA. In the following, we propose efficient functions that address this issue using the minimum and maximum completely possible values. The minimum and the maximum of completely possible values of $s_i.q_k$ are respectively the minimum and the maximum possible values of $s_i.q_k$ with possibility 1. They are denoted by $s_i.q_k^-$ and $s_i.q_k^+$, respectively. For example, $s_1.q_{price}^- = s_1.q_{price}^+ = 3$, $s_1.q_{responseTime}^- = 24$, and $s_1.q_{responseTime}^+ = 26$. Next, we delve into some useful lemmas that help us to improve the dominance tests.

Lemma 6.3

Assume two Web services s_i and s_j in $\mathcal{S}$. Then, given a possibility threshold pos, if there exists a QoS attribute $q_\ell \in \mathcal{Q}$ such that $\Pi(s_i.q_\ell \prec s_j.q_\ell) < pos$ then s_j is not pos-dominated by s_i.

Proof

Assume that there exists $q_\ell \in \mathcal{Q}$ such that $\Pi(s_i.q_\ell \prec s_j.q_\ell) < pos$ and $s_i \prec_{pos}^{\Pi} s_j$. From $s_i \prec_{pos}^{\Pi} s_j$ we have: $\Pi(s_i \prec s_j) \geq pos$, i.e., $\min_{q_k \in \mathcal{Q}} \Pi(s_i.q_k \prec s_j.q_k) \geq pos$. Which leads to a contradiction as $\Pi(s_i.q_\ell \prec s_j.q_\ell) < pos$. □

Lemma 6.4

Assume two Web services s_i and s_j in $\mathcal{S}$. Then, given a necessity threshold nec, if there exists a QoS attribute $q_\ell \in \mathcal{Q}$ such that $N(s_i.q_\ell \prec s_j.q_\ell) < nec$ then s_j is not nec-dominated by s_i.

Proof

In a similar way as Lemma 6.3. □

Lemma 6.5

Consider two Web services s_i and s_j in $\mathcal{S}$, and a QoS attribute $q_k \in \mathcal{Q}$. If $s_i.q_k^- < s_j.q_k^+$ then $\Pi(s_i.q_k \prec s_j.q_k) = 1$.

Proof

If $s_i.q_k^- < s_j.q_k^+$ then $\min(\pi_{s_i.q_k^-}, \pi_{s_j.q_k^+}) = \min(1,1) = 1$. Thus, $\Pi(s_i.q_k \prec s_j.q_k) = \max_{x<y} \min(\pi_{s_i.q_k}(x), \pi_{s_j.q_k}(y)) = 1$. □

Lemma 6.6

Consider two Web services s_i and s_j in $\mathcal{S}$, and a QoS attribute $q_k \in \mathcal{Q}$. If $s_i.q_k^+ \geq s_j.q_k^-$ then $N(s_i.q_k \prec s_j.q_k) = 0$.

Proof

If $s_i.q_k^+ \geq s_j.q_k^-$ then $\min(\pi_{s_i.q_k^+}, \pi_{s_j.q_k^-}) = \min(1,1) = 1$. Thus, $N(s_i.q_k \prec s_j.q_k) = 1 - \max_{x \geq y} \min(\pi_{s_i.q_k}(x), \pi_{s_j.q_k}(y)) = 1 - 1 = 0$. □

To determine if a Web service s_i *pos*-dominates (resp. *nec*-dominates) another Web service s_j, Lemma 6.3 (resp. Lemma 6.4) implies that it is not necessary to iterate all QoS attributes. On the other hand, Lemma 6.5 (resp. Lemma 6.6) allows to avoid comparisons between the possible values of $s_i.q_k$ and those of $s_j.q_k$ for any $q_k \in \mathcal{Q}$, when $s_i.q_k^- < s_j.q_k^+$ (resp. $s_i.q_k^+ \geq s_j.q_k^-$).

Based these Observations, we propose two efficient functions, *posDominates* (Algorithm 6.2) and *necDominates* (Algorithm 6.3), for optimization purposes.

Algorithm 6.2: $posDominates(s_i, s_j, pos)$

```
1 begin
2     float p ← 1;
3     foreach q_k ∈ Q do
4         if s_i.q_k^- ≥ s_j.q_k^+ then
5             p ← Π(s_i.q_k ≺ s_j.q_k);
6         if p < pos then
7             return false;
8     return true;
```

The details of *posDominates* are as follows. For each QoS attribute q_k in $\mathcal{Q}$, $s_i.q_k^-$ is first compared against $s_j.q_k^+$. If $s_i.q_k^- < s_j.q_k^+$, then the comparisons between the possible values of $s_i.q_k$ and those of $s_j.q_k$ are ignored as $\Pi(s_i.q_k \prec s_j.q_k) = 1$ (Lemma 6.5). Otherwise, i.e., $s_i.q_k^- \geq s_j.q_k^+$, each possible value of $s_i.q_k$ is compared against all possible values of $s_j.q_k$, to compute $\Pi(s_i.q_k \prec s_j.q_k)$. If $\Pi(s_i.q_k \prec$

$s_j.q_k) < pos$, then return $false$ as s_j is not *pos*-dominated by s_i (Lemma 6.3). If all QoS attribute have been iterated and $\Pi(s_i.q_k \prec s_j.q_k) \geq pos$ for any q_k in $\mathcal{Q}$, then return $true$ as s_i *pos*-dominates s_j.

Algorithm 6.3: $necDominates(s_i, s_j, nec)$

```
1  begin
2      float n;
3      foreach q_k ∈ Q do
4          if s_i.q_k^+ ≥ s_j.q_k^- then
5              return false;
6          else
7              n ← N(s_i.q_k ≺ s_j.q_k);
8          if n < nec then
9              return false;
10     return true;
```

necDominates proceeds as follows. For each QoS attribute q_k in $\mathcal{Q}$, $s_i.q_k^+$ is first compared against $s_j.q_k^-$. If $s_i.q_k^+ \geq s_j.q_k^-$, then return false because $N(s_i.q_k \prec s_j.q_k) = 0$ (Lemma 6.6); thus, s_j is not *nec*-dominated by s_i (Lemma 6.4). Otherwise, i.e., $s_i.q_k^+ < s_j.q_k^-$, each possible value of $s_i.q_k$ is compared against all possible values of $s_j.q_k$, to compute $N(s_i.q_k \prec s_j.q_k)$. If $N(s_i.q_k \prec s_j.q_k) < nec$, then return $false$ as s_j is not *nec*-dominated by s_i (Lemma 6.6). If all QoS attribute have been iterated and $N(s_i.q_k \prec s_j.q_k) \geq nec$ for any q_k in $\mathcal{Q}$, then return $true$ as s_i *nec*-dominates s_j.

6.4 Experimental Evaluation

In this section, we report our experimental study. More specifically, we conduct two sets of experiments. First, we focus on the size of our service skyline extensions, i,e., the *pos*-dominant service skyline and the *nec*-dominant service skyline. Second, we study the elapsed time for computing the skyline extensions. To show the benefits resulting from the use of *posDominates* and *necDominates* functions, we also developed baseline functions. Thus, we have four algorithms: pBTSA: TSA with a baseline *posDominates* function; pOTSA: TSA with our proposed *posDominates*

function; nBTSA: TSA with a baseline *necDominates* function; and nOTSA: TSA with our proposed *necDominates* function.

The algorithms were implemented in Java and all experiments were conducted on a core i5 with 8GB of RAM, running Mac OS X.

Table 6.3: Parameters and Examined Values

Parameter	Symbol	Values
Number of Web services	n	2K, 4K, **6K**, 8K, 10K
QoS dimensions	d	2, 4, **6**, 8, 10
possibility and necessity thresholds	t	0.1, 0.3, **0.5**, 0.7, 0.9
Parameter correlation	*corr*	cor, **ind**, ant

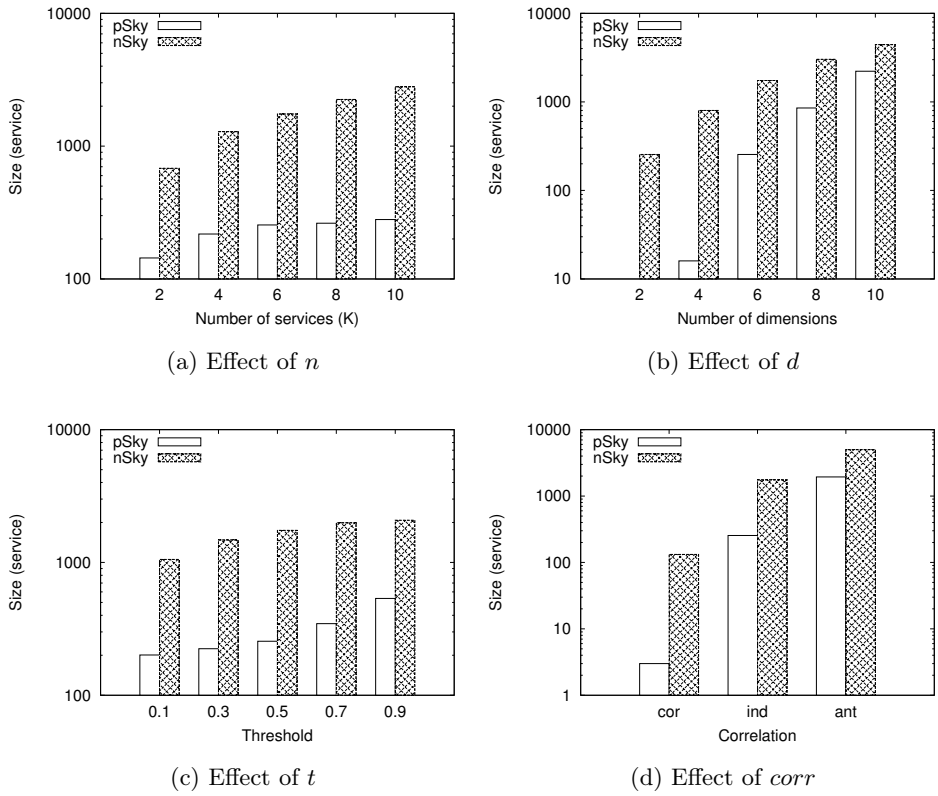
It is worth to note that due to the limited availability of real-world Web services with QoS measurements, in our experimental study, we focus on synthetic data. The uncertain QoS values of Web services are generated in three different ways: correlated (cor), where the QoS values of a Web service are positively correlated, i.e., a good value in some QoS attribute increases the possibility of a good value in the others; independent (ind), where QoS values of a Web service are assigned independently to each QoS attribute; and anti-correlated (ant), where the QoS values of a Web service are negatively correlated, i.e., good values (or bad values) in all QoS attributes are less likely to occur. Each QoS distribution of a Web service contains 10 possible values and at least one possible value is associated with a possibility 1, to ensure that all QoS distributions follow the normalization condition described in Section 2.4.1.

In each experimental setup, we vary a single parameter while setting the remaining to their default values. Table 6.3 displays the parameters under investigation and their corresponding ranges; default values are shown bold.

6.4.1 Size of the Service Skyline Extensions

Figure 6.1 shows the size (i.e., the number of Web services returned) of our service skyline extensions, i.e., the *pos*-dominant service skyline (pSky) and the *nec*-dominant service skyline (nSky) with respect to n, d, t and *corr*. Constantly, pSky

is less than nSky. This is consistent with Theorem 6.1.



(a) Effect of n

(b) Effect of d

(c) Effect of t

(d) Effect of $corr$

Figure 6.1: Effects of Parameters on the Size of the *pos*-dominant Service Skyline and the *nec*-dominant Service Skyline

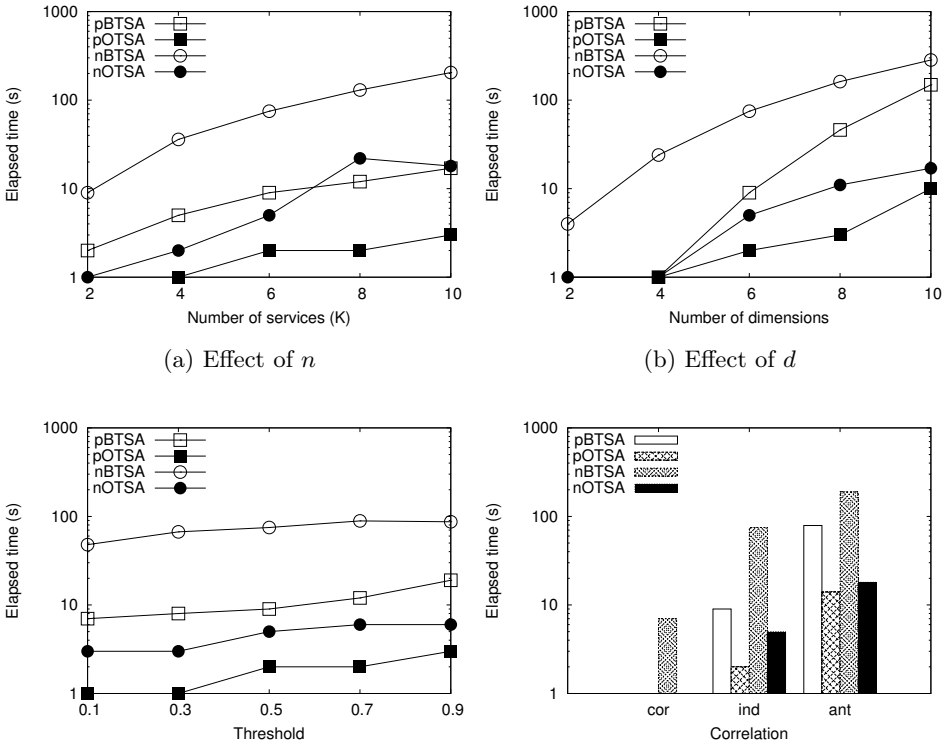
Figure 6.1a shows that both pSky and nSky increase with the increase of n. This is because more Web services have chances not to be dominated.

Figure 6.1b shows that both pSky and nSky increase significantly with higher d. As a Web service has better opportunity not to be dominated in all dimensions.

As shown in Figure 6.1c both pSky and nSky increase with the increase of t. This is because a *pos*-dominant service skyline (resp. *nec*-dominant service skyline) contains pos'-dominant service skyline (resp. nec'-dominant service skyline) if $pos > pos'$ (resp. $nec > nec'$), according to Lemma 6.1 and Lemma 6.2, respectively.

Figure 6.1d shows that both pSky and nSky are small for correlated data, while pSky and nSky are very large for anti-correlated data. For independent data pSky and nSky are somewhere in between. Since for correlated data, there is a few dominating Web services, i.e., they are good in all QoS attributes, for discarding

the other Web services, while for anti-correlated data, all Web service are very bad in at least one QoS attribute, so, a Web service has better opportunity not to be dominated. However, for independent data, where all QoS are uniformly distributed, pSky and nSky are in between. This is similar to the service skyline on certain QoS.



(a) Effect of n

(b) Effect of d

(c) Effect of t

(d) Effect of $corr$

Figure 6.2: Effects of Parameters on the Elapsed Time for Computing the *pos*-dominant Service Skyline and the *nec*-dominant Service Skyline

6.4.2 Elapsed time

Figure 6.2 investigates the runtime of the algorithms with respect to n, d, t and $corr$. Overall, both pOTSA and nOTSA are better than pBTSA and nBTSA. Specifically, pOTSA is faster than pBTSA and nOTSA is much faster than nBTSA. The results indicate that the proposed functions, i.e., *posDominates* and *necDominates* significantly save the cost of computing the *pos*-dominant service skyline and the *nec*-dominant service skyline, respectively. Let us now compare pOTSA against nOTSA. All experiments indicate that pOTSA in faster than nOTSA. This is because pSky

is less than nSky as shown in Figure 6.1. Therefore, the number of dominance tests in the first and the second scan (see Algorithm 6.1) is less for pOTSA.

As shown in Figure 6.2a n does not have a great effect on pOTSA as pSky increases slightly with the increase of n, while nSky increases significantly with the increase of n.

Figure 6.2b shows that both pOTSA and nOTSA follow similar trends with respect to d. This is because both pSky and nSky increase significantly with the increase of d.

Figure 6.2c shows that both pOTSA and nOTSA follow similar trends with respect to t. Also, Figure 6.2c shows that t does not have a great effect on both pOTSA and nOTSA. This is because both pSky and nSky increase slightly with the increase of t.

As shown in Figure 6.2d the elapsed time for computing the *pos*-dominant service skyline and the *nec*-dominant service skyline is more greater for anti-correlated data. However, it is reasonable for correlated and independent data. This is also related pSky and nSky (see Figure 6.1d).

6.5 Conclusion

In this chapter, we introduced two extensions of the service skyline on uncertain QoS to address the major limitation of the current approaches that assume that the delivered QoS of a Web service does not change over time, and devised appropriate skyline algorithms based on suitable dominance test functions. Our experimental results demonstrates both the effectiveness of the service skyline extensions and the efficiency of the proposed algorithms, and functions.

Chapter 7

Related Work

Contents

In this chapter, we give an overview of some work in the area of Web service selection and optimization which are most closely related to our work in Section 7.1. We then discus related work in the area of skyline computation in Section 7.2.

7.1 Web Service Selection and Optimization

During the last years, the problem of preference-based service selection has received a lot of attention. The main objective is to provide users with the most relevant services, i.e., that better satisfy their preferences, among those retrieved by service discovery. Agarwal and Lamparter proposed in [AL05] an approach for an automated selection of services for service composition. Service compositions can be compared with each other and ranked according to the user preferences, where preferences are modeled as a fuzzy IF-THEN rules. The IF part contains fuzzy descriptions of the various properties of a service, while 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 degree of acceptance are fuzzy sets. In [LASG07], the authors indicate that they model service configurations and preferences more compactly using utility function policies, which allows drawing from multi-attribute decision theory methods to develop an algorithm for optimal service selection. The authors also present the OWL ontology for the specification of configurable service offers and requests, and a flexible and extensible framework for optimal service selection that combines

declarative logic-based matching rules with optimization methods, such as linear programming. In [WXL08], the authors use a qualitative graphical representation of preferences, CP-nets, to deal with services selection in terms of user preferences. This approach can reason about user's incomplete and constrained preferences. In [WSZ+09], the authors propose a system for conducting qualitative 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. *ServiceTrust* [HYJY09] 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 [PCP09], 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. ServiceRank [WIS+09] 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.

Due to the limitation of these approaches to retrieve the most appropriate Web services, Skoutas et al. consider in [SSS+09, SSSS10] the dominance relationships between Web services based on their degrees of match to a given requested service in order to rank available services. Distinct scores based on the notion of dominance are defined for assessing when a service is objectively interesting. This work is the most related to our presented in Chapter 3. However, that work only considers selection of single services, without dealing with the problem of composition nor the user preferences.

Result diversification has recently attracted much attention as a means of increasing user satisfaction in recommender system and Web research [DP10]. In [SAN10], the authors propose a method to diversify Web service search results in

order to deal with users on the Web that have different, but unknown, preferences. The proposed method focuses on QoS parameters with non-numeric values, for which no ordering can be defined. However, this method provides the same services to all users without considering their personal preferences. In addition, the problem of composition is not addressed. In our diversification approach presented in Chapter 3 both the service composition with preferences and the result diversification are considered. In [McS02], Mc Sherry proposes an approach to retrieval that incrementally selects a diverse set of cases from a larger set of similarity-ordered cases. The same principle is adapted in our work for the diversification of the top-k Web service compositions but with different measurements.

Moreover, the problem of preference-based Web service selection under multiple users preferences is not addressed in the cited works, while in our study presented in Chapter 4 this problem is explicitly addresses.

On the other hand, the problem of QoS-based Web service selection has received considerable attention in the service computing community during the last years. In [LNZ04] the authors propose an extensible QoS computation model that supports an open and fair management of QoS data by incorporating user feedback. Zeng et al. [ZBD+03] [ZBN+04] propose a general and extensible model to evaluate QoS of both elementary and composite services. The authors use linear programming techniques to find the optimal selection of component services. Similar to this approach, Ardagna et al. [AP07] extend the linear programming model to include local constraints. In [YZL07] the authors studied the problem of service selection with multiple QoS constraints. The authors propose two models for the QoS-based service composition problem: (i) a combinatorial model and (ii) a graph model. A heuristic algorithm is introduced for each model. Wang et al. [WVKT06] introduces QoS-based selection of semantic web services, the authors present a QoS ontology and selection algorithm to evaluate multiple qualities. However, these approaches require the users to assign weights to QoS attributes. They thus suffer from lack of flexibility, in particular when the number of QoS attributes is high.

To overcome this limitation, skyline computation is adopted in Web service selection. The work in [ASR10] 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 [YB10b], the authors propose a skyline computation approach for service selection. The resulting skyline, called multi-service skyline, enables users to optimally and efficiently access sets of service as an integrated service package. The work presented in [YB10a] address the problem of uncertain QoS and compute the skylines from service providers. The authors define a concept called p-dominant skyline that contains the providers S that are not dominated with a probability p by any other provider. The authors provide also a discussion about the interest of p-dominant skyline with respect to the notion of p-skyline proposed in [PJLY07].

However, as shows in Section 5.2.2 these skyline-based approaches – based on the Pareto dominance relationship – gives the privilege to Web service with a bad compromise between QoS attributes, while the α-dominant service skyline presented in Chapter 5 privileges Web services with a good compromise between QoS attributes. It also gives users the flexibility to control the size of the returned services. In addition, the problem of uncertainty of QoS is not addressed in these works, excepted for [YB10a]. However, this approach is not suitable as it is based on probability theory, while Section 2.4.4 shows that the use of possibility theory is a better choice to tackle the problem of computing the service skyline from uncertain QoS. In our work presented in Chapter 6 we addressed this problem using on possibility theory.

7.2 Skyline Computation

To the best of our knowledge, skyline analysis, which came from old research topics like contour problem [McL74], maximum vectors [KLP75] and convex hull [PS85], was introduced into database domain by Börzsönyi et al. [BKS01]. The skyline is important for several applications involving multi-criteria decision making. Given a d-dimensional dataset, a point p is said to dominate another point q if and only if p is better than or equal to q in all dimensions and better than q in at least one. The skyline comprises the set of points in the dataset that are not dominated by any other point.

In [BKS01], Börzsönyi et al. develop three basic algorithms based on block nested loops (BNL), divide-and-conquer and index scanning (B-tree). Since, several algorithms have been developed to compute the skyline. Tan et al. [TEO01] introduce techniques, which can output the skyline without having to scan the en-

tire dataset. The work in [Cho03] observes that examining points according to a monotone (in all attributes) preference function reduces the average number of dominance checks. Based on this fact, the authors propose the Sort-first Skyline algorithm (SFS), which is similar to BNL but includes a presorting step. The SFS algorithm was further improved in [GSG05b] [GSG07] [BCP08]. Morse et al. propose in [MPJ07] an algorithm called LS using a static lattice structure for the special case of low-cardinality datasets. Recent works, propose partitioning-based algorithms without pre-computed indices [ZMC09] [LwH10].

Other works exploit appropriate indexes to speed-up the skyline computation process. In [KRR02], the authors present an improved algorithm, called NN due to its reliance on nearest neighbor search, which applies the divide-and-conquer framework on datasets indexed by R-trees. In the work [PTFS03], which also uses R-trees, the authors propose an optimal and progressive algorithm for skyline computation based on the Branch and Bound paradigm; our algorithm presented in Section 5.3 is also based on the Branch and Bound paradigm. In [LZLL07], Lee et al. propose ZSearch using ZB-tree as a new variant of B-tree.

Several extensions and related concepts to the skyline query have been studied. In [PTFS05], Papadias et al. propose the concept of k-dominating query, which retrieves the k points that dominate the largest number of other points; and the concept of k-skyband that contains the points dominated by less than k other points; the skyline is the 1-skyband. However, both k-dominating query and k-skyband do not always return skyline points. To resolve this, Lin et al. propose in [LYZZ07] the top-k representative skyline, so that the k skyline points with the maximal number of dominated points can be produced. However, this approach often return similar points [TDLP09]. For diversifying the result, Tao et al. propose in [TDLP09] the distance based representative skyline. The skycube query, proposed in [YLL+05], returns the points that are not dominated in a specified set of dimensions. In [CJT+06b], the authors propose the top-k skyline frequency. The skyline frequency of a point p is the number of subspaces where p is a skyline point. In [XZT08], Xia et al. introduce the ε-skyline that comprises the set of all points that are not ε-dominated by any other point. A point ε-dominates another points if and only if it is as good, better or slightly worse (up to ε) with regard to all dimensions and better in at least one dimension.

In [CJT+06a], Chan et al. propose the notion of k-dominance. A point p is said to k-dominate another point q if and only if there are k dimensions in which p dominates q. The k-dominant skyline consists of a subset of points that are not k-dominated. The authors develop three algorithm one scan algorithm (OSA); two scan algorithm TSA); and sorted retrieval algorithm (SRA) for computing the k-dominant skyline. This is the work the most related to ours presented in Chapter 4, where we adapted and improved OSA for computing efficiently the majority service skyline (Section 4.3.2); and that presented in Chapter 6, where we adapted and improved TSA by efficient functions for computing both the *pos*-dominant service skyline and the *nec*-dominant service skyline (Section 6.3).

In [BHP11], a variant of skyline queries over possibilistic relational databases is introduced. It relies on the possible interpretations of each tuple for computing the possibilistic degrees of dominance and no definition of uncertain skyline is proposed. Our proposal in Chapter 6 leverages the compact representation of the uncertain values of the services to define the skyline by using possibility and certainty degrees of dominance which are different from the degrees of the former approach, semantically and computationally speaking.

Skyline computation has been also studied in other environments. These include computing the skyline in a distributed environment, e.g., [BGZ04, WZF+06, WOTX07, VDKV07, CLX+08]; processing skyline queries over data streams, e.g., [LYWL05, TP06, SDKT08, ZLZ+09, ZLC09]; and skyline computation in mobile environment, e.g., [HJLO06, SCD09, QGLC10, LXH11, Lee11]. The works on skyline in service selection are presented in Section 7.1.

Chapter 8

Conclusions an Future Work

Contents

In this chapter, we first conclude this dissertation in Section 8.1. We then, describe several directions for the future work in Section 8.2.

8.1 Conclusions

It has been recognized that the Web services paradigm rapidly gains popularity constituting an integral part of many real-world applications. Due to the importance of Web service, many companies have invested very heavily in Web services technologies; e.g., Microsoft's .NET, IBM's Websphere, SUN's J2EE, to name just a few. These efforts have resulted in an increasing number of Web services deployed over the Web. Therefore, enhancing the capabilities of the current Web search engines with effective and efficient techniques for Web services retrieval and selection becomes an important issue.

In this dissertation, we provided optimization strategies to enable users to select the most appropriate Web services in a flexible way based on either their preferences or QoS. We summarize below our major contributions:

- *Top-k Web service compositions with fuzzy preferences* – We presented an approach for composing Web services while taking into account the user's fuzzy preferences. The (fuzzy) constraints of the relevant Web services are matched to those of the query to determine their degrees of match using a set of matching methods. We proposed a novel ranking criterion based on a fuzzification of Pareto dominance to select the most relevant services, then

compute the top-k Web service compositions. We propose also a method to improve the diversity of returned compositions while maintaining as possible the compositions with the highest scores. As the problem of Web service composition is known to be NP-hard, we developed for each method a suitable algorithm. We evaluated our approach through a set of thorough experiments.

- *Majority-rule-based Web service selection* – We dealt with the problem of preference-based Web service selection under multiple users preferences. We introduced a novel concept called majority service skyline based on the majority rule. This allows users to make a "democratic" decision on which Web services are the most appropriate. We developed a efficient algorithm for computing the majority service skyline. We conducted a set of thorough experiments to evaluate the effectiveness of the majority service skyline and the efficiency of our algorithm.

- *Computing skyline Web services using fuzzy dominance* – We proposed a skyline variant called α-dominant service skyline based on a fuzzification of Pareto dominance. The α-dominant service skyline allows the inclusion of Web services with a good compromise between QoS attributes, and the exclusion of Web services with a bad compromise between QoS attributes. It thus provides users with the most relevant Web services. It also gives user the flexibility to control the size of the returned Web services. We then developed an efficient algorithm based on R-Tree index structure for computing the α-dominant service skyline. We evaluated the effectiveness of the α-dominant service skyline and the efficiency of the algorithm through a set of experiments.

- *Selecting skyline Web services from uncertain QoS* – We modeled each uncertain QoS attribute using a possibility distribution, and introduced the notion of *pos*-dominant service skyline and the notion of *nec*-dominant service skyline that facilitates users to select their desired Web services with the presence of uncertainty in their QoS. We then develop appropriate algorithms to efficiently compute both the *pos*-dominant service skyline and *nec*-dominant service skyline. We evaluated our approach through a set of experiments.

8.2 Future Work

This dissertation leads to various fertile grounds for future research. We identify the following main directions for future work:

- In the current approaches, the selected Web services are returned to users at the end of the execution of the Web service query. An interesting future direction is to develop techniques so that the first selected Web services should be reported to users almost instantly and the result size should gradually increase. This essentially helps users to make a quick selection.

- The current approaches focus on all available Web service. However, users may take more interest in the more recent Web services that more precisely reflect the recent behavior of the corresponding service provider. Thus, an interesting future direction is to focus only on the most recent Web services.

- Context is an important concept to customize the service selection. For example, a user who looks for an online payment Web service prefers Web services with a high security level, and a user who looks for a search engine Web service may privileges Web services with a good response time. It is thus interesting to consider the context in the service selection.

- The current QoS-based service selection approaches assume that all QoS values of a Web service are provided. However, missing in QoS values may occur in real-world scenarios. Therefore, it is interesting to consider this case in the service selection.

- The current QoS-based service selection approaches relies on knowledge of QoS information from Web services. As Web service providers can not supply according to their betrothed QoS, an interesting extension is to develop a mechanism that precisely report the QoS values of Web services, then use this QoS values for the service selection.

Appendix A

Academic Achievements

---

1. **Karim Benouaret**, Dimitris Sacharidis, Djamal Benslimane, Allel Hadjali. Majority-Rule-Based Web Service Selection. In *the 13th International Conference on Web Information System Engineering (WISE 2012) – Short paper.*

2. **Karim Benouaret**, Djamal Benslimane, Allel Hadjali. WS-Sky: An Efficient and Flexible Framework for QoS-Aware Web Service Selection. In *the 9th International Conference on Services Computing (IEEE SCC 2012).*

3. **Karim Benouaret**, Djamal Benslimane, Allel Hadjali. Selecting Skyline Web Services from Uncertain QoS. In *the 9th International Conference on Services Computing (IEEE SCC 2012).*

4. **Karim Benouaret**, Djamal Benslimane, Allel Hadjali. Selecting Skyline Web Data Services for Multiple Users Preferences. In *the 19th International Conference on Web Services (IEEE ICWS 2012) – Short Paper.*

5. Soumaya Amdouni, Mahmoud Barhamgi, Djamal Benslimane, Allel Hadjali, **Karim Benouaret**, Rim Faiz. A User Centric-System for Answering Fuzzy Preference Queries Over Data Web Services. In *the 12th International Conference on Web Engineering (ICWE 2012) – Demo Paper.*

6. Idir Amine Amarouche, **Karim Benouaret**, Djamal Benslimane, Zaia Alimazighi, Michael Mrissa. Context-Driven and Service Oriented Semantic Mediation in DaaS Composition. In *the 4th International Conference on Networked Digital Technologies (NDT 2012).*

7. **Karim Benouaret**, Djamal Benslimane, Allel Hadjali. A Fuzzy Framework for Selecting Top-k Web Service Compositions. In *Applied Computing Review. Volume 11, Issue 3, August 2011 – Selected from ACM SAC 2011.*

8. **Karim Benouaret**, Ranking Web Service Compositions in the Context of User Preferences. In *the 37th International Conference on Very Large Databases (VLDB 2011) − PhD Workshop*.

9. **Karim Benouaret**, Djamal Benslimane, Allel Hadjali, Mahmoud Barhamgi. FuDoCS: A Web Service Composition System Based on Fuzzy Dominance for Preference Query Answering. In *the 37th International Conference on Very Large Databases (VLDB 2011) Demo Paper*.

10. **Karim Benouaret**, Djamal Benslimane, Allel Hadjali, Mahmoud Barhamgi. Top-k Web Service Compositions Using a Fuzzy Dominance Relationship. In *the 8th International Conference on Services Computing (IEEE SCC 2011)*.

11. **Karim Benouaret**, Djamal Benslimane, Allel Hadjali. On the Use of Fuzzy Dominance for Computing Service Skyline Based on QoS. In *the 9th International Conference on Web Services (IEEE ICWS 2011)*.

12. **Karim Benouaret**, Djamal Benslimane, Allel Hadjali. Top-k Service Compositions: A Fuzzy Set-Based Approach. In *the 26th Symposium On Applied Computing (ACM SAC 2011)*.

Bibliography

[AL05] Sudhir Agarwal and Steffen Lamparter. User preference based automated selection of web service compositions. In Kunal Verma; Amit Sheth; Michal Zaremba; Christoph Bussler, editor, *ICSOC Workshop on Dynamic Web Processes*, pages 1–12, Amsterdam, Netherlands, Dezember 2005. IBM. (Cited on page 101.)

[AMM08] Eyhab Al-Masri and Qusay H. Mahmoud. Investigating web services on the world wide web. In *WWW*, pages 795–804, 2008. (Cited on pages 2 and 68.)

[AP07] Danilo Ardagna and Barbara Pernici. Adaptive service composition in flexible processes. *IEEE Trans. Software Eng.*, 33(6), 2007. (Cited on page 103.)

[ASR10] Mohammad Alrifai, Dimitrios Skoutas, and Thomas Risse. Selecting skyline services for qos-based web service composition. In *WWW*, pages 11–20, 2010. (Cited on pages 31, 56, 69, 86 and 103.)

[BBM10] Mahmoud Barhamgi, Djamal Benslimane, and Brahim Medjahed. A query rewriting approach for web service composition. *IEEE T. Services Computing*, 3(3):206–222, 2010. (Cited on pages 26, 29, 31, 37, 44 and 45.)

[BBP96] Gloria Bordogna, Patrick Bosc, and Gabriella Pasi. Fuzzy inclusion in database and information retrieval query interpretation. In *SAC*, pages 547–551, 1996. (Cited on page 18.)

[BCP08] Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. Efficient sort-based skyline evaluation. *ACM Trans. Database Syst.*, 33(4), 2008. (Cited on page 105.)

[BGZ04] Wolf-Tilo Balke, Ulrich Güntzer, and Jason Xin Zheng. Efficient distributed skylining for web information systems. In *EDBT*, pages 256–273, 2004. (Cited on page 106.)

[BHP11] Patrick Bosc, Allel Hadjali, and Olivier Pivert. On possibilistic skyline queries. In *FQAS*, pages 412–423, 2011. (Cited on page 106.)

[BKS01] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001. (Cited on pages 31 and 104.)

[CDK+02] Francisco Curbera, Matthew J. Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. Unraveling the web services web: An introduction to soap, wsdl, and uddi. *IEEE Internet Computing*, 6(2):86–93, 2002. (Cited on pages 1 and 12.)

[Cho03] Jan Chomicki. Preference formulas in relational queries. *ACM Trans. Database Syst.*, 28(4):427–466, 2003. (Cited on pages 13, 31, 61 and 105.)

[CJT+06a] Chee Yong Chan, H. V. Jagadish, Kian-Lee Tan, Anthony K. H. Tung, and Zhenjie Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD Conference*, pages 503–514, 2006. (Cited on pages 56, 60, 63, 91 and 106.)

[CJT+06b] Chee Yong Chan, H. V. Jagadish, Kian-Lee Tan, Anthony K. H. Tung, and Zhenjie Zhang. On high dimensional skylines. In *EDBT*, pages 478–495, 2006. (Cited on page 105.)

[CLX+08] Bin Cui, Hua Lu, Quanqing Xu, Lijiang Chen, Yafei Dai, and Yongluan Zhou. Parallel distributed processing of constrained skyline queries by filtering. In *ICDE*, pages 546–555, 2008. (Cited on page 106.)

[DH73] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, New York, NY, 1973. (Cited on page 55.)

[DHM+04] Xin Dong, Alon Y. Halevy, Jayant Madhavan, Ema Nemes, and Jun Zhang. Simlarity search for web services. In *VLDB*, pages 372–383, 2004. (Cited on pages 31 and 54.)

[DP88] D. Dubois and H. Prade. *Possibility theory*. Plenum Press, New-York, 1988. (Cited on page 19.)

[DP96] Didier Dubois and Henri Prade. Using fuzzy sets in database systems: Why and how? In *FQAS*, pages 89–103, 1996. (Cited on page 18.)

[DP00] Didier Dubois and Henri Prade, editors. *Fundamentals of Fuzzy Sets* . The Handbooks of Fuzzy Sets Series. Kluwer, Boston, Mass., 2000. (Cited on pages 16, 17, 24 and 42.)

[DP10] Marina Drosou and Evaggelia Pitoura. Search result diversification. *SIGMOD Record*, 39(1):41–47, 2010. (Cited on pages 41 and 102.)

[DSV04] Alin Deutsch, Liying Sui, and Victor Vianu. Specification and verification of data-driven web services. In *PODS*, pages 71–82, 2004. (Cited on page 24.)

[EL04] Ahmed Elfatatry and Paul J. Layzell. Negotiating in service-oriented environments. *Commun. ACM*, 47(8):103–108, 2004. (Cited on page 10.)

[Emm00] Wolfgang Emmerich. *Engineering Distributed Objects*. John Wiley & Sons, Chichester, UK, April 2000. (Cited on page 10.)

[GSG05a] Parke Godfrey, Ryan Shipley, and Jarek Gryz. Maximal vector computation in large data sets. In *VLDB*, pages 229–240, 2005. (Cited on page 31.)

[GSG05b] Parke Godfrey, Ryan Shipley, and Jarek Gryz. Maximal vector computation in large data sets. In *VLDB*, pages 229–240, 2005. (Cited on page 105.)

[GSG07] Parke Godfrey, Ryan Shipley, and Jarek Gryz. Algorithms and analyses for maximal vector computation. *VLDB J.*, 16(1):5–28, 2007. (Cited on page 105.)

[Gut84] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD Conference*, pages 47–57, 1984. (Cited on page 76.)

[HJLO06] Zhiyong Huang, Christian S. Jensen, Hua Lu, and Beng Chin Ooi. Skyline queries against mobile lightweight devices in manets. In *ICDE*, page 66, 2006. (Cited on page 106.)

[HKP08] Allel Hadjali, Souhila Kaci, and Henri Prade. Database preferences queries - a possibilistic logic approach with symbolic priorities. In *FoIKS*, pages 291–310, 2008. (Cited on pages 18 and 24.)

[HKP11] Allel Hadjali, Souhila Kaci, and Henri Prade. Database preference queries - a possibilistic logic approach with symbolic priorities. *Ann. Math. Artif. Intell.*, 63(3-4):357–383, 2011. (Cited on page 13.)

[HYJY09] Qiang He, Jun Yan, Hai Jin, and Yun Yang. Servicetrust: Supporting reputation-oriented service selection. In *ICSOC/ServiceWave*, pages 269–284, 2009. (Cited on page 102.)

[KLP75] H. T. Kung, Fabrizio Luccio, and Franco P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22(4):469–476, 1975. (Cited on page 104.)

[KRR02] Donald Kossmann, Frank Ramsak, and Steffen Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB*, pages 275–286, 2002. (Cited on pages 31 and 105.)

[Lan03] Christoph Schlueter Langdon. The state of web services. *IEEE Computer*, 36(7):93–94, 2003. (Cited on page 1.)

[LASG07] Steffen Lamparter, Anupriya Ankolekar, Rudi Studer, and Stephan Grimm. Preference-based selection of highly configurable web services. In *WWW*, pages 1013–1022, 2007. (Cited on pages 56 and 101.)

[Lee11] Ken C. K. Lee. Efficient evaluation of location-dependent skyline queries using non-dominance scopes. In *COM.Geo*, page 14, 2011. (Cited on page 106.)

[LH03] Lei Li and Ian Horrocks. A software framework for matchmaking based on semantic web technology. In *WWW*, 2003. (Cited on page 54.)

[LNZ04] Yutu Liu, Anne H. H. Ngu, and Liangzhao Zeng. Qos computation and policing in dynamic web service selection. In *WWW (Alternate Track Papers & Posters)*, pages 66–73, 2004. (Cited on page 103.)

[LwH10] Jongwuk Lee and Seung won Hwang. Bskytree: scalable skyline computation using a balanced pivot selection. In *EDBT*, pages 195–206, 2010. (Cited on page 105.)

[LXH11] Xin Lin, Jianliang Xu, and Haibo Hu. Authentication of location-based skyline queries. In *CIKM*, pages 1583–1588, 2011. (Cited on page 106.)

[LYWL05] Xuemin Lin, Yidong Yuan, Wei Wang, and Hongjun Lu. Stabbing the sky: Efficient skyline computation over sliding windows. In *ICDE*, pages 502–513, 2005. (Cited on page 106.)

[LYZZ07] Xuemin Lin, Yidong Yuan, Qing Zhang, and Ying Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, pages 86–95, 2007. (Cited on page 105.)

[LZLL07] Ken C. K. Lee, Baihua Zheng, Huajing Li, and Wang-Chien Lee. Approaching the skyline in z order. In *VLDB*, pages 279–290, 2007. (Cited on page 105.)

[MBE03] Brahim Medjahed, Athman Bouguettaya, and Ahmed K. Elmagarmid. Composing web services on the semantic web. *VLDB J.*, 12(4):333–351, 2003. (Cited on page 2.)

[MBM+07] D. Martin, M. Burstein, D. Mcdermott, S. Mcilraith, M. Paolucci, K. Sycara, D.L. Mcguinness, E. Sirin, and N. Srinivasan. Bringing semantics to web services with owl-s. *World Wide Web*, 10(3):243–277, 2007. (Cited on page 29.)

[McL74] D. H. McLain. Drawing contours from arbitrary data points. *Comput. J.*, 17(4):318–324, 1974. (Cited on page 104.)

[McS02] David McSherry. Diversity-conscious retrieval. In *ECCBR*, pages 219–233, 2002. (Cited on pages 41, 42 and 103.)

[MPJ07] Michael D. Morse, Jignesh M. Patel, and H. V. Jagadish. Efficient skyline computation over low-cardinality domains. In *VLDB*, pages 267–278, 2007. (Cited on page 105.)

[PCP09] Matteo Palmonari, Marco Comerio, and Flavio De Paoli. Effective and flexible nfp-based ranking of web services. In *ICSOC/ServiceWave*, pages 546–560, 2009. (Cited on page 102.)

[PG03] Mike P. Papazoglou and Dimitrios Georgakopoulos. Service-oriented computing. *Commun. ACM*, 46(10), 2003. (Cited on page 10.)

[PJLY07] Jian Pei, Bin Jiang, Xuemin Lin, and Yidong Yuan. Probabilistic skylines on uncertain data. In *VLDB*, pages 15–26, 2007. (Cited on pages 34 and 104.)

[PKPS02] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. Semantic matching of web services capabilities. In *International Semantic Web Conference*, pages 333–347, 2002. (Cited on page 54.)

[PS85] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry - An Introduction*. Springer, 1985. (Cited on page 104.)

[PTFS03] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD Conference*, pages 467–478, 2003. (Cited on pages 31 and 105.)

[PTFS05] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005. (Cited on page 105.)

[PvdH07] Mike P. Papazoglou and Willem-Jan van den Heuvel. Service oriented architectures: approaches, technologies and research issues. *VLDB J.*, 16(3), 2007. (Cited on page 67.)

[QGLC10] Zhefeng Qiao, Junzhong Gu, Xin Lin, and Jing Chen. Privacy-preserving skyline queries in lbs. In *MVHI*, pages 499–504, 2010. (Cited on page 106.)

[SAN10] Dimitrios Skoutas, Mohammad Alrifai, and Wolfgang Nejdl. Re-ranking web service search results under diverse user preferences. In *VLDB, Workshop on Personalized Access, Profile Management, and Context Awareness in Databases*, pages 898–909, 2010. (Cited on page 102.)

[SCD09] Hailan Shen, Zhigang Chen, and Xiaoheng Deng. Location-based skyline queries in wireless sensor networks. In *Proceedings of the 2009 International Conference on Networks Security, Wireless Communications and Trusted Computing - Volume 01*, NSWCTC '09, pages 391–395, Washington, DC, USA, 2009. IEEE Computer Society. (Cited on page 106.)

[SDKT08] Nikos Sarkas, Gautam Das, Nick Koudas, and Anthony K. H. Tung. Categorical skylines for streaming data. In *SIGMOD Conference*, pages 239–250, 2008. (Cited on page 106.)

[Sin01] Munindar P. Singh. Being interactive: Physics of service composition. *IEEE Internet Computing*, 5(3):6–, 2001. (Cited on page 2.)

[SKP11] Kostas Stefanidis, Georgia Koutrika, and Evaggelia Pitoura. A survey on representation, composition and application of preferences in database systems. *ACM Trans. Database Syst.*, 36(3):19, 2011. (Cited on pages 13, 14 and 15.)

[SSS+09] Dimitrios Skoutas, Dimitris Sacharidis, Alkis Simitsis, Verena Kantere, and Timos K. Sellis. Top- dominant web services under multi-criteria matching. In *EDBT*, pages 898–909, 2009. (Cited on pages 32, 34, 36, 51 and 102.)

[SSSS10] Dimitrios Skoutas, Dimitris Sacharidis, Alkis Simitsis, and Timos K. Sellis. Ranking and clustering web services using multicriteria dominance relationships. *IEEE T. Services Computing*, 3(3):163–177, 2010. (Cited on pages 32, 34, 36, 51 and 102.)

[TDLP09] Yufei Tao, Ling Ding, Xuemin Lin, and Jian Pei. Distance-based representative skyline. In *ICDE*, pages 892–903, 2009. (Cited on page 105.)

[TEO01] Kian-Lee Tan, Pin-Kwang Eng, and Beng Chin Ooi. Efficient progressive skyline computation. In *VLDB*, pages 301–310, 2001. (Cited on pages 31 and 104.)

[TP06] Yufei Tao and Dimitris Papadias. Maintaining sliding window skylines on data streams. *IEEE Trans. Knowl. Data Eng.*, 18(2), 2006. (Cited on page 106.)

[VDKV07] Akrivi Vlachou, Christos Doulkeridis, Yannis Kotidis, and Michalis Vazirgiannis. Skypeer: Efficient subspace skyline computation over distributed data. In *ICDE*, pages 416–425, 2007. (Cited on page 106.)

[VN02] Steven J. Vaughan-Nichols. Web services: Beyond the hype. *IEEE Computer*, 35(2):18–21, 2002. (Cited on page 12.)

[WIS+09] Qinyi Wu, Arun Iyengar, Revathi Subramanian, Isabelle Rouvellou, Ignacio Silva-Lepe, and Thomas A. Mikalsen. Combining quality of service and social information for ranking services. In *ICSOC/ServiceWave*, pages 561–575, 2009. (Cited on page 102.)

[WOTX07] Shiyuan Wang, Beng Chin Ooi, Anthony K. H. Tung, and Lizhen Xu. Efficient skyline query processing on peer-to-peer networks. In *ICDE*, pages 1126–1135, 2007. (Cited on page 106.)

[WSZ+09] Hongbing Wang, Shizhi Shao, Xuan Zhou, Cheng Wan, and Athman Bouguettaya. Web service selection with incomplete or inconsistent user preferences. In *ICSOC/ServiceWave*, pages 83–98, 2009. (Cited on page 102.)

[WVKT06] Xia Wang, Tomas Vitvar, Mick Kerrigan, and Ioan Toma. A qos-aware selection model for semantic web services. In *ICSOC*, pages 390–401, 2006. (Cited on page 103.)

[WXL08] Hongbing Wang, Junjie Xu, and Peicheng Li. Incomplete preference-driven web service selection. In *IEEE SCC (1)*, pages 75–82, 2008. (Cited on pages 54 and 102.)

[WZF+06] Ping Wu, Caijie Zhang, Ying Feng, Ben Y. Zhao, Divyakant Agrawal, and Amr El Abbadi. Parallelizing skyline queries for scalable distribution. In *EDBT*, pages 112–130, 2006. (Cited on page 106.)

[XZT08] Tian Xia, Donghui Zhang, and Yufei Tao. On skylining with flexible dominance relation. In *ICDE*, pages 1397–1399, 2008. (Cited on page 105.)

[Yag80] Ronald R. Yager. An approach to inference in approximate reasoning. *International Journal of Man-Machine Studies*, 13(3):323–338, 1980. (Cited on page 17.)

[YB10a] Qi Yu and Athman Bouguettaya. Computing service skyline from uncertain qows. *IEEE T. Services Computing*, 3(1):16–29, 2010. (Cited on pages 31, 56, 63, 69, 86 and 104.)

[YB10b] Qi Yu and Athman Bouguettaya. Computing service skylines over sets of services. In *ICWS*, pages 481–488, 2010. (Cited on pages 31, 56, 63, 69 and 104.)

[YB12] Qi Yu and Athman Bouguettaya. Multi-attribute optimization in service selection. *World Wide Web*, 15(1):1–31, 2012. (Cited on pages 31, 56, 63 and 69.)

[YLBM08] Qi Yu, Xumin Liu, Athman Bouguettaya, and Brahim Medjahed. Deploying and managing web services: issues, solutions, and directions. *VLDB J.*, 17(3), 2008. (Cited on page 67.)

[YLL+05] Yidong Yuan, Xuemin Lin, Qing Liu, Wei Wang, Jeffrey Xu Yu, and Qing Zhang. Efficient computation of the skyline cube. In *VLDB*, pages 241–252, 2005. (Cited on page 105.)

[YZL07] Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *TWEB*, 1(1), 2007. (Cited on page 103.)

[Zad65] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965. (Cited on page 15.)

[Zad78] L.A. Zadeh. Fuzzy sets as a basis for theory of possibility. *Fuzzy Sets and Systems*, 1:3–28, 1978. (Cited on page 18.)

[ZBD+03] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng. Quality driven web services composition. In *WWW*, 2003. (Cited on page 103.)

[ZBN+04] Liangzhao Zeng, Boualem Benatallah, Anne H. H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. *IEEE Trans. Software Eng.*, 30(5), 2004. (Cited on page 103.)

[ZLC09] Ling Zhu, Cuiping Li, and Hong Chen. Efficient computation of reverse skyline on data stream. In *CSO (1)*, pages 735–739, 2009. (Cited on page 106.)

[ZLZ+09] Wenjie Zhang, Xuemin Lin, Ying Zhang, Wei Wang, and Jeffrey Xu Yu. Probabilistic skyline operator over sliding windows. In *ICDE*, pages 1060–1071, 2009. (Cited on page 106.)

[ZMC09] Shiming Zhang, Nikos Mamoulis, and David W. Cheung. Scalable skyline computation using object-based space partitioning. In *SIGMOD Conference*, pages 483–494, 2009. (Cited on page 105.)

[ZMKL05] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *WWW*, pages 22–32, 2005. (Cited on page 41.)