Towards optimal discovery of web services for multiple QoS constraints and preferences

Guobing Zou

School of Computer Engineering and Science,
Shanghai University,
Shanghai 200444, China
Email: gbzou@shu.edu.cn

Yanglan Gan*

School of Computer Science and Technology,
Donghua University,
Shanghai 201620, China
Email: ylgan@dhu.edu.cn
*Corresponding author

Sen Niu, Mei Zhao and Bofeng Zhang

School of Computer Engineering and Science,
Shanghai University,
Shanghai 200444, China
Email: nschina@shu.edu.cn
Email: zhaomei@shu.edu.cn
Email: bfzhang@shu.edu.cn

Abstract: Web service discovery (WSD) is the task of matchmaking a set of relevant web services. Quality of service (QoS) has recently been applied to represent non-functional properties of web services. Therefore, when those services provide the same functionality but have different QoS values, how to effectively filter out the services that cannot satisfy the QoS constraints and rank the remaining services is still an open research issue. In this paper, we propose an integrated approach that allows a service requester to specify a functionality request, multiple QoS constraints and their preferences, and our method discovers a set of the most appropriate ranked services with QoS utility aggregation. By conducting empirical experiments on simulated web services, we validate the feasibility of our service discovery approach. The running example shows that, our approach can find more appropriate services by the satisfiability of multiple QoS constraints and the ranking of aggregated QoS.

Keywords: web services; quality of service; QoS; service discovery; functionality matchmaking; QoS filtering.

Reference to this paper should be made as follows: Zou, G., Gan, Y., Niu, S., Zhao, M. and Zhang, B. (2014) 'Towards optimal discovery of web services for multiple QoS constraints and preferences', *Int. J. Web Engineering and Technology*, Vol. 9, No. 3, pp.277–299.

Biographical notes: Guobing Zou is an Assistant Professor in the School of Computer Engineering and Science at Shanghai University, China. He received his PhD in Computer Science from Tongji University, Shanghai, China, 2012. He has worked as a Visiting Scholar in the Department of Computer Science and Engineering at Washington University in St. Louis from 2009 to 2011, USA. His current research interests focus on web service composition, service discovery and uncertain planning. He has published around 30 papers on international journals and conferences, including *IEEE Transactions on Services Computing*, *Applied Intelligence*, *Knowledge-Based Systems*, AAAI and *Soft Computing*.

Yanglan Gan is an Assistant Professor in the School of Computer Science and Technology at Donghua University, Shanghai, China. She received her PhD in Computer Science from Tongji University in 2012. Her research interests include bioinformatics, data mining, web services and information retrieval. She has published more than 15 papers on international journals and conferences, including *Bioinformatics*, *BMC Bioinformatics*, *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, and *Knowledge-Based Systems*.

Sen Niu is currently a PhD student at Shanghai University from 2013. His research interests include service composition, service selection, and uncertain planning and optimisation.

Mei Zhao is currently a Master student at Shanghai University from 2013. Her research interests include web service selection and uncertain QoS evaluation.

Bofeng Zhang is a Full Professor in the School of Computer Engineering and Science at Shanghai University. He received his PhD degree from the Northwestern Polytechnic University (NPU) in 1997, China. He did a postdoctoral research at Zhejiang University from 1997 to 1999, China. He worked as a Visiting Professor at the University of Aizu from 2006 to 2007, Japan. His research interests include personalised service recommendation, intelligent human-computer interaction and data mining. He has published more than 120 papers on international journals and conferences.

1 Introduction

Web services are modular, self-descriptive, loosely coupled, and accessible distributed software components. After encapsulating the functionality of an application and providing accessible interfaces, they can be published over the internet in a web service repository, discovered by software agents and composed as new value-added and cross-organisational distributed applications. In most cases, web services can be advertised, discovered and invoked through XML-based standards and protocols, such as the standard web service description language (WSDL) that is used to describe the input and output interface of a Web service for its functionality at the syntactical level, the simple object access protocol (SOAP) applied to transfer and exchange messages among services, and the universal description, discovery and integration (UDDI) that is used to register and discover web services. Over the past few years, web services have been becoming more and more important in many real applications, as it offers an extremely versatile and powerful tool to dynamically create distributed applications on demand. Its

applications increase rapidly in many areas, such as electronic commerce, weather forecasting, credit check, enterprise application integration, and geographic information system.

Along with the rapid increment of services available on the internet, a large number of web services make the service-oriented architecture (SOA) much easier to be implemented and facilitate the dynamic creation of distributed applications. Web service discovery (WSD) aims at finding a set of the most relevant web services such that each of them can satisfy a specified service request. Given a large number of web services, however, it is still difficult to find satisfactory services that fulfil the desired functionality. Furthermore, quality of service (QoS) as an important metric consists of a group of non-functional properties, such as execution price, execution duration, availability, probability of success, and reputation (Zeng et al., 2004). These QoS criteria have been widely applied to represent non-functional features of a service. Therefore, when those services provide the same functionality, they may have different QoS values, the study on how to effectively filter out the services that cannot satisfy the QoS constraints posed by a user and then rank the remaining services using their QoS values is still an open research issue.

Although various service discovery problems have been intensively studied and different approaches have been proposed in the past few years (Dong et al., 2004; Nayak, 2008; Elgazzar et al., 2010; Yu, 2011; Wu et al., 2012; Patel and Chaudhary, 2009; Xiao et al., 2010; Hau et al., 2005; Atkinson et al., 2007; Plebani and Pernici, 2009; Meditskos and Bassiliades, 2010; Dietze et al., 2009b; Amorim et al., 2011; Paliwal et al., 2012; Al-Masri and Mahmoud, 2009, 2010; Lemos et al., 2012a), there are limitations to be resolved and we still face several research challenges.

1 Most traditional approaches for service discovery mainly take functionality request into account (Dong et al., 2004; Nayak, 2008; Elgazzar et al., 2010; Yu, 2011; Wu et al., 2012), instead of considering the satisfiability checking of multiple QoS constraints on desired services. As a result, although those services can satisfy functionality request, some of them may violate QoS constraints desired by service requesters. Moreover, without the computation of aggregated QoS of a service, we cannot differentiate those services providing the same functionality. Thus, designing an automatic service discovery approach that integrates QoS filtering and service ranking into functionality matchmaking is desirable.

2 In terms of efficiency of conventional service discovery approaches (Nayak, 2008; Yu, 2011; Wu et al., 2012; Xiao et al., 2010; Hau et al., 2005; Amorim et al., 2011; Paliwal et al., 2012; Al-Masri and Mahmoud, 2010), they acquire inputs and outputs of each service from a service repository when matching with a service request. However, the traditional methods for service matchmaking need to parse functionality interfaces of each service, which incurs expensive time cost when a user submits a service request. Due to the complexity of dynamically parsing web services, completing such service matchmaking can be a time-consuming task, such that it is inappropriate for service discovery in a large-scale web service repository.

3 Semantic Web has been used as a promising approach for automated WSD (Paolucci et al., 2002; Hau et al., 2005; Klusch et al., 2006; Atkinson et al., 2007; Plebani and Pernici, 2009; Dietze et al., 2009a, 2009b; Meditskos and Bassiliades, 2010; Amorim et al., 2011; Paliwal et al., 2012). Most of current approaches for semantic service

discovery need the support of semantic descriptions of web services through ontology languages, such as OWL-S and WSDL-S. However, these kinds of approaches encounter two challenges. First, the vast majority of already existing services over the internet have been published using WSDL instead of semantic tagged description. Thus, the translation from syntactic services to semantic annotated services would be a difficult task for semantic service discovery. Second, even though there exist Semantic Web services that can be available by manual annotation and deployment for WSD, we still cannot ensure the precision of semantic matchmaking between a syntactic functionality request and the inputs and outputs of a service.

To handle above limitations of existing WSD approaches, we propose an integrated approach that addresses two major issues related to automated service discovery, including service registration persistence, and QoS filtering and service ranking. Due to the lack of support from both the availability of Semantic Web services and domain ontology on the internet, we still perform syntactic matchmaking instead of semantic service matching. With this assumption, our proposed approach provides a service requester to specify a functionality request, multiple QoS constraints and their preferences on predefined QoS criteria, and our method discovers a set of the most appropriate ranked services, each of which can ensure the high QoS by QoS utility aggregation, while satisfying those specified functionality request.

In our proposed approach, the persistence registration of web services is performed by parsing functionality interfaces of a service (i.e., inputs and outputs) and putting them into a service database. By doing so, the matchmaking performance can be significantly improved during service discovery. The reason is that we only need to acquire the functionality interfaces and QoS information of web services from database once, instead of parsing the inputs and outputs of a service from a service repository whenever a user submits a service request. Moreover, for the QoS filtering and service ranking, we first discover a set of services using traditional service functionality matchmaking. Then, those functionally feasible services are checked against multiple QoS constraints to further narrow the appropriate services, such that we can filter out those services that cannot satisfy the QoS constraints. Finally, we apply QoS normalisation strategy to the refined set of Web services and calculate the aggregated QoS on each of them by a weighted sum of QoS values. As a consequence, depending on the comprehensive QoS, we employ an off-the-shelf highly efficient sorting algorithm to rank those services and respond to the service requester.

The proposed WSD approach has been developed and implemented as a prototype system in Java. Our approach significantly extends the capability of prior-work by integrating the checking of global satisfiability on QoS constraints and service ranking in terms of aggregated QoS. By conducting empirical experiments on simulated web services and their QoS information, we validate the feasibility of our approach. The empirical results on running example show that, we can find more appropriate services in terms of desired functional capabilities and multiple QoS constraints with preferences.

The rest of the paper is organised as follows. We review the related work on web service discovery in Section 2. We formulate the problem of WSD considering QoS in Section 3. We present our approach on QoS-based WSD (Q-WSD) in Section 4, including the overall framework of service discovery approach, service functionality matchmaking, QoS filtering and service selection, QoS normalisation, weight assignment

scheme, and service ranking, respectively. Section 5 presents an empirical evaluation. Finally, we conclude the paper and discuss our future work in Section 6.

2 Related work

We review existing works that are most relevant to the proposed approach. Based on the categorisation of WSD (Rambold et al., 2009), we divide WSD into syntactically clustering-based, context-aware, semantic-based, and QoS-aware approaches.

Clustering web services based on function similarity can greatly boost the accuracy and efficiency of web services search engines to retrieve the most relevant services. There are a number of approaches proposed in recent years for syntactic-based service functionality clustering and discovery (Dong et al., 2004; Nayak, 2008; Elgazzar et al., 2010; Yu, 2011; Wu et al., 2012; Vijayan and Balasundaram, 2013). Dong et al. (2004) proposed a service search engine called Woogle which computes the similarity between Web services by employing the structures of web services in WSDL. Taking service request into consideration, Nayak (2008) applied collaborative filtering clustering to recommend web services for a target user by search terms that similar users had used in similar queries. Instead of clustering user queries, Elgazzar et al. (2010) proposed a WSDL service clustering technique to bootstrap the discovery of web services. They mine WSDL documents and cluster them into functionally similar web service groups. Specifically, they selected five key features from WSDL descriptions as the features (i.e., content, types, messages, ports, and name of the web service) and then the quality threshold (QT) clustering algorithm was applied to cluster web services. Moreover, to handle the limitations of conventional clustering techniques applied in WSDL, Yu (2011) proposed a framework that applies non-negative matrix factorisation (NMF) to the WSDL corpus for service community discovery. The NMF-based community discovery is also augmented via semantic extensions of the WSDL descriptions outside the WSDL corpus. NMF demonstrated its effectiveness in clustering high-dimensional sparse data. For further improvement of the clustering performance of service discovery, Wu et al. (2012) proposed a novel clustering algorithm that groups web services by utilising both WSDL documents and tags. Particularly, they employed a hybrid web service tag recommendation strategy WSTRec to attack the problems of uneven tag distribution and noisy tags. In addition, other traditional clustering algorithms have been applied for service discovery, such as Vijayan and Balasundaram (2013) where they explored the resemblance among web services to generate clusters by K-means clustering algorithm using WSDL document features.

Context awareness has increasingly been considered for improving the QoS discovery results. It plays an important role in WSD. Context-aware service discovery mechanism requires a formal language to represent context operations and conditions. Patel and Chaudhary (2009) proposed a service discovery algorithm based on rule engine which acquires the contextual information and makes query of user information much richer for more precise service discovery. As context changes dynamically occur when developing real web applications, Cubo et al. (2010) proposed a model based on transition systems and extended with value passing, context information and conditions to control and evaluate the execution of the protocols. Especially, while capturing the user context information, the protocol compatibility is checked based on the deadlock-freeness. Different from existing approaches which depend on context models to know the

relations among context types and values, Xiao et al. (2010) employed multiple ontologies to automatically capture the relations among different context values. As a result, they mine potentially desired services.

The descriptive capacity of WSDL services has limited the effectiveness of current service discovery approaches. Some recent efforts have made to extend semantics of WSDL files to improve the effectiveness of service discovery (Paolucci et al., 2002; Hau et al., 2005; Klusch et al., 2006; Atkinson et al., 2007; Plebani and Pernici, 2009; Dietze et al., 2009a, 2009b; Meditskos and Bassiliades, 2010; Amorim et al., 2011; Paliwal et al., 2012). The semantic-based approaches adopt service description languages, such as OWL-S, DAML-S, and WSMO, to describe semantic functionality of web services and develop similarity-based matchmaking algorithms to retrieve desired services. At the beginning of semantic WSD exploration, Paolucci et al. (2002) proposed a semantic matching algorithm based on DAML-S and the matching degree between an advertised service and a service request can be divided into one of the four results, including exact, plug in, subsume, and fail. Based on this fundamental work, various semantic service discovery algorithms have been proposed. Hau et al. (2005) proposed a metric for measuring the similarity of semantic services annotated with OWL ontology, where similarity between an annotated service and a request is calculated by defining the intrinsic information value of a service description based on the 'inferencibility' of each OWL Lite constructs. Klusch et al. (2006) presented a hybrid OWL-S service matchmaker called OWLS-MX, which exploits means of both crisp logic-based and IR-based similarity matching. Based on different kinds of semantic similarity matchmaking (Ganesan et al., 2003), researchers paid more attention to the matching algorithms (Atkinson et al., 2007; Plebani and Pernici, 2009; Meditskos and Bassiliades, 2010) by similarity calculation between advertised services and a request for WSD.

QoS-aware dynamic discovery of web services has been recently becoming a hot research issue based on the service functionalities and non-functional properties. Al-Masri and Mahmoud (2009) associated QoS with web services that provides clients with ways to improve the discovery process, helps identify client goals when performing queries. Furthermore, based on the idea in Al-Masri and Mahmoud (2009, 2010) introduced a web service broker (WSB) that not only collects the web services disseminated throughout the web, but also enables clients to articulate service queries tailored to their needs. WSB is capable of ranking services according to QoS parameters. Lemos et al. (2012a) extended service matching algorithms based on the process model (PM) specification by making them sensitive to service requester preferences concerning service quality. As a result, PM can be augmented to represent service functionality and non-functional factors. Thus, the service discovery can been seen as a matching process between the user query PM and a target PM where quality preferences are taken into account at different stages. Borrowing the framework of Lemos et al. (2012a, 2012b) implemented a flexible tool called S-MatchMaker, capable of coupling different approaches for personalising service discovery based on structural and quality aspects.

During semantic service discovery, both the providers and requesters have to describe the services in terms of ontological concepts to avoid semantic heterogeneity. The requester may not be able to frame service request correctly because of strict semantic rules to specify service functionality. Moreover, Xu et al. (2011) demonstrated that the domain ontology for each area does not exist and the construction of domain ontology is still difficult. As explained above, we mainly focus on automatically discovering non-Semantic Web services using logic-based functionality matchmaking with QoS

filtering and ranking. Unlike the most existing QoS-based service discovery, we parse the functionalities of web services and their QoS values into a service database offline and also provide a lot of techniques for QoS filtering and service ranking, including satisfiability checking of multiple QoS constraints, the strategy of QoS normalisation, QoS weight assignment scheme, and aggregated QoS calculation of services. We do not need to read WSDL files when a user submits a service request, since their functionalities and QoS can be efficiently retrieved from service database. Thus, our proposed approach can provide the most appropriate web services from the view of functionality and high quality with aggregated QoS.

3 Q-WSD formulation

This section focuses on the understanding of Q-WSD problem which is fed into the service functionality matchmaking, QoS filtering and service ranking. They are comprised of the procedures of dynamic discovery of web services.

Before the definition of QoS-based service discovery problem, we give preliminary background by a set of formal definitions, and then clearly demonstrate what a service discovery problem and its solution are.

Definition 3.1 (web service): A service s is defined as a three-tuple (I, O, Q), where $I = \{I^1, I^2, \cdots\}$ is a set of input parameters, $O = \{O^1, O^2, \cdots\}$ is a set of output parameters, and $Q = \{Q^1, Q^2, \cdots\}$ is a set of QoS values that represent the non-functional features of s. We use $s.I$, $s.O$, and $s.Q$ to denote I, O, and Q of s, respectively.

Each service plays a role that can perform a specified task. A web service repository is a set of disjoint services. It is defined as follows.

Definition 3.2 (web service repository): A service repository $S = \{s_1, s_2, \cdots\}$ is a set of web services. Where, each $s \in S$ is a web service.

Given a service $s = (I, O, Q)$, its multiple QoS values Q correspond to a set of QoS criteria defined as below.

Definition 3.3 (QoS criteria set): A QoS property q represents a dimension of non-functional criteria in a web service s. QoS property set, $qos = \{q^1, q^2, \cdots, q^n\}$, are a set of QoS criteria. For $\forall q^i \in qos (1 \le i \le n)$, it corresponds to a QoS value $Q^i \in s.Q$, where s is a web service.

QoS criteria can be divided into two categories: positive and negative. Positive QoS criteria denote better quality with higher values, while negative ones correspond to lower quality with higher values.

Based on widely used QoS criteria (Zeng et al., 2004), we apply a QoS property set $qos = \{q^1, q^2, \cdots, q^n\}$ to model and specify the QoS values of each service s. More specifically, we employ $qos = \{$*execution time*, *execution price*, *availability*, *probability of success*, *reputation*$\}$ as references of QoS criteria for service providers to offer QoS values for a service. As mentioned above from the classification of QoS criteria, *execution time* and *execution price* belong to negative QoS properties, while *availability*, *probability of success*, and *reputation* are positive ones.

Definition 3.4 (service requester preferences): Given a set of QoS criteria $qos = \{q^1, q^2,\cdots,q^n\}$, service requester preferences, denoted as $W = \{w_1, w_2,\cdots,w_n\}$, are a set of corresponding QoS weights. For each q^i $(1 \le i \le n)$, a user (or an agent) can assign a QoS weight w_i $(1 \le i \le n)$ to represent the preference degree on the criterion, while all the weights must satisfy $\sum_{i=1}^{n} w_i = 1$ and $0 \le w_i \le 1$.

In addition to service requester preferences attached on a set of given QoS properties $qos = \{q^1, q^2,\cdots,q^n\}$, a user poses multiple QoS constraints on each criterion to enhance the satisfiability of non-functional features.

Definition 3.5 (QoS constraints): Given a set of QoS criteria $qos = \{q^1, q^2,\cdots,q^n\}$, $C = \{c^1, c^2,\cdots,c^n\}$ is denoted as a set of QoS constraints, where each $c^i \in C$ is a constraint on $q^i \in qos(1 \le i \le n)$.

Depending on the QoS features, for a positive QoS property $q^i \in qos(1 \le i \le n)$, we have $c^i = (q^i, \ge, v^i)$ as a QoS constraint, where v^i is the lower bound of q^i. Symmetrically, for a negative QoS property $q^j \in qos(1 \le j \le n)$, we have $c^j = (q^j, \le, v^j)$ as a QoS constraint, where v^j is the upper bound of q^j.

Definition 3.6 (service functionality request): A functionality request, denoted as $r = (I_r, O_r)$, where $I_r = \{I_r^1, I_r^2,\cdots\}$ is a set of input parameters provided by an end user as initial conditions, and $O_r = \{O_r^1, O_r^2,\cdots\}$ is a set of output parameters desirable to be returned to the user.

Given a set of available disjoint services distributed in a web service repository, a functionality request, a set of global QoS constraints and multiple service requester preferences associated with corresponding QoS criteria, we define a Q-WSD problem as below.

Definition 3.7 (Q-WSD): A QoS-based service discovery problem is defined as a five-tuple, denoted as Q-WSD = (S, C, W, I_r, O_r), where

1 $S = \{s_1, s_2,\cdots\}$ is a web service repository

2 $C = \{c_1, c_2,\cdots,c_n\}$ is a set of global QoS constraints on the specified QoS criteria

3 $W = \{w_1, w_2,\cdots,w_n\}$ corresponds to multiple service requester preferences on the QoS criteria

4 $I_r = \{I_r^1, I_r^2,\cdots\}$ is an initially functional condition with a set of input parameters

5 $O_r = \{O_r^1, O_r^2,\cdots\}$ is a goal specification with a set of desirable output parameters.

For a given Q-WSD = (S, C, W, I_r, O_r), one of the feasible solution to the problem is denoted as $s^f \in S$, such that the functionality request $r = (I_r, O_r)$ can be totally satisfied by $s^f.I$ and $s^f.O$, respectively. Assume that a subset of web services $S^f = \{s_1^f, s_2^f,\cdots, s_h^f\}$, $S^f \subseteq S$, are discovered as functionally feasible solutions to the given problem. After further QoS filtering and selection on a set of QoS constraints $C = \{c^1, c^2,\cdots,c^n\}$, we shrink the functionally feasible web services $S^f = \{s_1^f, s_2^f,\cdots, s_h^f\}$ to a smaller group

of services $S^q = \{s_1^q, s_2^q, \cdots, s_m^q\}$, where $m \leq h$ and the QoS values of each service $s_i^q.Q$ $(1 \leq i \leq m)$ can satisfy $C = \{c^1, c^2, \cdots, c^n\}$.

Based on the process of functionality matchmaking and QoS filtering, we finally rank all of the services in S^q by their aggregated QoS, among which an optimal solution, $s^* \in S^q$, is not only functionally feasible, but also ranked as the best one with the highest QoS. We elaborate how to generate solutions to a Q-WSD problem, and rank all of the services with their comprehensive QoS values after functionality matchmaking and QoS filtering in the subsequent sections.

4 QoS-based automatic service discovery

We first give the overall framework of QoS-aware WSD. Then, we present service functionality matchmaking, which is followed by QoS filtering and selection, QoS normalisation, QoS weight assignment scheme, and service ranking.

4.1 Framework of the Q-WSD approach

We develop a comprehensive approach to effectively solve Q-WSD problem. Figure 1 illustrates an overview of our approach. Apart from those components for performing service discovery tasks, there are additional two service roles: service provider and service requester. Service providers publish web services to a service repository, while service requesters consume those published web services.

Our approach has six steps.

1 Service providers publish their services including both functionality and QoS information.

2 Parse web services in the service repository and store their functionality and QoS into a service database.

3 A service requester submits a discovery request which consists of a functionality request, multiple QoS constraints, and their associated service requester preferences on the specified QoS criteria.

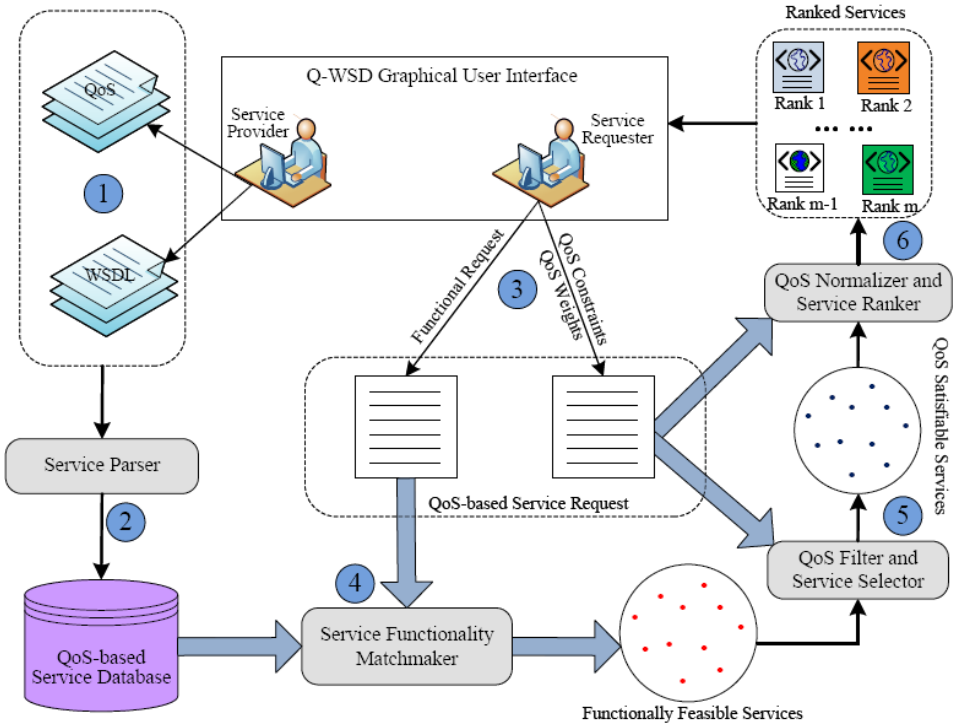
4 Perform functionality logic service matchmaking to discover a set of functionally feasible web services, each of which satisfies the functionality request.

5 Depending on the QoS constraints, we filter out those services that cannot fulfil the QoS constraints and only remain the functionally feasible and QoS satisfiable services.

6 Normalise the QoS values of each service that can satisfy both the functionality request and the specified QoS constraints. Finally, we calculate the aggregated QoS of each service and rank them by comprehensive QoS.

From the overview of the approach, we find that the QoS-based service discovery includes two phases: service registration and service discovery. Given a service request, we mainly focus on how to effectively find a set of ranked services by the satisfiability of their functionality and QoS constraints. Thus, we omit the explanation of the service registration process in this work and elaborate the service discovery.

Figure 1 Overview of the approach for QoS-based automatic service discovery (see online version for colours)



4.2 *Service functionality matchmaking*

Given a set of available web services $S = \{s_1, s_2, \cdots\}$ and a service functionality request $r = (I_r, O_r)$, the service functionality matchmaking is responsible for finding a subset of services $S^f = \{s_1^f, s_2^f, \cdots, s_h^f\}$ from S, such that each of discovered service $s_i^f \in S^f$ $(1 \le i \le h)$ can satisfy the request $r = (I_r, O_r)$. That is, the input parameters s_i^f $(1 \le i \le h)$ can be provided by the input parameters I_r of request r. Conversely, the output parameters O_r of request r must be achieved from the output parameters of s_i^f.

The satisfiability relationship of input and output parameters between a functionally feasible service s_i^f and a functionality request $r = (I_r, O_r)$ is derived as:

$$s_i^f.I \subseteq I_r \tag{1}$$

$$O_r \subseteq s_i^f.O \tag{2}$$

Based on above relationship analysis, given a set of services $S = \{s_1, s_2, \cdots\}$ and a functionality request $r = (I_r, O_r)$, Algorithm 1 shows the process of service functionality matchmaking.

Algorithm 1: Service Functionality Matchmaking

Input: a set of web services $S = \{s_1, s_2, \cdots\}$;
a functionality request $r = (I_r, O_r)$;

Output: functionally feasible services $S^f = \{s_1^f, s_2^f, \cdots, s_h^f\}$;

```
1   S^f ← ∅;
2   foreach s^i ∈ S do
3       included ← true;
4       foreach I^j ∈ s^i.I do
5           if I^j ∉ I^r then
6               included ← false; break;
7       if included then
8           foreach O_r^k ∈ O_r do
9               if O_r^k ∉ s^i.O then
10                  included ← false; break;
11      if included then
12          S^f ← S^f ∪ {s^i};
13  return S^f;
```

During the service functionality matchmaking, Algorithm 1 loops each $s^i \in S$ to check out whether it is functionally satisfied by the given request (lines 2–12). For a service $s^i \in S$, it first matches each input parameter $I^j \in s^i.I$ with the inputs of request I_r, if all the input parameters $s^i.I$ are subsumed by I_r, the algorithm continues the validation for the output parameters of s^i, otherwise s^i is excluded as one of the feasible candidate services (lines 4–6). In the second phase, it validates each output parameter $O_r^k \in O_r$ in the service request with the output parameters of service $s^i.O$. If all the outputs in O_r can be provided by the outputs of s^i (lines 7–10), the service s^i is added as a functionally feasible service in S^f (lines 11–12).

After matching input and output parameters of each service with a given service request, the algorithm returns a subset of feasible services $S^f = \{s_1^f, s_2^f, \cdots, s_h^f\}$, each of which satisfies the equations (1) and (2).

For the computational complexity of service functionality matchmaking, let Q-WSD = (S, C, W, I_r, O_r) be a Q-WSD problem, where $S = \{s_1, s_2, \cdots, s_N\}$ is a service repository with N services, $C = \{c^1, c^2, \cdots, c^n\}$ is a set of QoS constraints, $W = \{w_1, w_2, \ldots, w_n\}$ is a set of QoS weights, and $I_r = \{I_r^1, I_r^2, \cdots\}$ is a set of input parameters provided as initial conditions, and $O_r = \{O_r^1, O_r^2, \cdots\}$ is a set of output parameters as desired goal specifications. For each $s \in S$, we denote the number of input and output parameters as $P_s^I = |s.I|$ and $P_s^O = |s.O|$, respectively. Suppose that $P = \max_{s \in S}\{|s.I| + |s.O|\}$ is used to denote the maximum number of input and output parameters among all the services in S. Here, a service request r has the same assumption on its input and output

parameters. In the phase of service functionality matchmaking, it costs the time to compare the inputs and outputs between a request and each service in S, so we denote it as

$$TF = O\left(\sum_{i=1}^{|S|}\left(\sum_{j=1}^{|s_i.I|}2+\sum_{k=1}^{|O_r^k|}2+3\right)\right) = O\left(\sum_{i=1}^{|S|}(2P+3)\right) = O\big(2N*(P+3)\big) \tag{3}$$

4.3 QoS filtering and service selection

After the service functionality matchmaking, a subset of functionally feasible services $S^f = \{s_1^f, s_2^f, \cdots, s_h^f\}$ can be found from a given service repository S. Each feasible service in S^f provides the same functionality desired by a service requester without any discrimination. However, apart from service functionality, one observation is that service providers also offer their services with multiple non-functional QoS values. Therefore, it is mandatory to further filter out web services from $S^f = \{s_1^f, s_2^f, \cdots, s_h^f\}$ in order to ensure that all of the selected services can also satisfy multiple QoS constraints.

As mentioned in Section 3, five QoS criteria *qos* ={*execution time*, *execution price*, *availability*, *probability of success*, *reputation*} are applied for service providers to publish QoS values. Accordingly, we use $s.Q = \{q_{time}(s), q_{price}(s), q_{avail}(s), q_{succ}(s), q_{rep}(s)\}$ to denote the QoS values of a service s on each criterion. Here,

1 $q_{time}(s)$ is the execution time of service s, which measures the expected duration between the moment when a service request is sent and the moment when the result is returned to a user.

2 $q_{price}(s)$ refers to the execution cost of s, which service requesters have to pay for the invocation of s.

3 $q_{avail}(s)$ is the QoS value of availability of s, which measures the accessibility of the service in a specified time interval.

4 $q_{succ}(s)$ represents the QoS value of the probability of success of s, which computes the probability that s can be successfully executed within a given number of invocations.

5 $q_{rep}(s)$ is the QoS value of reputation of s, which calculates its trustworthiness by the average of a group of feedback from users.

Given QoS values of a service $s.Q = \{q_{time}(s), q_{price}(s), q_{avail}(s), q_{succ}(s), q_{rep}(s)\}$, we compare them with five QoS constraints $C = \{c_1, c_2, c_3, c_4, c_5\}$ submitted by a user, where c_1 = (*execution time*, ≤, v_{time}), c_2 = (*execution price*, ≤, v_{price}), c_3 = (*availability*, ≥, v_{avail}), c_4 = (*probability of success*, ≥, v_{succ}), and c_5 = (*reputation*, ≥, v_{rep}). The QoS filtering on a service s is performed by five inequalities as below.

$$q_{time}(s) \le c_1.v_{time} \tag{4}$$

$$q_{price}(s) \le c_2.v_{price} \tag{5}$$

$$q_{avail}(s) \ge c_3.v_{avail} \tag{6}$$

$$q_{succ}(s) \geq c_4 . v_{succ} \tag{7}$$

$$q_{rep}(s) \geq c_5 . v_{rep} \tag{8}$$

From the above five inequalities (4)–(8), we filter out those services from $S^f = \{s_1^f, s_2^f, \cdots, s_h^f\}$ where their QoS values cannot be satisfied by multiple QoS constraints. Thus, a subset of services $S^q = \{s_1^q, s_2^q, \cdots, s_m^q\}$ are selected from S^f. Each service $s_i^q \in S^q$ $(1 \leq i \leq m)$ not only represents a feasible solution to a functionality request, but also satisfies all the QoS constraints.

For the time complexity, during the QoS filtering we check the QoS values of each service to find out those services that can satisfy the QoS constraints, so the time spent on the QoS filtering is $T_Q = O(h * n)$, where h and n are the number of services discovered by a functionality request and QoS criteria.

After the QoS filtering and service selection, for each service $s_i^q \in S^q$, it both satisfies functionality request and QoS constraints. Furthermore, all these satisfiable services should be returned to a requester with service ranking by the calculation of their aggregated QoS.

4.4 QoS normalisation and utility aggregation

Given a number of QoS criteria $qos = \{q^1, q^2, \cdots, q^n\}$, we model each service in $S^q = \{s_1^q, s_2^q, \cdots, s_m^q\}$ as a row with n QoS values in a QoS value matrix $(QoS^q)_{m*n}$, which can be represented as

$$QoS^q = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ v_{m1} & v_{m2} & \cdots & v_{mn} \end{bmatrix} \tag{9}$$

where v_{ij} is the QoS value on the QoS criterion q^j in the service s_i^q, and we have $1 \leq i \leq m$, $1 \leq j \leq n$.

When calculating aggregated QoS of a service $s_i^q \in S^q$, we adopted a weighted sum of values on the five specified QoS criteria. Since they have different ranges, we cannot avoid frequent case, where several high QoS values on some QoS criteria in a service reduce the discrimination of those low QoS values on other several QoS criteria in the same service. Thus, we need to normalise the QoS values to the range of [0, 1] before calculating them in weighted QoS sum. Depending on the features of QoS criteria, QoS normalisation strategy for a service is classified for positive QoS criteria and negative ones.

For positive QoS criteria, such as *availability*, *probability of success*, and *reputation*, they are denoted better quality by higher QoS values. Given a service $s_i^q \in S^q$, for a positive QoS criterion which is indexed in the j^{th} column in QoS^q, the normalised QoS value of v_{ij} is calculated by the strategy as

$$v'_{ij} = \begin{cases} \dfrac{v_{ij} - \min_j}{\max_j - \min_j}, & \text{if } \max_j \neq \min_j \\ 1, & \text{otherwise} \end{cases} \tag{10}$$

where v_{ij} is the QoS value on the QoS criterion q^j in s_i^q, $\max_j$ and $\min_j$ are respectively the maximum and minimum QoS values on the j^{th} QoS criterion q^j among all the m services in QoS^q. v'_{ij} is the normalised QoS value.

For negative QoS criteria, such as *execution time* and *execution price*, they are denoted lower quality by higher values. Symmetrically, given a service $s_i^q \in S^q$, for a negative QoS criterion which is located in the j^{th} column in QoS^q, the normalised QoS value of v_{ij} is calculated as

$$v'_{ij} = \begin{cases} \dfrac{\max_j - v_{ij}}{\max_j - \min_j}, & \text{if } \max_j \neq \min_j \\ 1, & \text{otherwise} \end{cases} \tag{11}$$

After QoS normalisation, we transfer the original QoS matrix QoS^q into a normalised one QoS_N^q, where each element v'_{ij} keeps in the range of [0, 1]. The normalised QoS matrix is represented as

$$QoS_N^q = \begin{bmatrix} v'_{11} & v'_{12} & \cdots & v'_{1n} \\ v'_{21} & v'_{22} & \cdots & v'_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ v'_{m1} & v'_{m2} & \cdots & v'_{mn} \end{bmatrix} \tag{12}$$

After the conversion from QoS^q to its normalised QoS matrix QoS_N^q, the aggregated QoS of a service is calculated using a weighted sum of QoS values from a row in QoS_N^q.

$$QoS\left(s_i^q\right) = \sum_{j=1}^{n} w_j * v'_{ij} \tag{13}$$

where $QoS(s_i^q)$ is the aggregated QoS of a service $s_i^q \in S^q$, w_j is the service requester preference weight on the QoS criterion q^j, and v'_{ij} is the normalised QoS value of the service s_i^q on the j^{th} QoS criterion q^j in QoS_N^q.

For the time complexity of QoS normalisation, we take the most efficient algorithm to find out the maximum and minimum QoS values for each criterion, i.e., we take the time complexity $O(3m / 2)$ for searching the maximum and minimum QoS values on each criterion, where m is the number of services selected by QoS filtering. As a result, the complexity of QoS normalisation is $T_N = O((3m / 2 + m) * n) = O(5m * n / 2)$.

When calculating the aggregated QoS for a web service, one of the QoS weight assignment scheme on QoS criteria is given by an end user. In many cases, service requesters cannot decide how much service requester preferences on QoS criteria they should assign for service ranking. We discuss three QoS weight assignment scheme in the following.

4.5 QoS weight assignment

For ranking all the services in $S^q = \{s_1^q, s_2^q, \cdots, s_m^q\}$, the aggregated QoS of each service must be calculated with the weight preferences on QoS criteria. To adapt to the QoS weight assignment, three weight assignment scheme are as below for different application scenarios, including objective weight assignment by the calculation of aggregated QoS of services, subjective weight assignment by users, and comprehensive weight assignment.

4.5.1 Objective weight assignment

By using normalised QoS values in QoS_N^q, we derive the QoS weight for each QoS criterion. For a criterion $q^j \in qos$, we get a vector of QoS values $(v'_{1j}, v'_{2j}, \cdots, v'_{mj})$, and then calculate the ratio of each QoS value v'_{ij} in the vector as:

$$p_{ij} = \frac{v'_{ij}}{\sum_{k=1}^{m} v'_{kj}} \tag{14}$$

where v'_{ij} is the QoS value on the ij^{th} position in QoS_N^q. m represents the total number of services in S^q. By doing so, the information entropy of the j^{th} QoS criterion is calculated by each p_{ij} $(1 \le i \le m)$ as follows.

$$E_j = -(\ln m)^{-1} * \sum_{i=1}^{m} p_{ij} * \ln p_{ij} \tag{15}$$

where $E_j (1 \le j \le n)$ is the information entropy of the j^{th} QoS criterion, which denotes better with lower value. For the QoS weight assignment, we consider all of them on each QoS criterion. That is, the objective QoS weight is assigned as follows.

$$w_j = \frac{1 - E_j}{n - \sum_{k=1}^{n} E_k} \tag{16}$$

where n is the total number of QoS criteria. The QoS weight attached on a QoS criterion holds a relative proportion by comparing its information entropy with the summation on all the QoS criteria.

4.5.2 Subjective weight assignment

In comparison with objective assignment of QoS weights, the subjective weight assignment refers to the direct specification on n QoS weights by a service requester. The specified weights must satisfy

$$\sum_{j=1}^{n} w_j = 1 \tag{17}$$

where each w_j represents the service requester preference on the QoS criterion q^j, and $0 \leq w_j \leq 1$.

4.5.3 Comprehensive weight assignment

In conjunction with objective and subjective QoS weight assignment, we combine them as the comprehensive preference to assign a QoS weight on a QoS criterion. That is, we assign the QoS weight w_j by two QoS weights.

$$w_j = \alpha * w_j^o + \beta * w_j^s \tag{18}$$

where w_j^o and w_j^s are respectively the QoS weights obtained by objective and subjective assignment on the QoS criterion q^j. Especially, α and β represent proportional values on these two weights when merging them into a comprehensive QoS weight. For proportional value α, we calculate it by

$$\alpha = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} w_j^o * v'_{ij}}{\sum_{i=1}^{m}\sum_{j=1}^{n} w_j^o * v'_{ij} + \sum_{i=1}^{m}\sum_{j=1}^{n} w_j^s * v'_{ij}} \tag{19}$$

The proportional value α is determined by the proportion between the sum of the aggregated QoS of each service in S^q by objective weight assignment and the sum of the aggregated QoS of each service in S^q by objective and subjective weight assignments. Symmetrically, for the proportional valueβ, it is determined by

$$\beta = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} w_j^s * v'_{ij}}{\sum_{i=1}^{m}\sum_{j=1}^{n} w_j^o * v'_{ij} + \sum_{i=1}^{m}\sum_{j=1}^{n} w_j^s * v'_{ij}} \tag{20}$$

With the two parameters α and β, for a QoS criterion q^j, we comprehensively assign its QoS weight w_j by the given w_j^o and w_j^s.

Considering the worst case, we use the comprehensive QoS weight assignment scheme as an example to analyse the time complexity. It includes the case of objective QoS weight assignment, which costs the time of computing QoS value ratio, information entropy, and the weight of each QoS criterion. Thus, it is denoted as $T'_W = O((m + m) * n + m * n + n) = O(3mn + n)$. As a result, the time complexity of comprehensive weight assignment is $TW = T'_W + \mathrm{O}(2 * m * n + 2 * n) = \mathrm{O}(5mn + 3n)$.

4.6 QoS-based service discovery algorithm

Based on above service functionality matchmaking, QoS filtering and selection, QoS normalisation and weight assignment, we describe the overall QoS-based service discovery approach in the Algorithm 2. The algorithm takes a Q-WSD = (S, C, W, I_r, O_r)

as input, and outputs a set of ranked services $S^r = \{s_1^r, s_2^r, \cdots, s_m^r\}$, each of which can both satisfy service functionality request and multiple QoS constraints.

Algorithm 2: Q-WSD

Input: a QoS-based service discovery problem, Q-WSD=(S, C, W, I_r, O_r);

Output: a set of ranked services $S^r = \{s_1^r, s_2^r, \cdots, s_m^r\}$;

1 $S^f \leftarrow \emptyset$; $S^q \leftarrow \emptyset$; $S^r \leftarrow \emptyset$;

2 $QoS^q \leftarrow \emptyset$; $QoS_N^q \leftarrow \emptyset$;

3 Invoke **Algorithm 1** (S, I_r, O_r);

4 Match $S^f = \{s_1^f, s_2^f, \cdots, s_h^f\}$;

5 QoS filtering with $C = \{c^1, c^2, \cdots, c^n\}$ on S^f;

6 Select $S^q = \{s_1^q, s_2^q, \cdots, s_m^q\}$;

7 Generate $m * n$ QoS matrix QoS^q;

8 QoS normalization on QoS^q;

9 Convert QoS^q into QoS_N^q;

10 Specify a QoS weight assignment scheme;

11 Get $W = \{w_1, w_2, \cdots, w_n\}$;

12 Calculate $QoS(s_i^q) = \sum_{j=1}^{n} w_j * v'_{ij}$;

13 Rank services in S^q by $QoS(s_i^q)$;

14 Get $S^r = \{s_1^r, s_2^r, \cdots, s_m^r\}$;

15 **return** S^r;

Algorithm 2 first invokes Algorithm 1 to find those services $S^f = \{s_1^f, s_2^f, \cdots, s_h^f\}$, each of which can satisfy functionality request (lines 3–4). Then, we filter out those services that violate the QoS constraints $C = \{c^1, c^2, \cdots, c^n\}$ from S^f, thus a subset of services remain and store in $S^q = \{s_1^q, s_2^q, \cdots, s_m^q\}$ where each of them can also satisfy the specified multiple QoS constraints (lines 5–6). By using the selected services in S^q, the third step generates an $m * n$ QoS matrix QoS^q, and it is converted into a normalised one QoS_N^q by QoS normalisation strategy (lines 7–9). After that, with the designation of a QoS weight assignment, we get n weight preferences $W = \{w_1, w_2, \ldots, w_n\}$ on QoS criteria (lines 10–11). Finally, we calculate aggregated QoS of each web service in $S^q = \{s_1^q, s_2^q, \cdots, s_m^q\}$, and rank them as a set of sorted services in a descending order $S^r = \{s_1^r, s_2^r, \cdots, s_m^r\}$ (lines 12–14).

For the calculation of aggregated QoS of each service, we take a weighted sum of normalised QoS values with the assigned weights. So it is denoted as $T_{QoS} = O(\sum_{i=1}^{m} \sum_{j=1}^{n} 1) = O(m * n)$. To rank the selected services by QoS filtering, heap sorting algorithm with priority queue is applied to rank services by their aggregated QoS,

so the complexity is $T_R = O(m * \log_2 m)$, where m is the number of services selected by QoS filtering and service selection.

4.7 Analysis of computational complexity

Under the assumption of parameters that we have applied for previous analyses of time complexity, the computational complexity of Q-WSD is determined by six parts: service functionality matchmaking, QoS filtering, QoS normalisation, QoS weight assignment, aggregated QoS calculation, and service ranking.

With the combination of all the time complexity analyses above, the total complexity of the approach is $T = T_F + T_Q + T_N + T_W + T_{QoS} + T_R = O(2N * (P + 3) + h * n + 5m * n / 2 + 5m * n + 3n + m * n + m\log_2 m)$. Since in a large-scale service repository, we have the inequalities: $N >> P$, $N >> n$, $N > h$, $h > m$, and $m > n$. As a result, the time complexity is $T = O(2P * N + h * n + 17m * n / 2 + m * \log_2 m) = O(P * N + h * n + m * n + m * \log_2 m) = O(P * N + h * n + m * \log_2 m)$.

Table 1 Input and output parameters of each web service s

#	*Service*	*s.I*	*s.O*
1	s_1	$\{I^1, I^2, I^3\}$	$\{O^1, O^2, O^5, O^6\}$
2	s_2	$\{I^1, I^4\}$	$\{O^2, O^3, O^4\}$
3	s_3	$\{I^1, I^3, I^5\}$	$\{O^1, O^2, O^3, O^5\}$
4	s_4	$\{I^1, I^3, I^5, I^8\}$	$\{O^2, O^5, O^7, O^8, O^9\}$
5	s_5	$\{I^1, I^3\}$	$\{O^2, O^5\}$
6	s_6	$\{I^2, I^5, I^8\}$	$\{O^3, O^5, O^6\}$
7	s_7	$\{I^3, I^7, I^9\}$	$\{O^7, O^9\}$
8	s_8	$\{I^1, I^3, I^6, I^7\}$	$\{O^2, O^4, O^5\}$
9	s_9	$\{I^1, I^2, I^3\}$	$\{O^2, O^5\}$

Note: Column '*s.I*' denotes the input parameters of s, and '*s.O*' denotes the output parameters of s.

From the complexity analysis, we find that the computational complexity of a Q-WSD problem is dominated by the linear time of the total number of services in a service repository, the number of functionally feasible services, and the time of ranking services. Thus, for a large-scale service repository, the approach is almost a linear algorithm with the number of services. Thus, it can be efficiently performed in a polynomial time.

5 Empirical evaluation

In this section, we present an initial simulated Q-WSD problem to validate the feasibility of our approach. The empirical running example scenario includes service functionality match-making, QoS filtering and service selection, QoS normalisation, QoS weight assignment, and service ranking.

5.1 Empirical Q-WSD problem

Given a Q-WSD problem Q-WSD = (S, C, W, I_r, O_r), suppose that it has nine services, $S = \{s_1, s_2,\ldots,s_9\}$, and each of service has a set of input and output parameters, as shown in Table 1. For the QoS of a service, assume that its provider offers five QoS values by a set of specified QoS criteria, as illustrated in Table 2. Suppose that a requester submits a set of QoS constraints C = {(*time*, ≤, 35), (*price*, ≤, 28), (*avail*, ≥, 0.85), (*success*, ≥, 0.87), (*reputation*, ≥, 3.6)} on the five QoS criteria. Accordingly, the service requester also specifies a set of preferences $W = \{0.1, 0.5, 0.1, 0.2, 0.1\}$, and submits a service functionality request $r = (I_r, O_r)$ that includes two initial input parameters $I_r = \{I^1, I^3\}$ and two desired output parameters $O_r = \{O^2, O^5\}$.

The Q-WSD problem described above will be taken into account throughout the empirical evaluation. By doing so, we validate the feasibility of the approach for QoS-based automatic discovery of web services.

Table 2 Multiple QoS values of each web service in $S^f = \{s_1, s_3, s_4, s_5, s_8, s_9\}$

#	*Service*	*Time*	*Price*	*Avail*	*Success*	*Reputation*
1	s_1	15	9	0.85	0.97	4.2
2	s_3	20	16	0.79	0.89	3.6
3	s_4	13	25	0.95	0.90	4.5
4	s_5	9	34	0.88	0.86	3.7
5	s_8	28	14	0.96	0.87	4.8
6	s_9	32	7	0.80	0.93	3.9

Notes: For a web service, five QoS values are assigned on the specified QoS criteria. Column '*time*', '*price*', '*avail*', '*success*', '*reputation*' represent *execution time*, *execution price*, *availability*, *probability of success*, and *reputation*.

5.2 Functionality matchmaking and QoS filtering

During the service functionality matchmaking, we consider the Q-WSD problem in Section 5.1, where we only take its functionality request $r = (I_r, O_r)$ as an example. The input and output parameters are $I_r = \{I^1, I^3\}$ and $O_r = \{O^2, O^5\}$, respectively. After service functionality matchmaking, Algorithm 1 discovers six functionally feasible web services $S^f = \{s_1, s_3, s_4, s_5, s_8, s_9\}$, each of which satisfies the functionality request r.

For the QoS filtering and service selection, we still take the specified multiple QoS constraints that are shown in Section 5.1. There are five QoS constraints on the predefined QoS criteria, C = {(*time*, ≤, 35), (*price*, ≤, 28), (*avail*, ≥, 0.85), (*success*, ≥, 0.87), (*reputation*, ≥, 3.6)}. After QoS filtering and service selection, we further filter out those services that cannot satisfy the QoS constraints C from six functionally feasible services $S^f = \{s_1, s_3, s_4, s_5, s_8, s_9\}$. By doing so, s_5 does not satisfy C. Thus, we shrink S^f and only remain a subset of QoS satisfiable services $S^q = \{s_1, s_3, s_4, s_8, s_9\}$.

Note that throughout the empirical experiments, we select the parameters in terms of the QoS values of the simulated services. In real-world WSD application scenarios, however, the service requesters need to decide the setting of different parameters from the view of their real requirements, including the preferences on QoS criteria and multiple QoS constraints.

5.3 QoS normalisation and weight assignment

For the QoS normalisation, we reconsider the QoS-based service discovery problem Q-WSD = (S, C, W, I_r, O_r), as shown in Section 5.1. After service matchmaking, QoS filtering and service selection, a subset of QoS satisfiable services Sq = {s_1, s_3, s_4, s_8, s_9} remain for QoS normalisation. So we set up an original QoS matrix and normalise their QoS values. By using the QoS normalisation strategy in the equations (10) and (11), the normalised QoS values of each service in S^q are shown in Table 3.

In the QoS weights assignment, we first use QoS weights W = {0.1, 0.5, 0.1, 0.2, 0.1} specified in Section 5.1 as the subjective QoS assignment scheme. For the objective QoS weight assignment, we calculate each QoS weight by the equations (14)–(16) based on the normalised QoS matrix derived from Table 3. Finally, we combine the QoS weights from objective and subjective assignment scheme, so that comprehensive QoS weights can be calculated by using the equations (18)–(20). Table 4 shows QoS weights on five QoS criteria amongst three weight assignment scheme.

Table 3 The normalised QoS values of each service in S^q

#	*Service*	*Time*	*Price*	*Avail*	*Success*	*Reputation*
1	s_1	0.895	0.889	0.353	1	0.50
2	s_3	0.632	0.50	0	0.20	0
3	s_4	1	0	0.941	0.30	0.750
4	s_8	0.211	0.611	1	0	1
5	s_9	0	1	0.059	0.60	0.250

Note: The original QoS values of the corresponding web service are shown in Table 2.

Table 4 The QoS weights on the specified QoS criteria by objective, subjective and comprehensive QoS weight assignment scheme

Scheme	*Time*	*Price*	*Avail*	*Success*	*Reputation*
Objective	0.1860	0.1423	0.2747	0.2167	0.1803
Subjective	0.1	0.5	0.1	0.2	0.1
Comprehensive	0.1414	0.3278	0.1841	0.208	0.1387

Notes: The QoS weights are specified by the service requesters in subjective assignment way, while they are calculated in objective assignment way by the aggregated QoS of services shown in Table 3. The QoS weights of comprehensive assignment way are calculated together by subjective and objective assignment scheme.

5.4 Service ranking and empirical analysis

Based on QoS weights of three assignment scheme generated in Section 5.3, we calculate the aggregated QoS of each service in S^q = {s_1, s_3, s_4, s_8, s_9} discovered and filtered in Section 5.2, by using the QoS aggregation utility function in equation (13). The normalised QoS values of each service is shown in Table 3. The aggregated QoS of each service is shown in Table 5. On the basis of these results on the aggregated QoS of web services, we apply an efficient sorting algorithm (heap sort algorithm) to rank these services.

Table 5 The aggregated QoS of each service in S^q by three different QoS weight scheme

#	*Service*	*Objective*	*Subjective*	*Comprehensive*
1	s_1	0.6968	0.8193	0.7603
2	s_3	0.2320	0.3532	0.2949
3	s_4	0.6447	0.3291	0.4811
4	s_8	0.5812	0.5266	0.5529
5	s_9	0.3336	0.6509	0.4981

Note: The normalised QoS values of each service are shown in Table 3.

From the aggregated QoS of each service shown in Table 5, we rank the services from the perspective of three weight assignment scheme. In terms of objective QoS weight assignment, the services are ranked as $\{s_1, s_4, s_8, s_9, s_3\}$. On the contrary, they are ranked as $\{s_1, s_9, s_8, s_3, s_4\}$ in the subjective QoS weight assignment. Finally, in conjunction with the above two schemes, these services are ranked as $\{s_1, s_8, s_9, s_4, s_3\}$.

6 Conclusions and future work

Automatic and effective WSD can simplify the implementation of business processes in SOA. This paper presents an integrated QoS-based approach for automatic discovery of web services under multiple QoS constraints and service requester preferences, and proposes a number of novel techniques, including functionality matchmaking, QoS filtering, QoS normalisation, QoS weight assignment, and service ranking.

The method first performs the service persistence registration by parsing input and output interfaces of web services. Then, the method discovers functionally feasible services by service functionality matchmaking. A subset of those services are further filtered out with QoS filtering and service selection by the satisfiability of multiple QoS constraints. Subsequently, the method normalises the QoS values of all the QoS satisfiable services by normalisation strategy, and calculates the aggregated QoS of each service with utility function. Finally, the methods ranks and compares these services from three provided QoS weight assignment scheme with an efficient off-the-shelf sorting algorithm. We conduct empirical experiments on simulated web services. The experimental results validate the feasibility of our Q-WSD approach.

Our future work includes three directions. The first one is the extension of our current approach to support service functionality matchmaking in semantic level, which plans to integrate semantic descriptions into input and output interfaces with existing domain ontologies. The second one is to take the dynamics of multiple QoS values of services into account in real applications and plans to analyse the nature of dynamically changing QoS by uncertain evaluation strategies. The third direction we plan to investigate is to conduct more experimental experiments to further validate the effectiveness and efficiency of our approach in large-scale and real-world service repositories.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (61303096, 61300100), Shanghai Natural Science Foundation (13ZR1454600, 13ZR1451000), an Innovation Program of Shanghai Municipal Education Commission (14YZ017), and a Specialized Research Fund for the Doctoral Program of Higher Education (20133108120029).

We also would like to appreciate all the four anonymous reviewers for their insightful suggestions and comments that can greatly improve the quality of our manuscript.

References

Al-Masri, E. and Mahmoud, Q.H. (2009) 'Web service discovery and client goals', *Computer*, Vol. 42, No. 1, pp.104–107.

Al-Masri, E. and Mahmoud, Q.H. (2010) 'WSB: a broker-centric framework for quality-driven web service discovery', *Software: Practice and Experience*, Vol. 40, No. 10, pp.917–941.

Amorim, R., Claro, D.B., Lopes, D., Albers, P. and Andrade, A. (2011) 'Improving web service discovery by a functional and structural approach', *Proceedings of the IEEE International Conference on Web Services (ICWS)*.

Atkinson, C., Bostan, P., Hummel, O. and Stoll, D. (2007) 'A practical approach to web service discovery and retrieval', *Proceedings of the IEEE International Conference on Web Services (ICWS)*.

Cubo, J., Canal, C. and Pimentel, E. (2010) 'Context-aware service discovery and adaptation based on semantic matchmaking', *Proceedings of the International Conference on Internet and Web Applications and Services (ICIW)*.

Dietze, S., Benn, N., Domingue, J., Conconi, A. and Cattaneo, F. (2009a) 'Two-fold service matchmaking-applying ontology mapping for Semantic Web service discovery', *Proceedings of the Asian Semantic Web Conference (ASWC)*.

Dietze, S., Gugliotta, A. and Domingue, J. (2009b) 'Exploiting metrics for similarity-based Semantic Web service discovery', *Proceedings of the IEEE International Conference on Web Services (ICWS)*.

Dong, X., Halevy, A., Madhavan, J., Nemes, E. and Zhang, J. (2004) 'Similarity search for web services', *Proceedings of the International Conference on Very Large Data Bases (VLDB)*.

Elgazzar, K., Hassan, A.E. and Martin, P. (2010) 'Clustering WSDL documents to bootstrap the discovery of web services', *Proceedings of the IEEE International Conference on Web Services (ICWS)*.

Ganesan, P., Garcia-Molina, H. and Widom, J. (2003) 'Exploiting hierarchical domain structure to compute similarity', *ACM Transactions on Information Systems (TOIS)*, Vol. 21, No. 1, pp.64–93.

Hau, J., Lee, W. and Darlington, J. (2005) 'A semantic similarity measure for Semantic Web services', *Proceedings of the International World Wide Web Conference (WWW)*.

Klusch, M., Fries, B. and Sycara, K. (2006) 'Automated Semantic Web service discovery with OWLS-MX', *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Lemos, F., Gater, A., Grigori, D. and Bouzeghoub, M. (2012a) 'A framework for service discovery based on structural similarity and quality satisfaction', *Proceedings of the International Conference on Web Engineering (ICWE)*.

Lemos, F., Grigori, D. and Bouzeghoub, M. (2012b) 'Adding non-functional preferences to service discovery', *Proceedings of the International Conference on Web Engineering (ICWE)*.

Meditskos, G. and Bassiliades, N. (2010) 'Structural and role-oriented Web service discovery with taxonomies in OWL-S', *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, Vol. 22, No. 2, pp.278–290.

Nayak, R. (2008) 'Data mining in web services discovery and monitoring', *International Journal of Web Services Research (IJWSR)*, Vol. 5, No. 1, pp.63–81.

Paliwal, A.V., Shafiq, B., Vaidya, J., Xiong, H. and Adam, N. (2012) 'Semantics-based automated service discovery', *IEEE Transactions on Services Computing (TSC)*, Vol. 5, No. 2, pp.260–275.

Paolucci, M., Kawamura, T., Payne, T.R. and Sycara, K. (2002) 'Semantic matching of web services capabilities', *Proceedings of the International Semantic Web Conference (ISWC)*.

Patel, P. and Chaudhary, S. (2009) 'Context aware semantic service discovery', *Proceedings of the World Conference on Services (SERVICES)*.

Plebani, P. and Pernici, B. (2009) 'Urbe: web service retrieval based on similarity evaluation', *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, Vol. 21, No. 11, pp.1629–1642.

Rambold, M., Kasinger, H., Lautenbacher, F. and Bauer, B. (2009) 'Towards autonomic service discovery – a survey and comparison', *Proceedings of the IEEE International Conference on Services Computing (SCC)*.

Vijayan, A.S. and Balasundaram, S. (2013) 'Effective web service discovery using K-means clustering', *Proceedings of the International Conference on Distributed Computing and Internet Technology (ICDCIT)*.

Wu, J., Chen, L., Zheng, Z., Lyu, M.R. and Wu, Z. (2012) 'Clustering web services to facilitate service discovery', *Knowledge and Information Systems (KAIS)*, pp.1–23.

Xiao, H., Zou, Y., Ng, J. and Nigul, L. (2010) 'An approach for context-aware service discovery and recommendation', *Proceedings of the IEEE International Conference on Web Services (ICWS)*.

Xu, L., Xu, B., Chen, L. and Yang, H. (2011) 'Web service discovery based on user requirements', *Proceedings of the IEEE International Conference on High Performance Computing and Communications (HPCC)*.

Yu, Q. (2011) 'Place semantics into context: service community discovery from the WSDL corpus', *Proceedings of the International Conference on Service Oriented Computing (ICSOC)*.

Zeng, L., Benatallah, B., Ngu, A.H., Dumas, M., Kalagnanam, J. and Chang, H. (2004) 'QoS-aware middleware for web services composition', *IEEE Transactions on Software Engineering (TSE)*, Vol. 30, No. 5, pp.311–327.