Web service clustering using text mining techniques

Wei Liu* and Wilson Wong

School of Computer Science and Software Engineering
The University of Western Australia
35 Stirling Highway, Crawley, WA 6009, Australia
Fax: +61–8 64881089
E-mail: wei@csse.uwa.edu.au
E-mail: wilson@csse.uwa.edu.au
*Corresponding author

Abstract: The idea of a decentralised, self-organising service-oriented architecture seems to be more and more plausible than the traditional registry-based ones in view of the success of the web and the reluctance in taking up web service technologies. Automatically clustering Web Service Description Language (WSDL) files on the web into functionally similar homogeneous service groups can be seen as a bootstrapping step for creating a service search engine and, at the same time, reducing the search space for service discovery. This paper proposes techniques to automatically gather, discover and integrate features related to a set of WSDL files and cluster them into naturally occurring service groups. Despite the lack of a common platform for assessing the performance of web service cluster discovery, our initial experiments using real-world WSDL files demonstrated the great potential of the proposed techniques.

Keywords: web services; web service clustering; web service discovery; text mining; homogenous service community.


Biographical notes: Wei Liu received her PhD in Information Technology from the University of Newcastle, Australia, in 2003. She is a Lecturer at the School of Computer Science and Software Engineering of the University of Western Australia. Her research focuses on enabling technologies for the semantic web, including agent communication, knowledge acquisition through text mining, as well as issues regarding agent and service-oriented computing. She has published in these areas extensively in national and international journals, conference proceedings as well as book chapters.

Wilson Wong is currently a PhD candidate at the University of Western Australia in the field of Ontology Learning. He received his BIT (Hons) majoring in Data Communication and MSc in the field of Natural Language Processing from Malaysia. He has 17 internationally refereed publications over the period of three years in reputable conferences such as IJCNLP and journals like Data Mining and Knowledge Discovery. His areas of interest include ontology learning, statistical natural language processing and text mining.
1 Introduction

According to a well accepted definition by Peltz (2003), web services are distributed autonomous software components that are self-describing and designed by different vendors to provide certain business functionalities to other applications through an internet connection. They are conceived to leverage existing business processes creation from tightly coupled component-based models, to loosely coupled Service-Oriented Architectures (SOA). Business processes can therefore benefit from the services offered by other organisations, and are no longer limited to within the enterprise’s boundary. 

Note that in the definition, web services are designed to be used by other software programs automatically. However, software programs do not have any cognitive power to understand a programming interface like human programmers do. Despite more than half a decade’s effort, automatically discovering web services is still considered as difficult as looking for a needle in the haystack as shown by Garofalakis et al. (2004). It is widely accepted that the current SOA assumes the interactions between three types of players, namely, the service providers advertise their services with service registries and service consumers query the registries for providers that have matching services. Such registry-based SOA inevitably requires a semi-centralised structure, where registries become the bottleneck for scalability and robustness. In other words, if the registry (or the federation of registries) fails to perform, the service consumers and the service providers are left unconnected. Moreover, the registering of services is a static and labourious process which demands the programmer’s understanding of categorisation in a domain. This is against the open and dynamic nature of the web. Just like the current document-centric web where documents can be added or deleted with no central control, any web service should be free to join and leave the service-oriented web anytime. The registry-based SOA is fundamentally ill-fated because such a system assumes all service providers to register their new services and deregister unavailable services.

In fact, some major providers have even decided to advertise their services through their human-readable websites, rather than service-registries. For example, Google’s and Amazon’s web services all have dedicated web pages for human readers. In contrast, the UDDI Business Registry (UBR) operated collaboratively by Microsoft, IBM and SAP has been shutdown,\(^1\) which further confirmed the intrinsic problem of a registry-based SOA. Recent work by Al-Masri and Mahmoud (2007; 2008) have concrete results showing an increasing trend of adopting a search-engine based model for web service discovery. They compared the number, the liveliness and the validity of services hosted by major UBRs with that of those available through popular search engines, the trends are more than evident:

- The growth rate of UBR registered services from October 2006 to October 2007 is only 12.6%, as compared to a staggering 286% by search engines.
- More than 53% of the UBR hosted services are inactive, whereas 92% of the search engine cached services are active.
- Only 37% of the UBR hosted services have valid service interface that conform to WSDL standard, as compared to the 87% valid services crawled through search engines.
It is clear that registry-based SOA is not a choice for the public. The needs of a truly decentralised SOA without the presence of registries become more pressing than ever. Contemplating the enormous success of the web and the reluctance in taking up the web service idea, it is inevitable for us to revisit the fundamental basis of a SOA. Everyone can publish a web page so long as there is a valid URL for it. There is no need to ‘register’ the web page. Various search engines’ crawlers automatically trawl the web for pages and automatically rank and classify them into groups. The success is not due to any careful business plan, but the convenience and freedom provided by this free publishing search engine model. Can we borrow this idea to produce a search engine for web services, such that service providers can just publish their services with a valid URL and let the service search engine do the job of clustering, classification, match making and composition? What are the basic building blocks of a service-search engine?

To answer these questions, this paper proposed a mechanism for clustering web services to bootstrap a service search engine. This paper takes advantage of a document search engine (e.g., Google) that maybe unconsciously crawling web service description files (e.g., WSDL files), and using these files as seeds to start expanding the discovery of possible features in an attempt to cluster the web services into functionally similar groups. Because of the similar functionalities, we term such service clusters as homogeneous service communities. If the crawling and the clustering process are in continuous operation like a typical search engine does, the approach has the potential of enabling self-organisation of the web as proposed by Liu (2005). The proposed web service clustering approach assumes no registries, and can automatically reduce the search space of web services effectively. Therefore, it can be seen as a predecessor for web service discovery. This paper gathers real service description files from the web as opposed to working on hypothetical examples. The resulting clusters not only provide a useful glimpse on what services are out there, but also an insight into the types of technologies which have proliferated in this area. Theoretically, the paper introduced the use of text mining techniques to effectively separate content words from function words, and a spreading activation inspired algorithm for cluster selection.

The paper is organised as follows, Section 2 discusses the proposed approach in relation to web service discovery and document clustering. Section 3 introduces the overall architecture of the system and the detailed process of feature mining. Section 4 integrates the collected features using a web service cluster discovering algorithm. Section 5 demonstrated the effectiveness of the approach using existing service description files available on the web. Section 6 conclude the paper with an outlook to future work.

2 Related works

Web service discovery, according to Booth et al. (2004), is broadly defined as “the act of locating a machine-processable description of a web service that may have been unknown and that meets certain functional criteria”. As pointed out by Garofalakis et al. (2004), web service discovery mechanisms originated from the agent match-making paradigm by employing a middle or broker agent, then evolve to the various ways of querying through a standard UDDI registry or a cloud of federated UDDI registries. The discovery mechanisms also differ according to the way the web services are described.
Two dominant languages are co-existing in the industry and the academia for describing web services. WSDL is popular and adopted by the industry due to its simplicity, while OWL-S (formerly DAML-S) and WSMO are well accepted by researchers as they offer much structured and detailed semantic mark-ups. Hereafter, by web services we mean the services described in WSDL and semantic web services are for those described using either OWL-S or WSMO. The clear separation is necessary as the techniques required by these two types of languages can be quite different. Garofalakis et al. (2004) and Garofalakis et al. (2006) confirm that the discovery of web services in UDDI registries typically follows an Information Retrieval approach, whereas high-level match-making techniques described by Klusch et al. (2006) are utilised for semantic web services due to the fact that more structured annotations are available for service profiles. However, semantic web services are still only available at the academic level. In order to test out a practical methodology like the one proposed in this paper, instead, we opt for the more readily available format, namely, WSDL-based web services. The simplest approach to query a UDDI registry is the keyword-based information retrieval process of matching a query string against the textual description in the UDDI catalogue and in the tModel. To address the limitation of keyword-based queries, other more sophisticated information retrieval approaches are available, such as representing service descriptions as document vectors as shown by Sajjanhar et al. (2004) and then applying Latent Semantic Indexing (Berry et al., 1995) to reduce the vector space to more significant semantic concepts that characterise the web services.

The clustering of web service files is different from the traditional web service discovery problem because there are no queries to match against. However, the idea of representing a web service using document vectors is still relevant. As we will discuss in Section 3, gathering the features for a WSDL file is not as simple as collecting description documents when assuming no central UDDI registries. Another closely related area is the conventional document or web page clustering. They both involve the discovery of naturally-occurring groups of related documents (be it web pages or WSDL files). However, web service files do not usually contain sufficiently large number of words for use as index terms or features. Moreover, the small number of words present in the web service files are erratic and unreliable. Hence, conventional, detailed linguistic analysis, and statistical techniques using local corpora cannot be applied directly for web service files clustering. The use of link analysis between WSDL files to discover related web services has also been studied. In our experiments, we employed Google API’s search options for discovering web page referral or citation. However, it is discovered that most of the WSDL files do not have related web pages that provide hyperlinks to them. For the few that have hyperlinks referring to them, such WSDL files are typically educational examples for teaching how to program in a service-oriented paradigm. This observation is concurred by Li et al. (2007).

In short, the individual existing techniques borrowed from related research areas such as information retrieval are inadequate for the purpose of discovering functionally-related web service clusters. While there is a small number of existing approaches dedicated to the discovery of web services as mentioned above, most of them remain hypothetical in nature, and have yet to be implemented and tested with real-world datasets. With regard to working on real datasets, Li et al. (2007)’s work, Al-Masri and Mahmoud (2007; 2008) are closest in scope with our work. Li et al. (2007) built a web service investigation system that crawls some human identified service repositories to collect WSDL files.
The harvested WSDL files then went through automatic checking for liveliness and redundancy. Unavailable and redundant references to the WSDL files are removed. Then they carry out some statistical study on the complexity (indicated by file size) and the level of description (by descriptive words and the number of back link pages). Al-Masri and Mahmoud (2007) takes the idea of exploratory study one step further, integrating similar processes of crawling, liveliness checking and validation into a Web Service Crawler Engine (WSCE). As compared to Li et al. (2007), WSCE crawls not only Google cached web services, but also service registries including Microsoft, XMethods, SAP, National Biological Information Infrastructure (NBI), as well as other search engines such as Yahoo, AlltheWeb and Baidu. Therefore, results in Al-Masri and Mahmoud (2008) are much more extensive and reflects a more realistic status of the current web service adoption.

Although our work is carried out in parallel to these work, it is reassuring to see the needs of gaining an birds-eye view of the current status of web service adoption. They recognise the importance of getting to know the current status of web service development as whole. However, these work are of exploratory nature, they can tell us how many services are out there (not behind firewalls), but not what they are. They do recognise the need of carry out further study to assist web service discovery, and emphasised that “a possible solution (for adding comments to WSDL files) is to develop algorithms which can gain illustration from the web services through the design documents or comments in the source code”. Analysing content to obtain the functional capability of the available web services is precisely the focus of our system, which provides the solution to discover relevant content bearing words to be associated with the typically inadequately described web services.

On this basis, we propose an integrated feature mining and clustering approach dedicated to web service clustering, which is an important predecessor to web service discovery, and in turn building a more useful web service search crawler engine. In the following three sections, we will discuss the proposed approach in detail, and then present some experiments using real-world WSDL files.

3 Features mining for web service files

In this paper, we propose a system that can automatically cluster a group of WSDL files obtained by querying a search engine (e.g., Google) based on the type of file (in this case, files with a .wsdl extension). Figure 1 illustrate the process of mining four types of features of a WSDL file, namely, the content, the context, the service host and the service name. In this system, each web service is physically represented by its corresponding WSDL file $s_i$. Collectively, the set of WSDL files to be considered for web service cluster discovery is represented as $S$. For each $s_i$, there are four types of features, namely:

1. the content of the web service is characterised by the application-specific terms located in the WSDL file
2. the context of the web service is represented by the application-specific terms appearing in all index web pages of publicly accessible parent directories of the current directory containing the WSDL file
the service host is the second- and top-level portion of the domain name (i.e., a segment of the authority part of the URI) of the host containing the WSDL file. The service name is the name of the WSDL file.

As one may note, the above four features are by no means the best or the only ones available for describing a web service. However, they are the most accessible and feasible ones to use to conduct this research. These four features are collected, measured and then integrated using a web service clustering algorithm presented in Section 4.

Figure 1  Architecture of features mining

The cascaded word analyser consists of five subsequent modules for processing tokens extracted from WSDL or HTML files to extract application-specific terms as features.

The inclusion of the first and second type of features (i.e., web service content and context) is motivated by the use of index terms to determine document relevance in document retrieval. However, unlike the abundance of index terms in textual documents which can adequately support document retrieval, content-bearing words in WSDL files are limited and difficult to identify. The web service content and context are essentially two sets of application-specific terms \( C \) and \( X \), respectively, which describe the services being offered by the corresponding web service. A cascaded word analyser as shown in Figure 1 is utilised to construct the sets \( C \) and \( X \). The cascaded word analyser begins by tokenising the WSDL or HTML files to construct the initial sets of \( C \) and \( X \), and later remove non-words from these sets. Next, words in the two sets are conflated, and analysed for their content-bearing property to remove function words. The remaining content words in the two sets are then clustered to identify application-specific terms and general computing terms. The latter, which do not contribute to the description of the services being offered by the corresponding web service, are removed, leaving the sets \( C \) and \( X \) to contain only application-specific terms. The detail of each modules in the
Web service clustering using text mining techniques

cascaded word analyser is discussed in the subsequent Sections 3.1, 3.2 and 3.3. As for
the third and fourth feature type, we utilise regular expressions to extract the service
name, sname and the service host address, shost. These steps are implemented as
modules identifyServiceHost and identifyServiceName in Figure 1. As an
example, the service name of this WSDL file http://weather.terrapin.com/
soap/HurricaneService.wsdl is ‘HurricaneService’. As for the fourth type of
feature, we first extract the authority part of the URI, which is the full domain name of
the host containing the WSDL file. In our example above, the authority part is identified
as weather.terrapin.com. To cater for our URI similarity measurement, we further
trim the authority part of the URI by removing all other host labels beginning from
the left, until (but excluding) the second- and top-level domains. The domain name
weather.terrapin.com is reduced to terrapin.com.

3.1 Service content and context extraction

To obtain the web service content, the WSDL files are first tokenised by splitting their
content based on white spaces to produce a set of tokens C. The tokens a ∈ C are
essentially incomplete segments of XML elements, and can appear in various forms
such as incomplete tags <xsd: schema, complete start/end tags <wsdl:types>,
attribute-value pairs elementFormDefault=”qualified” and the text content of
the elements (i.e., descriptions encapsulated by start/end tags). A set of heuristic rules
implemented as regular expressions are utilised to remove non-word tokens. For
example, tokens which exhibit signs of being part of an XML tag are removed. This
process essentially reduces the set C to contain only valid words. As for the second
type of features (i.e., web service context), content of web pages ‘surrounding’ the
WSDL file is utilised as context for describing the corresponding web service. We
first identify the path segment in the hierarchical part of the URI of the WSDL file,
and then recursively crawl the directories along the path. This function is depicted
as the extractPathSegment module in Figure 1. For example, attempts are made
to request for the index web pages from the directories /portal/boulder/
and /portal/ which are part of the URI http://wsrp.bea.com/portal/
boulder/weather.wsdl. All accessible index web pages are extracted and their
content will undergo similar treatment as the content of the WSDL file described earlier,
namely, tokenisation and removal of non-word tokens to create a second set of words X.
This first step of converting XML and HTML contents into tokens, and removing
non-word tokens is depicted as the tokenizeText and removeNonWord modules
in Figure 1. In the second step, the morphological variants of the words in C and
X are conflated through stemming and pattern matching. This step appears as the
conflateWord module in Figure 1. Words are first reduced to their word stem.
For example, using Porter Stemmer (Porter, 1980), the word stem for ‘editing’, ‘editors’
and ‘editorial’ is ‘edit’, ‘editor’ and ‘editori’, respectively. Then, we apply regular
expressions to identify all word stems which are part (i.e., substring) of a longer stem.
Hypothetically, we refer to such group of word stems as word variant cluster. Each
cluster is represented by the shortest unstemmed word of a corresponding cluster
member. In our earlier example, the word stems ‘edit’, ‘editor’ and ‘editori’ are grouped
into the same variant cluster represented by ‘editors’. The different variants of words are
conflated into word variant clusters. Computationally, the elements of C and X are
reduced to only the shortest unstemmed words which represent the hypothetical word variant clusters. The number of variants in each cluster or the cluster size is also an important piece of information that we will utilise later. Words which occur more often, including their different morphological variants, can be considered as more important features. As such, associated with each element $a$ (i.e., normalised word appearing as shortest unstemmed word) in the new conflated sets $C$ and $X$ is the count of the different variants of that same word, $c_a$.

3.2 Content words recognition on the web

After the initial sets $C$ and $X$ of the WSDL files have been obtained, the normalised words in these sets are analysed for their content-bearing property as shown in the module `removeFunctionWord` in Figure 1.

3.2.1 Separating content words from function words

Content words are typically nouns, verbs or adjectives, and are often contrasted with function words which have little or no contribution to the meaning of texts. One of the properties of content words is that they tend to ‘clump’ or to re-occur whenever they have appeared once, according to Manning and Schutze (1999). On the other hand, the occurrence of function words tend to be independent of one another. Very often, Church and Gale (1995) stated that such contrasting property can be captured through the inability of the Poisson distribution to model word occurrences in documents. In other words, unlike content words, function words tend to be Poisson distributed. Following this, one way to decide if a token $a$ is a content word or a function word is by assessing the degree of overestimation of the observed document frequency of the word $a$, denoted by $n_a$ using Poisson distribution. The ratio of the estimated, $\hat{n}_a$, to the observed frequency, $n_a$, of word $a$ is defined as:

$$\Lambda_a = \frac{\hat{n}_a}{n_a}.$$  

A high value of $\Lambda_a$ implies an overestimation which can be used as an indicator of token $a$ being a possible content word. In our case, any word $a$ with $\Lambda_a$ larger than the threshold $\Lambda_T$ is considered as content word where:

$$\Lambda^T = \begin{cases} E[\Lambda] & \text{if } (E[\Lambda] > 1) \\ 1 & \text{otherwise} \end{cases}$$  

and $E[\Lambda]$ is the average of the observed document frequency of all tokens considered. Using Equation (2), we can identify and remove non-content words from the two sets $C$ and $X$. As a result, only content words which are important in describing the associated WSDL files remain in $C$ and $X$. To compute $\Lambda_n$, the main challenge arises when we attempt to obtain the estimated $\hat{n}_a$. Mathematically, $\hat{n}_a$ is given by:

$$\hat{n}_a = N(1 - P(0; \lambda_a))$$  


where $N$ is the size (i.e., number of documents) of the text corpus, and $P(0; \lambda_a)$ is the single-parameter Poisson distribution evaluated at $k = 0$ or in other words, the probability of word $a$ having no occurrence in a document in the corpus. In information retrieval, the single parameter in Poisson, $\lambda_a$ is commonly determined as the average number of occurrences of word $a$ per document (Manning and Schutze, 1999) or $\frac{f_a}{N}$, where $f_a$ is the word frequency of $a$ in the corpus. Since we are dealing with words extracted from online sources (i.e., WSDL files), we need to obtain the corresponding word frequency $f_a$ and the size of the text corpus $N$ required for computing $\hat{n}_a$ from the web too.

According to Kilgarriff and Grefenstette (2003), in corpus linguistics, the entire indexable web (i.e., surface web) can be treated as a large text corpus and search engines such as Google can be employed to obtain the required frequencies and counts. Assuming that the number of web pages on the indexable web is represented by Google’s search index, we can then estimate the value $N$, which is the size of the web corpus in terms of web pages (i.e., documents) using function words as predictors. Function words such as ‘a’, ‘is’ and ‘with’, as opposed to content words, appear with frequencies that are relatively stable over many different genres. However, unlike $N$, to obtain the actual word frequency $f$ on the web is not as straightforward. The pagecounts returned by search engines for each query are in fact the document frequency of the search term. In the case of web corpus, the pagecount for search term $a$ represents $n_a$, and not $f_a$. While $n_a$ is required for determining the overestimation $\Lambda_a$ in Equation (1), we need to obtain $f_a$ beforehand to compute the probability $P(0; \lambda_a)$ in Equation (3). We will briefly discuss our proposed new method for estimating $f_a$ using pagecount on the web in the following Section 3.2.

3.2.2 Estimating word frequency using page count

Conventionally, word frequency $f_a$ and document frequency $n_a$ of word $a$ can be easily determined from local corpora. In local corpora, texts are stored locally and hence, can be manipulated using various database or text search facilities such as regular expressions. However, finding word frequency in the web corpus where one has no direct access to, is less straightforward. In other words, while search engines such as Google are able to provide the number of web pages which contain a certain search term, the total number of times the search term occurred in all those web pages is not accessible. Hereby, we introduce a formula for estimating word frequency using page count. This formula is derived from the K-mixture word distribution by Katz (1996). Using the K-mixture model, the probability of $k$ occurrence of word a in a document is defined as Katz (1996):

$$P(k; \alpha_a, \beta_a) = (1-\alpha_a)\delta_{k,0} + \frac{\alpha_a}{\beta_a+1} \left( \frac{\beta_a}{\beta_a+1} \right)^k.$$  

For the probability of word $a$ occurring exactly once in a document, the general K-mixture model in Equation (4) is reduced to:

$$P(1; \alpha_a, \beta_a) = \frac{\lambda_a}{(\beta_a+1)^2}.$$  

(5)
where $\beta_a = (f_a - n_a) / n_a$, $\lambda_a = f_a / N$, and $\alpha_a = \lambda_a / \beta_a$. As for $N$, $n_a$, and $f_a$, they are the total number of documents in the corpus, the number of documents in the corpus containing word $a$, and the word frequency of $a$ in the corpus, respectively.

For brevity, let $KM_a (1) = P (1; \alpha_a, \beta_a)$. Next, we move on to simplify $KM_a (1)$:

$$KM_a (1) = \frac{(n_a)^2}{N f_a}.$$  \hspace{1cm} (6)

Assuming that the K-mixture model offers a near-perfect fit for the single occurrence (i.e., $k = 1$) of word $a$ in a document, we have that:

$$n_{a,1} \approx KM_a (1) \times N$$  \hspace{1cm} (7)

where $n_{a,1}$ is the observed number of documents in the corpus containing only a single occurrence of $a$. Substituting Equation (6) into Equation (7):

$$n_{a,1} \approx N \frac{(n_a)^2}{N f_a} \approx \frac{(n_a)^2}{f_a}$$  \hspace{1cm} (8)

$$f_a \approx \frac{(n_a)^2}{n_{a,1}}.$$

Let $\hat{f}_a$ be the predicted word frequency for $a$, it follows from Equation (8):

$$\hat{f}_a = \frac{(n_a)^2}{n_{a,1}}.$$  \hspace{1cm} (9)

At the moment, there is no definitive way of obtaining accurate $n_{a,1}$ from the web using any search engines. However, as we will demonstrate later in Section 5 during our experiments, ‘educated guess’ of $n_{a,1}$ within a controlled environment is possible for the purpose of supporting Equation (2) for content words recognition on the web.

### 3.3 Application-specific terms recognition through clustering

Words such as *proxy*, *runtime*, *button*, *valign* and *etc.*, are inevitably present in many WSDL or HTML files, and very often qualify as content words during the analysis of content-bearing property in the previous step. To obtain application-specific terms that potentially describe the functionalities of the web service, here we employ a two-pass clustering algorithm by Wong et al. (2007) known as the *Tree-Traversing Ant (TTA)* to identify application-specific terms. This step is depicted as the last module `removeComputingTerm` in Figure 1. TTA utilises the *Normalised Google Distance (NGD)* by Cilibrasi and Vitanyi (2007) for featureless similarity measurement to partition term clusters during the first pass, and another featureless distance measure called the $n^o$ of *Wikipedia* ($n^o W$) during the second pass for cluster refinement. For the purpose of computation, we consider the structure produced by TTA as a directed acyclic graph, $G$.

The immediate results of the TTA require human interpretation for analysing the word clusters. To facilitate the automatic selection of the desired clusters, which are groups of application-specific terms, we propose a cluster selection algorithm to complement the functioning of TTA. This cluster selection algorithm is based on the iterative propagation
of penalty weights, $\rho$ across the graph, and is inspired by the use of spreading activation algorithm in extending ontologies (Liu et al., 2005). Our selection algorithm requires an oracle of sort, $O$, which is a predefined set of general computing terms. One may question the necessity of clustering and cluster selection, and instead, prefer the use of solely dictionaries for removing general computing terms. However, the use of dictionary raises questions regarding its completeness and maintenance. With the discovery of content word clusters proposed in the previous section, only a few trigger words in a small oracle $O$ is required to ensure the accuracy of the removal using the algorithm discussed in this section.

Figure 2 An example of the output of content word clustering using the tree-traversing ant algorithm with featureless similarities

There are three types of vertices in the graph, namely, sinks vertices (e.g., all the rounded rectangles in Figure 2), interior vertices (e.g., Vertices 2, 3, 4 and 5 in Figure 2) and a source vertex (e.g., Vertex 1 in Figure 2). We begin assigning weights to the sink vertices, and subsequently, to the remaining interior vertices which are the predecessors of the sink vertices. Given that $V_u$ is the set of successor vertices of $u$, each vertex $u$ is assigned a weight $0 \leq \rho_u \leq 1$:

$$
\rho_u = \begin{cases} 
1 & \text{if}(h_u = 0 \land u \in O) \\
0 & \text{if}(h_u = 0 \land u \notin O) \\
|K|^{-1} \sum_{v \in V_u} \rho_v & \text{if}(h_u = 1) \\
e^{-\chi \tau_u} \sum_{v \in V_u} \rho_v & \text{if}(h_u > 1) 
\end{cases} 
$$

(10)

where:

- $|K|$ = the number of sink vertices in $G$
- $h_u$ = the length of the longest path from vertex $u$ to a sink (i.e., height of vertex $u$)
- $\chi$ = a decay constant
- $\tau_u$ = a measure of departure of vertex $u$ from the origins of the weights (i.e., sink vertices).

Lower values of $\chi$ result in slower decay, or in other words, the penalty weights will retain their intensity over longer distances from the sink vertices. $\tau_u$ is only defined for interior vertices $u$ with height $h_u > 1$:
\[
\tau_u = \log_{10} \left( \frac{l}{1 - h_u + 1} \right)
\]

(11)

where \( l \) is the length of the longest directed path in \( G \) (i.e., length of \( G \)). As vertex \( u \) moves further away from the sink vertices, which are the origins of the weights, its height \( h_u \) increases and hence, its \( \tau_u \) increases too. Similar to the exponential decay of electromagnetic radiation with distance into the absorbing medium, the weights assigned to vertices in \( G \) decreases as they are located further away from the sink vertices.

To illustrate, assume that we have the set \( O = \{ \text{webservice} \} \). To begin, all sink vertices \( u \) have \( h_u = 0 \). Hence, using \( O \) and based on Equation (10), all sink vertices \( u \) in Figure 2 are assigned \( \rho_u = 0 \) except \( \text{webservice} \). At the next level, all the interior vertices have \( h_u = 1 \). They are Vertices 3, 4 and 5. Since the sum of the weights of the successors of the interior Vertices 3 and 4 are both 0, their weights too, amount to 0. However, the weight of Vertex 5, \( \rho_5 = 0.1429 \) since \( |K| = 7 \) and one of its successors, namely, \( \text{webservice} \) has a weight of 1. From Figure 2, we can observe that \( l = 3 \). Moving on, the height of the interior Vertex 2 is \( h_2 = 2 \). Substituting \( l \) and \( h_2 \) into Equation (11) produces \( \tau_2 = 0.1761 \). Using the decay constant \( \chi = 0.2 \), the weight of the interior Vertex 2 is \( \rho_2 = 0.1308 \). Using the same calculation with \( h_1 = 3 \), the source Vertex 1 has the weight \( \rho_1 = 0.1031 \). In this paper, we utilise the average penalty weights, \( E[\rho] \) of all the vertices in \( G \), to automatically decide on removing groups of general computing terms in the two sets \( C \) and \( X \). An interior vertex \( u \) with \( \rho_u > E[\rho] \) is deleted and this deletion is propagated to all of its successor vertices. The remaining application-specific terms (i.e., sink vertices) in \( G \) are the new elements of \( C \) and \( X \). More refined methods to determine the threshold for cluster selection are possible but the discussion regarding these methods is beyond the scope of this paper.

4 Features integration for web service clustering

To discover related web services, we perform clustering using the four types of features discussed in Section 3. Similar to content words clustering during features mining, we utilise the tree-traversing ant algorithm by Wong et al. (2007) for clustering the web services. However, instead of using the word-based featureless similarity measurements with \( \text{NGD} \) and \( n^w \text{W} \), we introduce a new semantic relatedness measure based on the combination of the four types of features produced using our techniques described in Section 3. As pointed out before, this combination of features is necessary and critical since we do not have sufficient number of web service files for counting document frequency, and the content of the web service files is inadequate for obtaining word frequency required by vector space or probabilistic relevance models. Such issues are well recognised by Li et al. (2007)’s exploratory study of the WSDL files on the web.

We propose a grand relatedness measure between two web services \( s_i \) and \( s_j \), \( 0 \leq \Phi(s_i, s_j) \leq 1 \) as:
\[
\Phi(s_i, s_j) = 0.4S(C_i, C_j) + 0.3S(X_i, X_j)
+ 0.2\text{sim}(\text{shost}_i, \text{shost}_j) + 0.1\text{sim}(\text{sname}_i, \text{sname}_j).
\]

(12)
The coefficients attached to each similarity function reflect our subjective assignment of significance of the associated features. More refined methods for combining the four features and assigning significance coefficients are possible but such discussion is beyond the scope of this paper. Despite the ad-hoc nature of feature combination, our initial experiments in Section 5 demonstrated acceptable performance of web service clustering using Equation (12). Both $S(C_i, C_j)$ and $S(X_i, X_j)$ are the average group similarities calculated using the formula below:

$$S(A_j, A_i) = \frac{\sum_{a \in A_i} \sum_{b \in A_j} \text{sim}(a, b)}{|A_i||A_j|}$$  \hspace{1cm} (13)

where $\text{sim}(a, b)$ is the featureless similarity computed based on the co-occurrence of words on the web using the Normalised Google Distance (NGD) proposed by Cilibrasi and Vitanyi (2007). $\text{sim}(a, b)$ is obtained through:

$$\text{sim}(a, b) = 1 - \text{NGD}(a, b)$$ \hspace{1cm} (14)

where $\theta = (0,1]$ is traditionally a constant for scaling the distance NGD. However, for the sole purpose of computing $S$ in Equation (13), we modify $\theta$ to become a variable $\theta_{ab}$ by incorporating the word variant count $c$ associated with each conflated content word in the set $C$ or $X$. The more times a word occur, either as itself or as variants in a file, the more significant that word is for describing that file. High word variant count should result in low $\theta_{ab}$ in order to produce high similarity:

$$\theta_{ab} = 0.5 \left( \frac{z_a + z_b}{z_{\text{max}}} \right)$$ \hspace{1cm} (15)

where $z_a = (c_a + q)^{-1} H^{-1}$ and $q$ is a constant for adjusting the magnitude of $z_{ab}$. Considering the fact that $z_{\text{max}}$ is always achieved with the lowest word variant count $c = 1$, the value of $z_{\text{max}}$ does not depend on word variant count but instead, a function of $q$ and $H$. Note that the computation of $z_a$ is based on the discrete probability distribution known as the Zipf-Mandelbrot model (Mandelbrot, 1967). In our evaluations, $q$ is set to 10 to obtain a more linearly distributed $z_{ab}$. $H$ is the harmonic mean defined as:

$$H = \sum_{i=1}^{c_{\text{max}}}(v + q)^{-1}$$ \hspace{1cm} (16)

where $c_{\text{max}}$ is the highest word variant count in the corresponding set of feature (i.e., either $C$ or $X$) across all services in set $S$.

5 Experiments and results

Our experiments were conducted in three phases. Firstly, we assessed the accuracy of word frequency estimation as discussed in Section 3.2 using two local corpora. Secondly, we evaluated the steps of feature mining using 22 real-world WSDL files (September 2007). Thirdly, we extended the initial experiments with 22 new WSDL files (July 2008) for further verification. Here, we will briefly report on the first two phases, a full account is available in a shorter version of this paper at SOCASE2008, see Liu and Wong (2008). We will then provide more details on the extended experiment of the third phase.
5.1 Phase 1: performance of word frequency estimation

To assess the accuracy of the estimated word frequency using the estimator in Equation (9), we conducted an evaluation using two different local corpora. The first is a local general corpus, namely, the British National Corpus (BNC) (Leech et al., 2001), and the second is a local domain corpus, namely, the GENIA Corpus (Kim et al., 2003) for ‘molecular biology’. Approximately 1400 words are selected from each of the three groups of words based on their appearance in the respective local corpora: frequent words, moderately-used words, and rare words. Following this, we obtained two sets of words for evaluation, namely, a set of generally-occurring words from BNC, and a set of domain-specific terms from the GENIA corpus. Using the two sets of words and the two local corpora, we examine the accuracy of the estimation from two perspectives. Firstly, we assess the residual or error of the estimation by looking at the difference between the observed frequency $f$ and the estimate frequency $\hat{f}$. Secondly, we examine the ratio $R = \frac{\hat{f}}{f}$ of the estimated frequencies $\hat{f}$ to the observed frequency $f$. Table 1 summarises the results.

Table 1 Results of evaluating the accuracy of word frequency estimation using the GENIA domain corpus, and the BNC general corpus

<table>
<thead>
<tr>
<th>Types of statistics</th>
<th>Variables</th>
<th>GENIA domain corpus</th>
<th>BNC general corpus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics on text corpus</td>
<td>Corpus size, $N$</td>
<td>2000</td>
<td>4124</td>
</tr>
<tr>
<td></td>
<td>Total word count, $F$</td>
<td>402 483</td>
<td>100 106 029</td>
</tr>
<tr>
<td></td>
<td>Sample size</td>
<td>4319</td>
<td>4572</td>
</tr>
<tr>
<td>Statistics on residuals of estimation</td>
<td>SSR</td>
<td>1 193 439</td>
<td>306 774 565</td>
</tr>
<tr>
<td></td>
<td>MSR</td>
<td>276.2590</td>
<td>67 083.8760</td>
</tr>
<tr>
<td></td>
<td>RMSR</td>
<td>16.6210</td>
<td>259.0056</td>
</tr>
<tr>
<td>Statistics on ratio of estimated to observed frequency, $R$</td>
<td>$E[R]$</td>
<td>1.0111</td>
<td>0.9741</td>
</tr>
<tr>
<td></td>
<td>$SD[R]$</td>
<td>0.0847</td>
<td>0.0766</td>
</tr>
</tbody>
</table>

Table 1 provides empirical evidence demonstrating the good approximation of the actual observed frequency $f$ by the estimated frequency $\hat{f}$. This is evident through the relatively low RMSR, the average ratio of estimated to observed frequency $E[R]$ which is close to 1, and the extremely low deviation $SD[R]$ of the ratios from the mean. For example, in the case of the set of words from the GENIA corpus, the low RMSR at 16.6210 implies that the estimated frequencies, on average, miss the actual observed frequency by 16 word count. Similarly, the estimation of word frequency using the BNC achieves the same accuracy. Relative to $F$, the average residual RMSR in BNC is slightly higher than what is achieved by the GENIA corpus. All in all, the consistency of the accurate approximation across both the GENIA corpus and the BNC shows that the estimator is applicable to both general and domain-specific language.

5.2 Phase 2: results for initial web service cluster discovery

Since there are no benchmarks nor readily available datasets for clustering web service files, we have resorted to manually selecting files from the top 420 query results returned by a Google search filetype:wsdl. Many returned results are erroneous, some are
normal HTML files but happen to use .wsdl as the file extension. Since automatic processing does not guarantee a reliable set of test data, we manually constructed a small test set of 22 WSDL files as summarised in Table 2. To demonstrate the performance of the various aspects of features mining discussed in Section 3, we will use the outputs related to the WSDL file at the following URI http://studentmasjid.com/Quran/QuranService.wsdl for discussion. This service offers access to the verses and content of Islamic scriptures. The service name for this WSDL file is QuranService while its host is studentmasjid.com.

Table 2  The manually categorised dataset obtained from the web for our experiments

<table>
<thead>
<tr>
<th>WSDL File URI</th>
<th>Service Name</th>
<th>Service Host</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://studentmasjid.com/Quran/QuranService.wsdl">http://studentmasjid.com/Quran/QuranService.wsdl</a></td>
<td>QuranService</td>
<td>studentmasjid.com</td>
<td>scripture</td>
</tr>
<tr>
<td><a href="http://www.scriptoriumchurchide.org/wsdl/Bible.wsdl">http://www.scriptoriumchurchide.org/wsdl/Bible.wsdl</a></td>
<td>Bible</td>
<td>sgc-text教堂.org</td>
<td>scripture</td>
</tr>
<tr>
<td><a href="http://developer.ebay.com/_ShoppingService.wsdl">http://developer.ebay.com/_ShoppingService.wsdl</a></td>
<td>ShoppingService</td>
<td>ebay.com</td>
<td>retail</td>
</tr>
<tr>
<td><a href="http://soap.amazon.com/schemas/AmazonWebService.wsdl">http://soap.amazon.com/schemas/AmazonWebService.wsdl</a></td>
<td>AmazonWebService</td>
<td>amazon.com</td>
<td>retail</td>
</tr>
<tr>
<td><a href="http://weather.gov/forecasts-wsdl/ndfdXML.wsd1">http://weather.gov/forecasts-wsdl/ndfdXML.wsd1</a></td>
<td>ndfdXML</td>
<td>weather.gov</td>
<td>weather</td>
</tr>
<tr>
<td><a href="http://weather.terrapin.com.soap/HurricaneService.wsdl">http://weather.terrapin.com.soap/HurricaneService.wsdl</a></td>
<td>HurricaneService</td>
<td>terrapin.com</td>
<td>weather</td>
</tr>
<tr>
<td><a href="http://soap.be.com/portal">http://soap.be.com/portal</a> burel</td>
<td>weather.wsdl</td>
<td>bureau.be</td>
<td>weather</td>
</tr>
<tr>
<td><a href="http://hodes.cs.uga.edu/projects/WeatherFetcher.wsdl">http://hodes.cs.uga.edu/projects/WeatherFetcher.wsdl</a></td>
<td>WeatherFetcher</td>
<td>uga.edu</td>
<td>weather</td>
</tr>
<tr>
<td><a href="http://www.cs.uni-paderborn.de/~GlobalWeather.wsdl">http://www.cs.uni-paderborn.de/~GlobalWeather.wsdl</a></td>
<td>GlobalWeather</td>
<td>uni-paderborn.de</td>
<td>weather</td>
</tr>
<tr>
<td><a href="http://www.zachhaus.org/dso/trackers/WeatherForecast.wsdl">http://www.zachhaus.org/dso/trackers/WeatherForecast.wsdl</a></td>
<td>WeatherForecast</td>
<td>codemhaus.org</td>
<td>weather</td>
</tr>
<tr>
<td><a href="http://services.bhp.com/weather/weather.wsdl">http://services.bhp.com/weather/weather.wsdl</a></td>
<td>weather</td>
<td>bhp.com</td>
<td>weather</td>
</tr>
<tr>
<td><a href="http://services.bhp.com/weather/weather.wsdl">http://services.bhp.com/weather/weather.wsdl</a></td>
<td>weather</td>
<td>bhp.com</td>
<td>weather</td>
</tr>
<tr>
<td><a href="http://genome.d%EF%BF%BD.heidelize.de/meno">http://genome.d�.heidelize.de/meno</a> ...trofet.wsdl</td>
<td>trefte</td>
<td>d�.heidelize.de</td>
<td>bioinformatics</td>
</tr>
<tr>
<td><a href="http://genome.d%EF%BF%BD.heidelize.de/meno">http://genome.d�.heidelize.de/meno</a> ...extractRef.wsdl</td>
<td>extractRef</td>
<td>d�.heidelize.de</td>
<td>bioinformatics</td>
</tr>
<tr>
<td><a href="http://www.ncbi.nlm.nih.gov/entrez/wsd">http://www.ncbi.nlm.nih.gov/entrez/wsd</a> ...Annotation_2.wsd1</td>
<td>Annotation_2</td>
<td>ncbi.nih.gov</td>
<td>annotation</td>
</tr>
</tbody>
</table>

Notes: There are four categories manually identified from the 22 WSDL files, namely, ‘scripture’, ‘retail’, ‘weather’ and ‘bioinformatics’. These categories are used to assess the result of web service cluster discovery using the approach described in this paper. Some of the URIs and service names have been truncated with ‘…’ due to space constraints.

Figure 3 shows the results of clustering the content words in set $X$ of service QuranService to identify naturally-occurring groups of words based on their genres using TTA. The two biggest clusters, represented by their Centroids 7 and 8, are content words describing ‘general computing’ and ‘Islamic studies’, respectively. More generally, we can see that terms which are closely related to ‘general computing’ are mainly successors of the Vertex 27. Table 4 shows part of the results during content word cluster selection to identify application-specific terms. For this purpose, we rely only on a small oracle containing the words $O = \{\text{runtime, webservice, developer, module, data}\}$. Using $\chi = 0.1$, the penalty weight of Vertex 27 is $\rho_{27} = 0.1246$. Since $\rho_{27} > E[\rho]$, Vertex 27 is removed and the deletion is propagated to all successors. As a result, the final feature set $X$ (i.e., web service context) is $\{\text{tafseer, sunnah, recitation, ramadhan, quran, prophet, masjid, hadith, islamic}\}$. 
Figure 3  The result of content word clustering on the web service context set X from the service QuranService using the tree-traversing ant algorithm based on featureless similarities.

Notes: Visually, one can easily identify the naturally occurring content word clusters. The two biggest clusters, represented by their Centroids 7 and 8, are content words describing ‘general computing’ and ‘Islamic studies’.

5.3 Phase 3: extended dataset

To show the effectiveness and the repeatability of the experiments, we revised the initial experiment by including an additional 22 WSDL files. Because of the time lagging between two experiments (from September 2007 to July 2008), some web services need to be updated and some become unavailable. As a result, we updated the WSDL files for two web services, and removed references to four unavailable WSDL files. In total, we have 40 WSDL files for this extended experiment.

These web service files are classified into six categories beforehand to assist in the final inspection of the result. Two new categories, namely, ‘calculator’ and ‘web search’ were added to the existing four used in the first experiment. We changed the category ‘retail’ to ‘payment’, and retained the other three which are ‘weather’, ‘bioinformatics’ and ‘religious scripture’. Table 3 summarises the revised list of WSDL files used in this experiment.

During the course of this extended experiment, we realised that a number of new WSDL files do not contain descriptive text in the form of documentation or comments. For instance, the service file HurricaneService.wsdl hosted on http://weather.terrapin.com/ contains only non-descriptive elements in the
form XML tags with attributes. On the other hand, the service file `ndfdXML.wsdl` in http://www.weather.gov/ make use of the `<documentation>` element to provide descriptions about the service and certain functions. Services that do not have descriptive text in their WSDL files will have inadequate words in the feature set C to support further analysis. To increase the number of words for describing the service file, we included a step in the `tokeniseText` module in our cascaded word analyser to identify potential content words using the values of the attribute name. For example, in the file `HurricaneService.wsdl`, there is an abundance of tags such as `complexType` and `element` with attributes in the form of `name="StormDataResult", name="getStormForecastPositionResponse", name="pressure", name="wind"` and so on. We utilise regular expressions to extract potential content words from these attributes based on the difference in letter casing. To illustrate, the application of the regular expression `/([A-Z][a-z]+)\+` on this string `name="getStormForecastPositionResponse"` produces 'Storm', 'Forecast', 'Position' and 'Response' as potential content words.

Table 3  Dataset Used for this extended experiment

<table>
<thead>
<tr>
<th>WSDL File URI</th>
<th>Service Name</th>
<th>Service Host</th>
<th>Category</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://ws.apache.org/axis/docs/axis2/Axis2wsdl">http://ws.apache.org/axis/docs/axis2/Axis2wsdl</a></td>
<td>Axis2</td>
<td>ws.apache.org</td>
<td>axis2</td>
<td></td>
</tr>
<tr>
<td><a href="http://ws.apache.org/axis/docs/axis2/Soap12">http://ws.apache.org/axis/docs/axis2/Soap12</a> wsdl</td>
<td>Soap12</td>
<td>ws.apache.org</td>
<td>soap12</td>
<td></td>
</tr>
<tr>
<td><a href="http://ws.apache.org/axis/docs/axis2/Soap12">http://ws.apache.org/axis/docs/axis2/Soap12</a> wsdl</td>
<td>Soap12</td>
<td>ws.apache.org</td>
<td>soap12</td>
<td></td>
</tr>
<tr>
<td><a href="http://ws.apache.org/axis/docs/axis2/Soap12">http://ws.apache.org/axis/docs/axis2/Soap12</a> wsdl</td>
<td>Soap12</td>
<td>ws.apache.org</td>
<td>soap12</td>
<td></td>
</tr>
<tr>
<td><a href="http://ws.apache.org/axis/docs/axis2/Soap12">http://ws.apache.org/axis/docs/axis2/Soap12</a> wsdl</td>
<td>Soap12</td>
<td>ws.apache.org</td>
<td>soap12</td>
<td></td>
</tr>
<tr>
<td><a href="http://ws.apache.org/axis/docs/axis2/Soap12">http://ws.apache.org/axis/docs/axis2/Soap12</a> wsdl</td>
<td>Soap12</td>
<td>ws.apache.org</td>
<td>soap12</td>
<td></td>
</tr>
<tr>
<td><a href="http://ws.apache.org/axis/docs/axis2/Soap12">http://ws.apache.org/axis/docs/axis2/Soap12</a> wsdl</td>
<td>Soap12</td>
<td>ws.apache.org</td>
<td>soap12</td>
<td></td>
</tr>
<tr>
<td><a href="http://ws.apache.org/axis/docs/axis2/Soap12">http://ws.apache.org/axis/docs/axis2/Soap12</a> wsdl</td>
<td>Soap12</td>
<td>ws.apache.org</td>
<td>soap12</td>
<td></td>
</tr>
<tr>
<td><a href="http://ws.apache.org/axis/docs/axis2/Soap12">http://ws.apache.org/axis/docs/axis2/Soap12</a> wsdl</td>
<td>Soap12</td>
<td>ws.apache.org</td>
<td>soap12</td>
<td></td>
</tr>
<tr>
<td><a href="http://ws.apache.org/axis/docs/axis2/Soap12">http://ws.apache.org/axis/docs/axis2/Soap12</a> wsdl</td>
<td>Soap12</td>
<td>ws.apache.org</td>
<td>soap12</td>
<td></td>
</tr>
<tr>
<td><a href="http://ws.apache.org/axis/docs/axis2/Soap12">http://ws.apache.org/axis/docs/axis2/Soap12</a> wsdl</td>
<td>Soap12</td>
<td>ws.apache.org</td>
<td>soap12</td>
<td></td>
</tr>
<tr>
<td><a href="http://ws.apache.org/axis/docs/axis2/Soap12">http://ws.apache.org/axis/docs/axis2/Soap12</a> wsdl</td>
<td>Soap12</td>
<td>ws.apache.org</td>
<td>soap12</td>
<td></td>
</tr>
<tr>
<td><a href="http://ws.apache.org/axis/docs/axis2/Soap12">http://ws.apache.org/axis/docs/axis2/Soap12</a> wsdl</td>
<td>Soap12</td>
<td>ws.apache.org</td>
<td>soap12</td>
<td></td>
</tr>
<tr>
<td><a href="http://ws.apache.org/axis/docs/axis2/Soap12">http://ws.apache.org/axis/docs/axis2/Soap12</a> wsdl</td>
<td>Soap12</td>
<td>ws.apache.org</td>
<td>soap12</td>
<td></td>
</tr>
<tr>
<td><a href="http://ws.apache.org/axis/docs/axis2/Soap12">http://ws.apache.org/axis/docs/axis2/Soap12</a> wsdl</td>
<td>Soap12</td>
<td>ws.apache.org</td>
<td>soap12</td>
<td></td>
</tr>
<tr>
<td><a href="http://ws.apache.org/axis/docs/axis2/Soap12">http://ws.apache.org/axis/docs/axis2/Soap12</a> wsdl</td>
<td>Soap12</td>
<td>ws.apache.org</td>
<td>soap12</td>
<td></td>
</tr>
<tr>
<td><a href="http://ws.apache.org/axis/docs/axis2/Soap12">http://ws.apache.org/axis/docs/axis2/Soap12</a> wsdl</td>
<td>Soap12</td>
<td>ws.apache.org</td>
<td>soap12</td>
<td></td>
</tr>
<tr>
<td><a href="http://ws.apache.org/axis/docs/axis2/Soap12">http://ws.apache.org/axis/docs/axis2/Soap12</a> wsdl</td>
<td>Soap12</td>
<td>ws.apache.org</td>
<td>soap12</td>
<td></td>
</tr>
</tbody>
</table>
The use of attribute values to increase the number of words for content word recognition (i.e., the removeFunctionWord module) has inevitably contributed to more general computing terms related to the web such as response, result, search, query, index, get, post and url. We simply enumerated such terms from several sample HTML and XML files, and increased the size of our oracle \(O\). This increase in computing terms propagates to the removeComputingTerm module which involves term clustering. If one recalls, the output of term clustering using \(TTA\) is in the form of a binary tree. As such, a higher number of words involved during clustering contributes to potentially more clusters which leads to greater tree depth and breadth. As a result, we increased the decaying rate of the penalty weights (i.e., faster dissipation) during cluster selection to prevent the over pruning of the binary tree of clusters produced by \(TTA\). This was accomplished by increasing the decay constant \(\chi\) from 0.1 in the first experiment to 0.7.

Figure 4 shows the result of web service clustering automatically performed by \(TTA\) using the four types of features discussed in Section 3. The clusters shown in this figure illustrates six groups of WSDL files which have been objectively determined as providing similar services. The ‘weather’ cluster of web services is represented by Vertex 28. The remaining ‘bioinformatics’, ‘religious scriptures’, ‘payment’, ‘calculator’ and ‘web search’ clusters are represented by Vertices 33, 4, 30, 5 and 10, respectively. The clusters produced by \(TTA\) in this extended experiment accurately reflects the natural grouping of the web services except in two cases. In these two cases, two calculator services, namely, parasoft.com/calculator and roesweb.nl/calc were misplaced into other categories. parasoft.com/calculator was grouped into
the ‘payment’ cluster, while roesweb.nl/calc was placed in the Sub-cluster 25 in the ‘weather’ category instead. Upon inspection, we realised that many of the WSDL files offering calculator services were very small in size and do not have adequate content words to describe them. Similar to the previous experiments, the sub-clusters within each main cluster demonstrate certain pattern of service groupings. The most obvious example is the ‘bioinformatics’ cluster where the WSDL files were further grouped into sub-clusters according to their host. For instance, the sub-cluster represented by Vertex 40 contains bioinformatics services hosted by dkfz-heidelberg.de, while the services hosted by niq.ac.jp and dtu.dk were grouped into Sub-clusters 38 and 39, respectively. Sub-clusters 20 and 24 in the ‘weather’ category are similar examples. In the case of Sub-cluster 18 in the ‘weather’ category, the web services were grouped based on the top level domain .gov.

6 Conclusion and future work

Clustering web services into functional similar groups can greatly reduce the search space of a service discovery task. Therefore, it can be seen as a predecessor of web service discovery or an important functionality provided by future service search engines. However, very few research has looked into this area. This paper proposed mining different types of features of a web service and use these features for web service clustering. To realise the proposed techniques, difficult issues such as differentiating content words from function words, and obtaining word frequencies from the web are resolved. A spreading activation based automatic cluster selection algorithm is also implemented.

In this paper, we have demonstrated the feasibility of discovering functionally-related web services through the mining and the clustering of four types of features. Such an approach answers the plea of automatically adding meaningful description or comments to poorly described web services. It also provides a good starting point for the development of practical service search engines. The web service clusters discovered using our approach provide systems with access to redundant services in the case of a failure. Moreover, options for service redundancy do not only exist at the functional level, but also at the physical host level. In this regard, the service consumers can simply opt for functionally-similar services regardless of hosts, or can be selective in terms of the providers. In particular, it can serve as a modular step that can be integrated with architectures for semantic web service discovery and automatic SOAP message understanding to further extract meanings from web services.

The contributions of this paper extend beyond the web service community where service discovery and redundancy are important issues. The proposed approach and the output which follows are potentially useful to the text mining research community for discovering emergent semantics in a service-oriented environment.

More work is planned to refine the process of features mining, especially the various modules in the cascaded word analyser. The current results are only based on datasets from Google, a larger repository of services is made available through WSCE by Al-Masri and Mahmoud (2007), therefore we plan to extend this work so that it complements WSCE through web service clustering. In addition, the current measurement of the grand relatedness was derived in an ad hoc manner, and reformulation may be necessary in search of mathematical justifications.
Acknowledgement

This work is partially supported by a University of Western Australia Research Grant 2008. Author Wong is grateful of the Australian Commonwealth Government’s Endeavour International Postgraduate Research Scholarships (EIPRS) and the University of Western Australia’s Postgraduate Award for International Students. We also would like to thank the anonymous reviewers for the constructive comments.

References


Web service clustering using text mining techniques


Note

1 http://uddi.microsoft.com/about/FAQshutdown.htm