Latent Semantic Analysis – The Dynamics of Semantics Web Services Discovery

Chen Wu, Vidyasagar Potdar, and Elizabeth Chang

Digital Ecosystems and Business Intelligence Institute
Curtin University of Technology, Perth 6845, WA, Australia
{Chen.Wu,Vidyasagar.Potdar,Elizabeth.Chang}@cbs.curtin.edu.au
http://debii.curtin.edu.au

Abstract. Semantic Web Services (SWS) have currently drawn much momentum in both academia and industry. Most of the solutions and specifications for SWS rely on ontology building, a task needs much human (e.g. domain experts) involvement, and hence cannot scale very well in face of vast amount of web information and myriad of services providers. The recent proliferation of SOA applications exacerbates this issue by allowing loosely-coupled services to dynamically collaborate with each other, each of which might maintain a different set of ontology. This chapter presents the fundamental mechanism of Latent Semantic Analysis (LSA), an extended vector space model for Information Retrieval (IR), and its application in semantic web services discovery, selection, and aggregation for digital ecosystems. First, we explore the nature of current semantic web services within the principle of ubiquity and simplicity. This is followed by a succinct literature overview of current approaches for semantic services/software component (e.g. ontology-based OWL-s) discovery and the motivation for introducing LSA into the user-driven scenarios for service discovery and aggregation. We then direct the readers to the mathematical foundation of LSA – SVD of data matrices for calculating statistics distribution and thus capturing the 'hidden' semantics of web services concepts. Some existing applications of LSA in various research fields are briefly presented, which gives rise to the analysis of the uniqueness (i.e. strength, limitations, parameter settings) of LSA application in semantic web services. We provide a conceptual level solution with a proof-of-concept prototype to address such uniqueness. Finally we propose an LSA-enabled semantic web services architecture fostering service discovery, selection, and aggregation in a digital ecosystem.

1 Introduction

Semantics play an important role in the complete lifecycle of Web services as it is able to help service development, improve service reuse and discovery, significantly facilitate composition of Web services and enable integration of legacy applications as part of automatic business process integration. Unfortunately, current Web Service Description Language (WSDL) standard operates at the syntactic level and lacks the semantic expressivity needed to represent the requirements and capabilities of Web

T.S. Dillon et al. (Eds.): Advances in Web Semantics I, LNCS 4891, pp. 346–373, 2008.

Services. This gap has motivated a lot existing research effort towards the Semantic Web Services (SWS). The fundamental idea underlying current SWS community is that in order to achieve machine-to-machine integration, a markup language must be descriptive enough that a computer can automatically determine its meaning. Following this principle, many semantic annotation markup languages have thus come into view, among them are OWL-S (formerly known as DAML-S) and WSDL-S that have gained great momentum in recent years. The main goal of both OWL-S and WSDL-S is to establish a framework within which service descriptions are made and shared.

The premise of such an ontology-based markup language approach is that every SWS user (be it normal website or end customer) is able to employ a standard ontology, consisting of a set of basic classes and properties, for declaring and describing services. One concern about this descriptive annotation-driven approach is its feasibility: since it would be much more time-consuming to create and publish ontology-annotated (WSDL) content as they would need to be done by domain human experts and powerful editing tools for common users. Other problems might occur when different groups of users and communities want to manage the shared ontology. With this being the case, it would be much less likely for industry companies to adopt these practices as it would only slow down their progress.

In this chapter we carry out SWS research, in particular the service discovery, from another empirical perspective. We believe that one thing distinguishes Web (services) semantics from other forms of semantics is its 'user-centred' commitment towards ubiquity and simplicity, the two most renowned factors leading to the great success of today's Web. By ubiquity, we mean that the underlying technology (such as HTTP and TCP/IP) has to be very robust, lightweight, and non-human intervened to serve for various applications and users. By simplicity, we mean millions of end users can easily access to and personally use the technology without too much expertise in both domain and IT areas. This idea drives us to come up with a novel method to approach the SWS using Latent Semantic Analysis (LSA) technique – the main theme of this chapter. Nevertheless, proposing alternate approach does not mean we completely go against ontology-based approach. On the contrary, we acknowledge that adding semantics in the form of ontology to represent the requirements and capabilities of Web services is essential for achieving unambiguity and machine-interpretability for web services, and hence becomes our long term research objectives. For example, we are currently seeking effective ways that can convert some of our research result – the higher-order association (the very initial stage semantic space) in this chapter – into lightweight ontology in a semi-automatic manner.

The chapter is organised as follows. Section 2 presents OWL-S and WSDL-S, the two widely accepted SWS specifications and characteristics. Section 3 provides in-depth mathematics technique and working mechanism on LSA. Applications of LSA in IR, cognitive, and psychology are introduced in Section 4. This is followed by the rationale to apply LSA in the area of SWS, which has some issues to address. Section 5 proposes the conceptual model of LSA-based SWS. Section 6 then presents semantic search engine prototype based on our conceptual model. Experiment results are reviewed in Section 7, where three LSA and one WSDL parameters are manipulated to gain further understanding of our approach. The chapter concludes in Section 8.

2 Related Work

In this chapter we provide a succinct survey on existing SWS approaches. In particular, the OWL-s and WSDL-s specifications are examined from an empirical perspective.

Semantic Web Services (SWS) attempts to address the problem associated with automatic service discovery, dynamic service selection, composition and aggregation, and other relevant tasks involved in implementing web services. In this section we provide a succinct survey on existing SWS approaches. In particular, the OWL based Web service ontology (OWL-S), Web services modelling ontology (WSMO), and WSDL-S specifications are examined from an empirical perspective. Of these three approaches OWL-S and WSMO can be categorized together because they propose a model for developing WS Ontologies, on the other hand WSDL-S utilizes the existing web service description and attempts to add a layer of semantics to it [28, 29].

From the first category i.e. OWL-S and WSMO, OWL-S is more successful compared to WSMO although both of these approaches define their own set of detailed semantic models, which need to be used when expressing the semantic meaning to web services. On the other hand WSDL-S incorporates the existing WSDL file and adds semantics to it, thus reducing a reasonable amount of time to reconstruct the whole WSDL using a model similar to OWL-S or WSMO. Thus it takes more of a bottom up approach when adding semantics to web services. We now explain each of these approaches in more detail [28].

2.1 OWL Based Web Service Ontology

The W3C standard OWL is based on RDF(S) and supports the modelling of knowledge/Ontologies. Within the semantic web services framework, OWL-S provides a set of markup language constructs that can be used to unambiguously describe the properties and functionalities of web service, which in turn adds machine understanding capabilities to web services [29]. This machine understandability can facilitate dynamic service discovery, selection, composition and aggregation. OWL-S describes each and every instance of web service using an OWL ontology which comprises of service profile, service model and service grounding. *Service profile* describes what the service does, *service model* explains how the service works or how it can be used and finally *service grounding* details how to interact with this service. These three components within OWL-S provide the backbone for adding machine intelligence to web services. There are several editors which can be used to create OWL-S. Protégé provides an OWL-S extension, which can be used for creating semantic service descriptions [http://owlseditor.semwebcentral.org/documents/tutorial.pdf].

2.2 Web Services Modelling Ontology

WSMO has four top-level elements, which needs to be described in order to express Semantic Web services [28, 30]. These include:

- Ontologies, which provide the terminology to capture the relevant aspects of the domain.
- Web services, which describe the computational entity providing access to services.

- Goals, which represent user's desires, which can be fulfilled by discovery and executing a Web service and finally
- Mediators, which describe elements that overcome interoperability problems between different WSMO elements. WSMO handled mediation at three levels – data level, protocol level and process level, which is essential for interoperability.

The WSMO elements referred above includes Concepts, Relations, Functions, Instances and Axioms. *Concepts* refer to the subsumption hierarchy and their attributes, including range specification e.g. "person" with attributes like "name, family, DoB etc". *Relations* describe interdependencies between a set of parameters. *Functions* are a special type of relations that have an unary range beside the set of parameters. Instances define the concepts explicitly by specifying concrete values for attributes and *Axioms* are specified as logic expressions and help to formalize domain specific knowledge.

2.3 Web Service Description Language - Semantics

WSDL describes web services so that they can be interoperable however it lacks the semantic information which is required to facilitate machine understanding. OWL as described earlier supports developing of ontologies in a powerful way, but it lacks the technical details required to express web services. WSDL-S incorporates these advantages by adopting semantic annotation to web services using OWL. In other words WSDL-S connects WSDL and OWL in a very practical manner [27, 28]. It is much better than OWL-S and WSMO because of the following advantages:

- It is compatible to WSDL as semantic information is added to the WSDL file itself by relating it to an external domain model.
- There is a option to choose any modeling technology like OWL or UML
- It is very simple to apply and is based on a stabile standard, which is important for practical use

WSDL-S is the latest submission to W3C. It extends the functionality of WSDL by defining new elements and annotations for already existing elements. This annotation provides a mechanism for adding semantics by linking it to semantic concepts defined in some external domain model or ontology. Semantics can be associated with inputs, outputs, operations, preconditions etc. The main advantage with this approach is that the semantic models are not necessarily being tied with OWL; any existing semantic models can be reused as long as they can be referenced from within the WSDL. This is a huge advantage because there are already a lot of semantic model developed over time and this approach can adopt the same rather than replicating from scratch. The following extensibility elements and attributes can be used in WSDL-S to add semantics to web services. These include

- modelReference – Element: Input and Output Message Types:
    - This extending attribute supports the connections between a WSDL document and an ontology
- schemaMapping - Element: Input and Output Message Types
    - This extending attribute solves differences in schemas by XSL transformation

- modelReference - Element: Operation
  - Captures the semantics of the functional capabilities of an operation, which can be used for dynamic web service composition or aggregation.
- pre-conditions – Parent Element: Operation
  - This extending attribute defines the pre-conditions that need to be satisfied before the operation can be invoked.
- effects – Parent Element: Operation
  - This extending attribute defines the set of semantic statements that must be true after the operation is executed.
- category - Parent Element: Operation
  - This element is used in the classification of the service interface. It is basically used for semantic lookup in a service registry.

The flexibility offered by WSDL-S in incorporating existing semantic domain models shows that this approach is going to be very successful as well as practical. Creating semantic markup of web services is a realization of true semantic web and can only be possible by a mix of these technologies. This concludes the section on semantic web services and we now discuss focus on web services discovery using LSA.

To our best knowledge, [1] is the only work to date that has attempted to leverage LSA in facilitating web services discovery. However, both implementation and the experiment result is not very thoroughly stated and analysed.

3 Background

3.1 LSA and VSM

Latent Semantic Analysis (LSA), also known as Latent Semantic Indexing, is a technique for document retrieval based on text vector representation. It evolves from the Vector Space Model (VSM), a widely used IR model. In VSM, each document is encoded as a vector, where each vector component reflects the importance (e.g. frequency or *tf* * *idf*) of a particular term in representing the meaning of that document. The vectors for all documents in a collection are stored as the 'columns' of a single matrix – the term-by-document matrix $\boldsymbol{A}$, in which each document is represented as a vector with a set of dimensions, each of which measures the importance of a keyword in that particular document. On the other hand, each term is considered as a vector, each component of this vector measures the weight of this term in the document. A user query is then converted to a 'pseudo-document' that can be compared with all existing documents through basic vector operations, such as the cosine value of two vectors (vector dot product divided by the norm product). The cosine value is referred to as the 'similarity' between user query and each document, and is used for ranking the document relevance against the user query.

The main problem of VSM is its reliance on keyword-based matching that has several well-known limitations such as low recall, issues caused by synonymy and polysemy. This is due to the assumption in VSM that keyword can precisely represent the concepts inferred in the human natural langue. As stated in [2], "users want to retrieve on the basis of conceptual content, and individual words provide unreliable

evidence about the conceptual topic or meaning of a document. There are usually many ways to express a given concept, so the literal terms in a user's query may not match those of a relevant document. In addition, most words have multiple meanings, so terms in a user's query will literally match terms in documents that are not of interest to the user." To solve these limitations, LSA is firstly proposed in [2] that supports a different idea – "a particular concept is stated not just through a unique word, but with a distribution of terms focused around that concept". This different approach – indexing and retrieving based on matching concept distribution rather than a particular term/keyword – should firstly reduce the consequence of synonyms as the searching is not based on a single term that may have many synonyms, but a distribution of terms around that concept. It should also reduce the influence of polysemy as the distribution of terms will differ for each unique concept, on which the indexing and searching are based. Evidently, the main challenge in LSA is to convert keyword-based term space into a concept-centred semantic space. While a thorough discussion on mathematics detail of LSA is beyond the scope of this chapter, a brief introduction of mathematics aspects, especially the matrices techniques, is presented as follows.

3.2 Mathematical Foundation

LSA is a variant of the VSM in which the original VSM matrix is replaced by a low-rank approximation matrix. Therefore, the original vector space is reduced to a 'sub-space' as close as possible to the original matrix in order to: (1) Remove extraneous information or noise from the original space, (2) Extract and infer more important and reliable relations of expected contextual usage of terms. As a result, LSA overcomes traditional VSM problems by building 'hidden' meaningful concepts from existing keywords. This is achieved by finding the higher-order association between different keywords using pure mathematical model: a two-mode factor analysis, i.e. the Singular Value Decomposition (SVD).

Formally, given a rectangular m-by-n matrix $\boldsymbol{A}$ ($m \geq n$), and rank ($\boldsymbol{A}$) = $\boldsymbol{r}$ the singular value decomposition of $\boldsymbol{A}$, denoted by **SVD**($\boldsymbol{A}$), is any factorisation of the form:

$$\boldsymbol{A} = \boldsymbol{T}_0 \, \boldsymbol{S}_0 \, \boldsymbol{D}_0^{\,t},$$

where $\boldsymbol{T}_0$ is an m-by-m orthogonal matrix ($\boldsymbol{T}_0^{\,t} \, \boldsymbol{T}_0 = \boldsymbol{I}_n$) having the left singular vectors of $\boldsymbol{A}$ as its columns, $\boldsymbol{D}_0$ is an n-by-n orthogonal matrix ($\boldsymbol{D}_0^{\,t} \, \boldsymbol{D}_0 = \boldsymbol{I}_n$) having the right singular vectors of $\boldsymbol{A}$ as its columns, and $\boldsymbol{S}_0$ is an m-by-n diagonal matrix $\boldsymbol{S}_0$ = diag(σ_1, σ_2, $\cdots$, σ_n), $\sigma_i > 0$ for $1 \leq i \leq r$, $\sigma_j = 0$ for $j \geq r + 1$, and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_n = 0$ of $\boldsymbol{S}_0$ in decreasing order along its diagonal. The white area in $\boldsymbol{S}_0$ stands for zeroes. That is to say the rank r of the matrix $\boldsymbol{A}$ is equal to the number of non-zero singular values. This factorisation exists for any matrices. There exists dedicated research on calculating SVD in the literature such as [3]. Here we focus on how and why SVD can be applied in the application of Information Retrieval (IR). Figure 1 depicts the application of LSA in IR, where the element $\mathbf{a}_{ij}$ in term-document matrix $\boldsymbol{A}$ denotes the frequency in which term **i** occurs in document **j**. In practice, this frequency value is replaced with local and global weights to adjust the importance of terms within or amongst documents. Based on the two-mode factor analysis theory, the columns in $\boldsymbol{T}_0$ represent term vectors (i.e. the 'entity' set) and columns in $\boldsymbol{D}_0$ (or rows in $\boldsymbol{D}_0^{\,t}$) represent document vectors (i.e. the 'attribute' set).

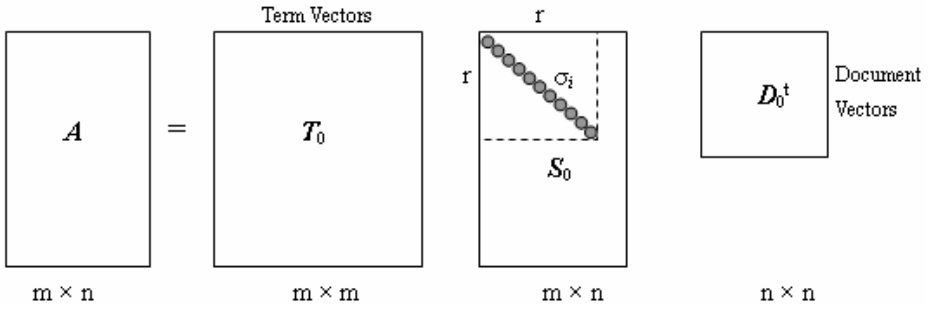


Fig. 1. SVD before dimension reduction

A primary concern of factor analysis with such a matrix is to determine "whether the table may be simplified in some way" [4]. Hence, it aims to reduce the rank of $\boldsymbol{A}$ and obtain the structural information of matrix. The decreasing order of σ_i values gives rise to a simple strategy for such an optimal approximation – reducing the rank of original $\boldsymbol{A}$ by keeping the first $\boldsymbol{k}$ largest singular values in $\boldsymbol{S}_0$ and set others small σ_i ($k < i \leq r$) to zero. In fact, rigorous mathematical proofs [5] have demonstrated that an reduced matrix $\boldsymbol{A}_k$ constructed from the k-largest singular triplets of $\boldsymbol{A}$, is the best approximation to $\boldsymbol{A}$. It can reveal important information about the structure of a matrix. The representation of $\boldsymbol{S}_0$ can then be simplified by deleting the zero rows and columns to obtain a new k-by-k diagonal matrix $\boldsymbol{S}$. Likewise, the corresponding columns of $\boldsymbol{T}_0$ and $\boldsymbol{D}_0$ are removed to derive a new left singular vector $\boldsymbol{T}$ and a new right singular vector $\boldsymbol{S}$. The resultant matrix is a reduced model:

$$\boldsymbol{A} \approx \boldsymbol{A}_k = \boldsymbol{T}\,\boldsymbol{S}\,\boldsymbol{D}^t,$$

Where $\boldsymbol{T}$ is the m-by-k matrix whose columns are the first k columns of $\boldsymbol{T}_0$, $\boldsymbol{D}$ is the n-by-k matrix whose columns are the first k columns of $\boldsymbol{D}_0$. The new model contains $\boldsymbol{k}$ factors that form the basis of the semantic vector space $\boldsymbol{A}_k$ that is closest to the original vector space $\boldsymbol{A}$ in the sense of least-square-fit. This is illustrated in Figure 2, where shaded rectangles represent the reduced matrices.

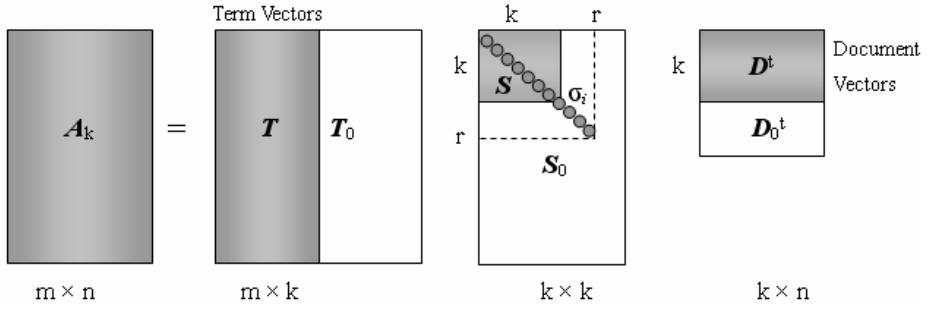


Fig. 2. SVD after reduction to k dimension

It is this reduced shaded $\boldsymbol{A}_k$ that states the hidden relations – i.e. the semantics – between terms, terms and documents, and documents. For example, the term-by-term comparison can be conducted using inner product of smaller factorised matrices as follows: $\boldsymbol{A}\boldsymbol{A}^t \approx \boldsymbol{A}_k\boldsymbol{A}_k{}^t = \boldsymbol{T}\boldsymbol{S}\boldsymbol{D}^t\,(\boldsymbol{T}\boldsymbol{S}\boldsymbol{D}^t)^t = \boldsymbol{T}\boldsymbol{S}\boldsymbol{D}^t\boldsymbol{D}\boldsymbol{S}^t\boldsymbol{T}^t$, since $\boldsymbol{D}$ is orthogonal, so $\boldsymbol{D}^t\boldsymbol{D} =$ I, hence $\boldsymbol{T}\boldsymbol{S}\boldsymbol{D}^t\,\boldsymbol{D}\boldsymbol{S}\,{}^t\boldsymbol{T}\,{}^t = \boldsymbol{T}\boldsymbol{S}\boldsymbol{S}^t\boldsymbol{T}^t = \boldsymbol{T}\boldsymbol{S}\,(\boldsymbol{T}\boldsymbol{S})^t$. That is to say the term-by-term similarity is the dot product of two rows in the smaller matrix $\boldsymbol{T}\boldsymbol{S}$. Similarly, the document-by-document comparison can be formulated as $\boldsymbol{A}^t\boldsymbol{A} \approx \boldsymbol{A}_k{}^t\,\boldsymbol{A}_k = \boldsymbol{D}\boldsymbol{S}\,(\boldsymbol{D}\boldsymbol{S})^t$, i.e. the dot product between two rows in the smaller matrix $\boldsymbol{D}\boldsymbol{S}$. Moreover, the similarity between a term and a document is literally the value of the cell in the reduced matrix $\boldsymbol{A}_k$, thus $\boldsymbol{A}_k = \boldsymbol{T}\boldsymbol{S}\boldsymbol{D}^t = \boldsymbol{T}\boldsymbol{S}^{1/2}\,(\boldsymbol{D}\boldsymbol{S}^{1/2})^t$, i.e. the dot product between two rows in smaller matrix $\boldsymbol{T}\boldsymbol{S}^{1/2}$ and $\boldsymbol{D}\boldsymbol{S}^{1/2}$ respectively.

For information retrieval, a generic user query can be converted into a pseudo-document $\boldsymbol{D}_q = \boldsymbol{A}_q{}^t\boldsymbol{T}\boldsymbol{S}^{-1}$, where $\boldsymbol{A}_q$ is the query's term vector in the original vector space $\boldsymbol{A}$. $\boldsymbol{T}\boldsymbol{S}^{-1}$ means to scale the original query term vector using the inverse of a corresponding singular values on the $\boldsymbol{T}$ space. Once such a conversion is achieved, any user queries can be compared to existing documents and terms in the reduced semantic space $\boldsymbol{A}_k$ using a score function. While inner product, as illustrated earlier, has been proven the most natural score function in the reduced space [6], most LSA applications have used the cosine between two vectors in the classical VSM model as shown in Section 3.1. Research in [7] has summarised four variants of LSA cosine score functions used by existing LSA research and applications, but the author was also unable to find a theoretical basis for opting for one score function over another. For applications other than IR, the original documents are often used for the purpose of training – i.e. creating the *semantic space*. We will discuss the semantic space in following sections.

3.3 Rationale

The hidden association derived by LSA are not just simple proximity frequencies, co-occurrence counts, or correlations in usage, but are based on SVD, which captures not only the surface adjacent pair-wise co-occurrences of terms (keywords) but the thorough patterns of occurrences of a great number of terms over very large numbers of text corpus. The word constraint information provided by higher-order associations is extracted by LSA by analysing tens of thousands of different episodes of past sentences and paragraphs. This extracted information enables computers to generate meaning similarity and distance judgments.

The central idea of SVD is to derive matrix $\boldsymbol{A}_k$ not by recreating the original term document matrix $\boldsymbol{A}$ exactly. This, in one sense, captures most of the important underlying structure in the association of terms and documents, yet at the same time removes the noise or variability in word usage that affect keyword-based retrieval methods. For example, sine the number of dimension $\boldsymbol{k}$ is much smaller than the number of documents $\boldsymbol{n}$ and number of original terms $\boldsymbol{m}$, some seemly different terms will be 'squeezed' into the vicinity locations on this newly created the semantic space only because they occur in similar documents rather than in the same document.

Why SVD, and in particular, the dimension reduction, is so helpful in finding the true relations (e.g. the similarity)? While this is still an open topic that is lack of

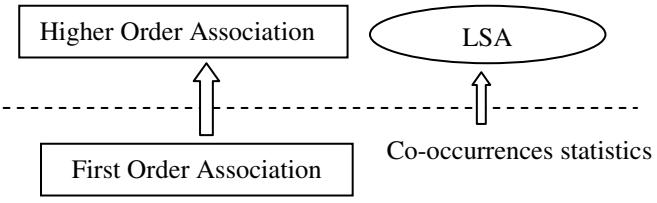


Fig. 3. Higher-order association

rigorous mathematical proof for the potential answers (e.g. removing the noise data, finding the statistical distribution), authors in [8] hypothesise that one reason could be that the original data are generated from a source of the same dimensionality and general structure as the reduction. This assumes that there exists a true semantic space, from which the original documents corpora are generated. This true semantic space has the same dimensions and structure of the reduced SVD matrix. In other words, the reduced matrix reflects the truth of the semantics, by which the data source is created. Furnas et al. in the first paper [9] on SVD applied in IR state that "the problem, of course, is that we do not have access to the true matrix; we must settle for only an estimate". Here the true matrix is the true semantic space, the estimate could be the original (what they call 'observed') data source matrix, or could be the reduced estimated low-dimensional matrix. But the reduced matrix has made a 'better' estimation due to it attempts to capture the "structural model of associations/patterns among terms". One reason could be that the cells of the observed matrix are used to estimate parameters of the underlying model. Since there are fewer parameters than data cells, this model can be more statistically reliable than the raw data, and hence can be used to reconstruct an improved estimate of the cells of the true matrix (the semantic space).

It has been found that most of the hidden association that LSA has inferred among words is in documents where particular words did not occur. As described in [8], relationships inferred by LSA are "relations only of similarity that has mutual entailments of the same general nature, and also give rise to fuzzy indirect inferences that may be weak or strong and logically right or wrong". Such a higher-order hidden structure of the association of terms and documents is revealed with the use of linear algebra (SVD). This is different from "readily appreciated" [10] first-order associations (as shown in Figure 3), which do not contain the only type of information that exists amongst words; nor do such associations necessarily hold the most important information. Higher-order association complies with the basic way how human brain relates similar words: humans have an intuitive grasp of which meanings are similar and which are not, and no philosophical or other analysis to can even fully explain how or why people understand intuitively which words can be swapped in certain contexts.

4 Applications

Most applications of LSA come from two major research fields: Information Retrieval and Psychology-related disciplines such as psycholinguistic, cognitive science, etc. The abundance application of LSA in IR field is not surprising as mathematical foundation

of LSA is built on top of the VSM, the most popular IR model to date. On the other hand, the nature and motivation of LSA (SVD) is the two-mode factor analysis. As well known, it is in the field of psychology that factor analysis has its start and where undoubtedly the greatest amount of theoretical and empirical work has been done. Hence, it is very natural for LSA flourishing in this field during the past two decades.

4.1 Application in Information Retrieval (IR)

LSA has been extensively applied in the area of Information Retrieval. In IR field, LSA is more often referred to as Latent Semantic Indexing (LSI) as automatic indexing is the central issue dealt with in most IR research, and LSA adds an important step to this document indexing process [2]. One prominent advantage of LSA over existing VSM is that LSI does not require an exact match to return useful results under the circumstances that two documents may be semantically very close even if they do not share a particular keyword. Where a plain keyword search will fail if there is no exact match, LSI will often return relevant documents that don't contain the keyword at all as will be shown in our prototype experiment, but have semantically overlapping (context or topic) vectors with the used keyword. LSA will perform better is more keywords are used, thus providing context information for the search and mapping more document vectors. On the whole, using LSA for indexing can significantly improve three important characteristics of a search engine [11]: recall (find every document relevant to the query), precision (no irrelevant documents in the result set) and ranking (most relevant results come first). We are also particularly interested in LSA in component retrieval area as the concept of Web service evolves from the concept of component. Investigating component retrieval can provide insightful information for service discovery. Authors in [12] has built a Java reuse repository using LSA for component retrieval. Similarly, research in [13] proposed an active component repository systems that support "reuse-within-development" using real-time LSA component retrieval.

4.2 Applications in Human Knowledge Acquisition

LSA has also been successfully applied in modelling human conceptual knowledge. The capability of LSA's reflection of human knowledge has been established in a variety of ways. For example, its scores overlap those of humans on standard vocabulary and subject matter tests; it mimics human word sorting and category judgements; it simulates word-word and passage-word lexical priming data; and it can even accurately estimates passage coherence, learnability of passages by individual students, and the quality and quantity of knowledge contained in an essay. LSA acquires words at a similar pace to human children, sometimes exceeding the number of words to which it is exposed [14]. LSA's knowledge of synonyms is as good as that of second-language English speakers as evidenced by scores on the Test of English as a Foreign Language (TOEFL [14]). LSA can tell students what they should read next to help them learn (see Wolfe, 1998). Some research reported that LSA can even interpret metaphors like, "My lawyer is a shark" [15].

5 Issues

In this section, we discuss several well-known issues associated with LSA that have been constantly reported from the literature and applications. Further more, we identify three major issues needed to be addressed when applying LSA in semantic Web services, in particular, service discovery.

5.1 Issues in LSA

The first issue for LSA is that it makes no use of word order, thus of syntactic relations or logic, or of morphology. As shown in [16], LSA does not do very well on single sentences as it does pretty satisfactory on single words or non-syntactic texts such as a long passages of words. However, several approaches have been proposed to address this well-known LSA weakness. One method [17] seems promising: it uses surface parsing of sentences so that the components can be compared separately with LSA and String Edit Theory is employed to infer structural relations.

The first dimension of the k-dimensions is problematic [18]. Since all elements in the original matrix $\boldsymbol{A}$ are non-negative, the entries in first dimension of $\boldsymbol{T}$ always have the same sign. And the mean of their values is much larger than that of other dimensions. As a result, the cosine value between any two documents, if the term method (i.e. the text vector can be computed as a function of the term vectors) is employed, is inevitably related to the number of terms in the document. This property makes it very hard to reveal the real similarity without considering the sizes of the documents. However, this issue is not very prominent when retrieving documents rather than comparing documents is desired. In a nutshell, when using LSA cosine value for the measure of similarity between documents, the size of the texts has to be thoroughly considered.

Following the first dimension issue, a more fundamental question is: "what do all these dimensions really mean?" So far, there is no clear definition or interpretation on these latent (hidden) factors, which are used in a rather abstract mathematical manner. First dimension phenomena is the only observed findings regarding the pattern of these dimensions (factors), no further research has done beyond that. Such a blur way of conducting LSA causes many confusing, if not totally wrong, solutions in face of various applications. For example, the appropriate number of dimensions (factors) is a well-known question plaguing LSA researchers. Currently, only practical experiments can perhaps provide some hints, which is however only limited to those experiments settings.

There is also considerable debate as to what extent LSA captures first order co-occurrence or higher order co-occurrence. Recent evidence from [19] shows that although second order effects do occur, large changes in the similarity measure between two words can be seen when a document is added to a training corpus in which both words occur (first order co-occurrence). In other words, first order association seems also receive adequate support from LSA.

Last, performance is also an issue that needs to be addressed since large matrix reduction such as SVD does take tremendous resources in terms of CPU and RAM. We will discuss the LSA complexity and scalability in Section 7.6 and 7.7.

5.2 Issues in SWS

Introducing LSA to semantic Web service is a novel yet challenging task. It is true that LSA can extract complex patterns of association contained in word usage from many tens of thousand of sentences. However, the uniqueness of Web services, in particular WSDL files, has raised a number of issues. First of all, WSDL is not a human natural language. It is an XML-based interface language meant to be processed automatically by machines. Therefore, it is far more structural and compact than general natural language. For example, an online foreign exchange Web service can have an operation with signature "decimal AUD2USD(decimal quantity)". Intuitively, it would be difficult for LSA to build the association between this operation and the concept of 'Currency Conversion' that a service consumer is looking for. This is because WSDL is designed for machine-to-machine interaction at the syntactic level. Therefore, word meanings that LSA can infer accurately might not apply WSDL. One promising approach appears to conduct rigorous analysis on WSDL corpus with sub-language [20] patterns.

One may wonder that WSDL has provided a flexible extension mechanism to support annotations and comments written in natural language. To our best knowledge, most existing WSDL files are automatically generated from binary interfaces written in advanced programming languages such as Java, C#, or Delphi, etc. To avoid the error-prone XML syntax and seemingly daunting schema definitions, software vendors and open sources have provided powerful tools that can transparently convert object models into WSDL (e.g. Java2WSDL from Axis[1]). As a result, most developers provide Web services without dealing with actual WSDL files. For example, this can be done by just pressing the "import from WSDL" or "export to WSDL/ Endpoint" buttons in many IDEs. We believe this has its own benefits for wider and quicker adoption of Web services. However, it does bring some problems. For example, it is not feasible for Web services users to annotate thousands of WSDL overnight. Nor can we have them include well-built ontology definitions in WSDL files at this stage. This calls for a solution that can suffice most Web users at the scale of Web in terms of time, space, and human resources. This chapter is our initial work towards this end.

Third, unlike traditional Information Retrieval (IR), service consumers cannot choose relevant Web services by just browsing the WSDL. While a document (e.g. a Web page) has static ('as is') information for a person to read, a Web service provides some level of skills, ingenuity, and experience that is consumed by various software applications. Such fuzzy and abstract characteristics of a service made service discovery and selection far more difficult than what traditional search engines have provided. Moreover, interacting with any web service always necessitates a transaction between two distantly located software programs that have never 'talked' with each other before. Such uncertainty and loose-coupling also bring about many technical issues that have not been addressed by existing search engines and IR techniques. We leave this part as our future work. In this chapter, we provide our preliminary experiment results of LSA application in semantic Web services.

[1] http://ws.apache.org/axis

6 Experiment

We have conducted several experiments, where LSA has been utilised in finding web services based on semantic concepts rather than simple keywords. This section discusses the experiment settings, i.e. the environment where Web service discovery can be conducted using the LSA model. In order to provide an overall picture to readers, we first introduce the experiment conceptual model that converts the service discovery problem into an IR problem that can be solved by LSA. Based on the conceptual model, we then implement a proof-of-concept software prototype that has realised the LSA techniques and that is able to convert Web services into regular documents for LSA process. Last, we provide parameter setting information that will be manipulated during the experiment to verify the performance of LSA applied in SWS discovery.

6.1 Conceptual Model

Central to the conceptual model is our main idea, which aims to convert the semantic Web service discovery problem into IR problems that can be processed by the LSA model. To that end, Web service discovery can be partially considered as WSDL discovery. This is because WSDL file contains the textual information describing the capability of a Web service in a standardised format. This way, each Web service can be seen as an individual document containing important service advertisement information. These documents can then be used by IR models such as LSA or VSM. It is true that some important information (e.g. reputation, trustworthiness, quality of services) is missing in the current WSDL specification. However, in this Chapter, we focus solely on the functional capability of a Web service as we believe this is the fundamental discovery problem need to be solved. We leave for our future work factors (e.g. QoS) determining other aspects of the problem.

With this basic idea, we then present the conceptual model. As shown in Figure 5, the conceptual model consists of two parts: run time and design time. Design time provides techniques (e.g. LSA) that can semantically examine and analyse Web services from system perspective. Based on what we have discussed earlier, Web services in this model have been instantiated into WSDL files corpus. The initial corpus is then processed using traditional IR methods (e.g. VSM), which produce WS indices, the compact form of WS corpus with metadata stored in a search-efficient way. With the help of semantic techniques such as LSA, a semantic space is thus constructed to help users to find the most appropriate services using intelligent techniques such as matchmaking, aggregating, and clustering available at run time. Run time provides infrastructure (WS-discovery UI and intelligent algorithms) that enables 'social' activities from a user-centred perspective. We believe it is essential to incorporate adequate user participation such as recommending, blogging, and tagging to facilitate the semantics services discovery. In the rest part of this chapter, we will however focus solely on dealing with the semantic techniques provided in design time, i.e. using LSA to capture the 'latent' semantics hidden in WS indices and WSDL corpus. We believe this will pave the way for building a self-organised complete semantic space in the long run to foster increasingly growing user activities.

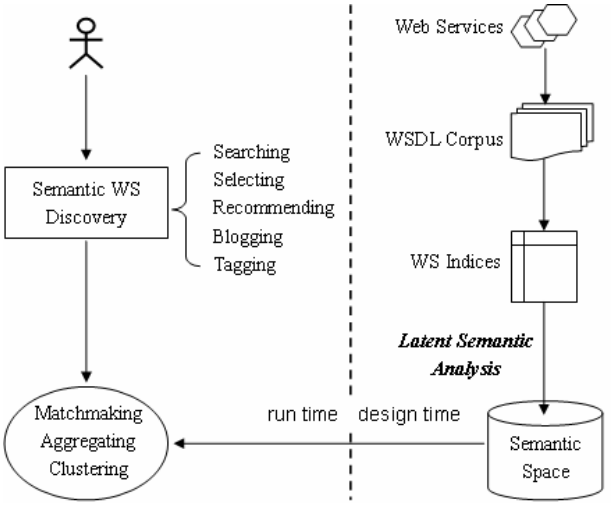


Fig. 4. Conceptual Model for Semantic Service Discovery Experiment

6.2 Software Prototype

To examine the effect of LSA indexing and retrieval accuracy for semantic Web services discovery, we have developed a semantic service search engine prototype based on the conceptual model (Figure 4). The main hindrance of implementing LSA lies in several practical engineering problems in matrix processing. We had attempted to use the JAMA (a Java Matrix Package[2]) to process the SVD. Unexpectedly, it constantly receives the "out of memory" exception and takes unacceptable long time (maximum RAM heap is set to 1G bytes) to achieve the SVD for a 24230 (terms) × 3577 (WSDL) matrix. This is because the SVD method used in JAMA applies orthogonal transformations directly to the sparse matrix $\boldsymbol{A}$. Previous studies have shown that [3] this normally costs tremendous amount of memory consumption. Moreover, the matrix representation in JAMA is simply implemented as an in-memory two dimensional array. When the dimension of a matrix reaches the magnitude level of ten thousands, the actual memory consumption will reach the magnitude level of a hundred million that cannot be met by the physical RAM.

We eventually used software tool General Text Parser (GTP[3]) for LSA indexing and querying. This works gracefully in terms of time and space complexity due to several reasons. First of all, during the process of SVD, it generates binary interim matrix on the disk, and only partial matrix information is read into memory at a time when necessary to participate the calculation. Moreover, Harwell-Boeing [21] compressed matrix is used to reduce the memory consumption. Most importantly, the LAS2, a single-vector method suitable for solving large, sparse, symmetric eigenproblems is used for handling the very large-scaled sparse SVD. Readers can refer to [3] for the detailed algorithm of LAS2.

[2] http://math.nist.gov/javanumerics/jama/
[3] http://www.cs.utk.edu/~lsi

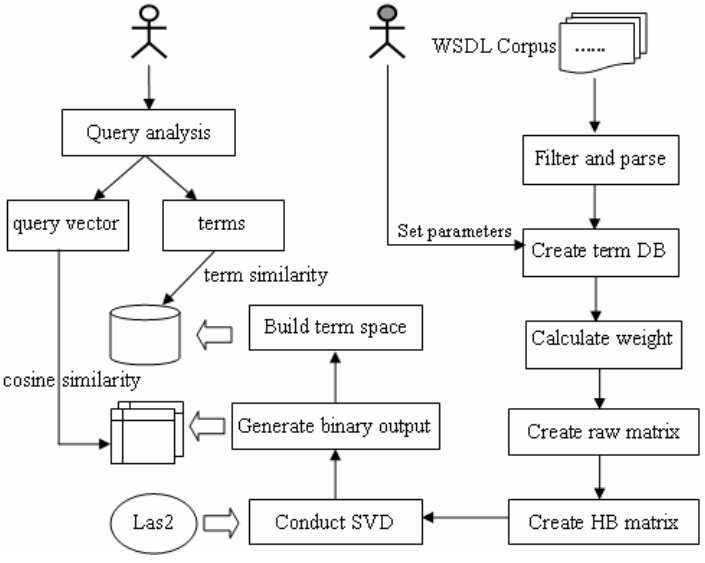


Fig. 5. Workflow of LSA in our prototype

We have also used VSM approach to generate WS indices in order to compare the performance of these two methods in semantic Web service discovery. In particular, for the same user queries, we provide a quantitative comparison between LSA method and VSM method using two IR measurement metrics – ***Precision*** and ***Recall*** (Section 7.2). A full description of this software prototype is beyond the scope of this chapter. In this chapter, we focus on the LSA experiment settings and results analysis. As shown in Figure 5, the basic LSA experiment environment consists of a number of steps that constitute a 'U' shape workflow. It is our intention to ensure the workflow is consistent with the conceptual model defined in Figure 4.

The right part of 'U' illustrated detailed steps for LSA indexing, and the left part of 'U' depicts how a user query is processed using the LSA index and the semantic term space. The workflow starts from the top right, where a collection of WSDL files is provided as the input of the software prototype. The source of this WSDL collection can be obtained through various mechanisms such as UDDI business registry, public Web services registries (e.g. *XMethod*), or data sources from previous studies (such as [22]). The WSDL corpus is then sent to the 'Filter and parse, which is fairly crucial for the success of LSA. This is because we have found that only 30% of WSDL files that we obtained from public registries are valid. They thus might not represent the capability of a Web service precisely. Without losing generality, we screen out all valid WSDL files and discarding all invalid ones. More importantly, WSDL files are XML documents with predefine XML tags. These tags (WSDL elements) do not contribute to representing the capability of a Web services. Therefore, a WSDL parser is essential to remove all XML tags and produce XML-striped plain texts ready for IR models such as VSM and LSA. Moreover, the filter can easily select desired part of the WSDL file to manipulate the dataset as we will show in Section 7.5.

When all preparation work is ready, the software administrator (shown as the grey human in Figure 5) needs to set the parameters required by the LSA experiment. Detailed parameter information will be discussed in 58633056. When parameters setting are completed, the system starts to work in a batch mode. The analysis work starts from generating the term DB, also known as the 'vocabulary' widely used in VSM to create the inverted index data structure. As the central task in this stage is to construct the term-by-document matrix, the term weights, which fill in each entry of the matrix, is of great concern. After all term weights are obtained using the scheme specified in the parameter setting, a raw matrix $\boldsymbol{A}$ can thus be built. The raw matrix is then compressed into the HB matrix, and is written to the disk. The SVD module reads the HB matrix from the disk, and loads the LAS2 algorithm to conduct the SVD, which generates the binary output consisting of a number of triples "<singular values, left vector, and right vector>".

The SVD result serves for two purposes. First, it can be directly used by service consumer (shown as the white human in Figure 5) query for general service retrieval, where each WSDL vector is compared with the query vector in order to rank WSDL files based on the cosine value. Second, the SVD result can be used for building the term space, a preliminary version of semantic space. The term space records semantic distance between any two terms collected from the term DB. The distance is calculated based on the dot product between two k dimensional term vectors. Each component in the term vector represents a factor value. The term space enables service consumer to enhance the query results, discover related Web services, and even provide more implications such as the semantic service composition.

We have also fixed up some problems in the original GTP HB matrix file representation that might cause infinite SVD calculation loop. The cosine score function discussed in [23] is used to calculate the similarity measure between a query vector and document vectors. Query vectors are generated by (1) summing the term vectors of terms in the query; (2) scaling up each term vector dimension by the inverse of a corresponding singular value. The scaling is done to emphasise the more dominant LSA factors. We have experimented in our prototype the effect of such a scaling as shown in Section 7.3. The whole prototype is developed on Java2 (JDK1.4.1) platform with JSP web user interface running on Tomcat 5.0.

6.3 Parameter Settings

As mentioned earlier, the prototype administrator needs to set several important parameter values before the experiment goes to the batch mode. In this section, we discuss these essential parameters listed in Table 1. The default values will be used if the administrator does not explicitly change them. While most parameters can be manipulated, some parameters have constant values, and thus cannot be changed. In what follows, the meaning of each parameter is introduced.

"# docs" represents the number of valid WSDL files in the corpus. By default, the size of the corpus is 2,817. However, the administrator can reduce this value by excluding certain number of WSDL files. This is sometimes necessary in order to examine the scalability of the LSA discussed in Section 7.7. Note that excluding WSDL files will affect the matchmaking performance. Therefore, this value stays at 2,817 except for the scalability test which only focuses on time and memory consumption.

Table 1. Default parameter settings for the LSA experiment

Parameter	Description	Default Value
# docs	Number of valid WSDL docs	3577
# terms	Number of terms parsed from the docs	24230
# factors	Number of reduced dimensions	105
corpus type	Different parts of WSDL	All
scale to singular value?	Whether query vector is scaled	Yes
normalise doc vector?	Whether docs vector is normalised	No
local weight	local term weight scheme	log
global weight	global term weight scheme	entropy

"# terms" records the number of terms parsed from the WSDL corpus. As mentioned earlier, these terms are all XML-stripped words in the plain texts. This value also indicates the size of the term DB, where each term has an individual entry. "# factors" represents the number of factors retained in the reduced matrix. If not specified, GTP software will find the 'optimised' value. However, as discussed in the issues of LSA, there are rigorous proofs exist supporting the optimised number of this value. Hence, it is useful to test this parameter to examine its effectiveness as will be discussed in Section 7.3.

"Corpus type" refers to the WSDL elements that will be indexed and searched during the LSA-based service discovery. Currently, it has enumeration values such as "All", "PortType", "Operation", and "Messages". Section 7.5 will discuss the different matchmaking performance based on different values for this parameter. "Scale to singular value?" refers to an option as to whether the query vector is scaled by the singular values before calculating the cosine similarity. Such a scaling can be used to emphasise the more dominant LSA factors. "Normalise doc vector?" is another option that determines whether or not the length of the document vector should be taken into account. By default, it is switched off. This means that the length of the document vector is considered as a factor affecting the similarity score of each WSDL file. Section 7.3 will demonstrate the effect of these two parameters. "Local weight" refers to how to calculate the importance of a term in one WSDL file, whereas "global weight" refers to how to work out the importance of a term in the whole WSDL corpus. As discussed previously, these two parameters determine the term weight, i.e. the original entry value for matrix $\boldsymbol{A}$. Test results related to these two will be discussed in Section 7.4.

7 Evaluations

In this section, we provide the experiments result collected from our prototype system. We start by two qualitative experiments which demonstrate the semantics that our LSA-enabled prototype can bring to standard web services – the improved *recall* measure and potential term ontology learning. Given that improved *recall* might accompany the poorer *precision*, we then present experimental *precision* measure by varying three eminent LSA parameters and one web services specific parameter. For each manipulation we conduct predefined 10 queries, which contain the most frequently entered query terms that had been captured by prototype's logging system.

For the moment we only test the single term query, future work will be carried out on complex terms queries. The mean retrieval precision is then collected for the first 20 returned services under each manipulation. Experiment performance is provided in the end to show the empirical applicability of our prototype system.

7.1 Higher-Order Association

The higher-order association is one of the most appealing and unique characteristics of LSA. It enables the semantic (topic)-based discovery rather than keyword-based search. In this section, we demonstrate the terms-based higher order association, which is constructed through LSA service indexing techniques.

Figure 6 shows a screenshot of our prototype system GUI – a typical search engine web page that displays a list of Web services on the topic "calculator", which has been input in the search box. Our LSA-based search engine has returned twenty Web services that can do certain kind of 'calculation'. Perusing the first service in the ranking list, (http://ausweb.scu.edu.au/aw02/papers/refereed/kelly/MathService.wsdl), we are unable to find any occurrences of string "calculator" in its WSDL file, or in its URL. However, this service is ranked at the first place when a service consumer needs a service such as a 'calculator'. Hence, LSA has automatically built a hidden semantic association between 'calculator' and 'maths' even though they do not co-occur in any WSDL files. Such a higher-order association cannot be captured by the VSM model with the original term-by-document matrix or any keyword-based service searching mechanisms.

Table 2 presents a partial term space based on higher-order associations. For each term (bold face) in the left column, we provide ten semantically-close terms shown in the right column. These ten predefined query terms ("maths", "book", "weather", "search", "credit card", "fax", "stock", "quote", "bank", and "SMS") are chosen because they are frequently submitted by service consumers according to the search engine

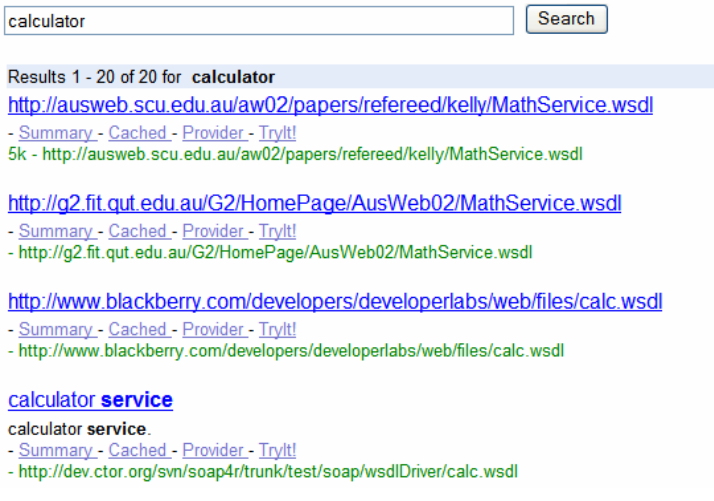


Fig. 6. Higher-order association between "Calculator" and "Math"

Table 2. Higher-order association between terms

maths	parameter - 0.91 number - 0.7 result - 0.31 add - 0.23 complex - 0.16 text - 0.16 address - 0.14 document - 0.12
book	bible - 0.97 word - 0.93 order - 0.92 key - 0.83 title - 0.82 app - 0.24 set - 0.23 address - 0.18 real - 0.16 music - 0.13
weather	city - 0.99 zip - 0.87 global - 0.72 arg - 0.4 search - 0.39 info - 0.33 service - 0.32 job - 0.23 set - 0.17 add - 0.1
search	google - 0.73 query - 0.72 simple - 0.46 page - 0.45 key - 0.39 cached - 0.37 shopping - 0.29 email - 0.16 find - 0.15 set - 0.14
credit card	process - 0.67 send - 0.41 product - 0.37 expiration - 0.36 password - 0.29 address - 0.27 sale - 0.26 submit - 0.17 detail - 0.16 cc - 0.16
fax	send - 0.96 address - 0.58 message - 0.48 image - 0.43 document - 0.35 code - 0.19 incoming - 0.18 order - 0.18 outgoing - 0.16 elements - 0.09
stock	quote - 0.96 index - 0.91 daily - 0.67 headline - 0.65 symbol - 0.41 earning - 0.31 schema - 0.27 license - 0.21 key - 0.17 split - 0.15
quote	stock - 0.92 symbol - 0.64 document - 0.3 text – 0.28 top - 0.27 gainer - 0.2 loser - 0.2 ticker - 0.19 currency - 0.18 translate - 0.15
bank	number - 0.95 detail - 0.91 sale - 0.77 card - 0.48 voice - 0.46 credit - 0.38 key - 0.34 arg - 0.28 check - 0.19
SMS	send - 0.98 email - 0.63 status - 0.52 address - 0.2 process - 0.19 inp - 0.16 credit - 0.14 currency - 0.12 oid - 0.1

query log. The number right after each term indicates the similarity between this term and the term in the left column. For example, the similarity between 'maths' and 'add' is "0.23".

The higher-order associations between terms have perhaps provided a very promising approach to build a light-weight ontology or taxonomy in a semi-automatic manner. An interesting research direction is thus to integrate these higher-order associations with end user activities such as feedback, blogging, and tagging to build and maintain a generic semantic space serving the user-centred semantic web services discovery and aggregation.

7.2 Comparison between LSA and VSM

In order to objectively compare these two methods, we use two metrics from the Information Retrieval (IR): *precision* and *recall*. In IR [24], **Precision** is defined as "*the fraction of the retrieved documents which is relevant*"; and **Recall** is defined as "*the faction of the relevant documents which has been retrieved*". We thus conduct two sets of experiments for service discovery: the keyword-based VSM and semantic-based LSA. In each experiment setting, we carry out the same 10 queries as we did in Section 7.1 for demonstrating the higher order association. We then measure the Precision and Recall respectively. We finally calculate the average precision values [24] at each recall level as shown in the Precision-Recall curves (Figure 7).

In Figure 7, the concept-based LSA has poorer precision result (74% compared to 100% for keyword-based matchmaking) when the recall is low. This is caused by the inaccuracy of higher-order association found in LSA. Hence, some irrelevant services are included and ranked very high in the discovery result list. For example, when we

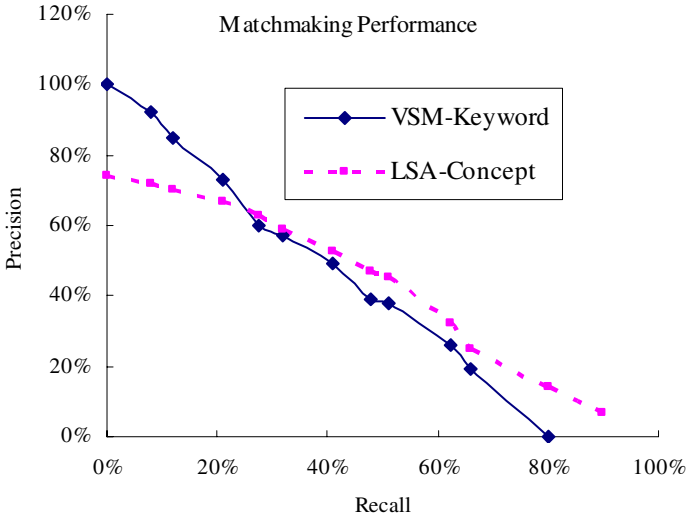


Fig. 7. Precision-Recall Curves for VSM and LSA

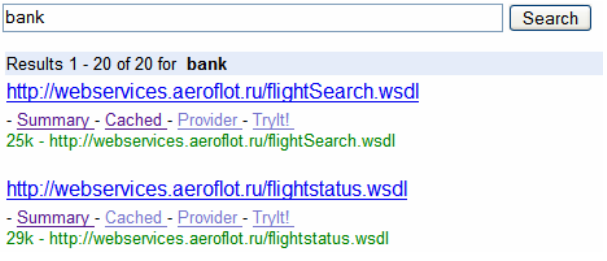


Fig. 8. LSA-based service discovery shows poorer precision

query "bank", the LSA presents the first two services as shown in Figure 8. However, neither of them ("flightSearch" and "flightstatus") is relevant to Web services that are related to banking services.

On the other hand, LSA has generally shown better recall result especially when the precision becomes lower. In the keyword-based VSM (Figure 7), however, precision at levels of recall higher than 80% drops to 0. This is because keyword-based discovery approach fails to find the remaining 20% relevant Web services due to the limitation of literal keyword matching. Take another example, when we query "credit card", the LSA-based service discovery can find two extra Web services that do not contain the keyword "credit card" as illustrated in Figure 9. The "address finder" service is chosen because we have found most online credit card Web services provide services to validate the address of card holders. To some degree, such a semantic-based discovery has implications for semantic-based service composition. Similarly, the "GeoFinder" service is returned as it is an essential part of the "address validation". The hidden "has-a" relationship between two concepts "credit card" and "address" is implicitly captured by LSA.

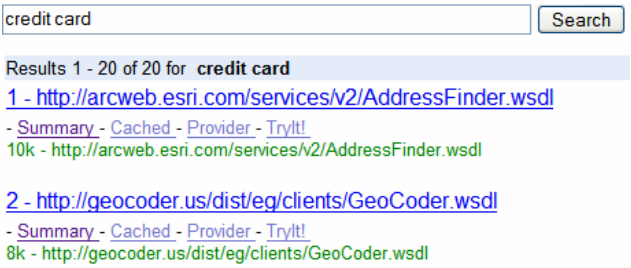
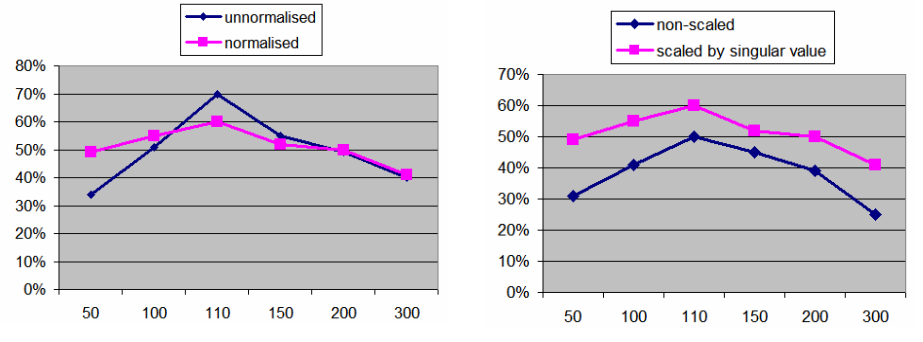


Fig. 9. LSA-based service discovery shows better recall

7.3 K-Factor and Score Function

Selection of the number of dimensions (***k***) is a challenging task. There is so far no rigorous mathematical proofs to show which values have the best performance, neither is there a intuitive way to determine the best k. Currently, there are repeated empirical experiments evaluations that can provide some hints for choosing the ***k***, leaving it an open issue for LSA. Author in [25] have found that the precision of her results set peaked at around 100 factors and then slowly dropped as the factors increased. She also notes that "the number of dimensions needed to adequately capture the structure in other collections will probably depend on their breadth". In our experiment, we have chosen range (50 – 300) to conduct the service discovery and the result is shown in Figure 10, which shows the average precision is at a maximum for 50 to 300 factors.

The score function computed in the reduced dimensional space is normally the cosine between vectors. Empirically, this measure tends to work well, and there are some weak theoretical grounds for favouring it [7, 23]. In order to preserve cosine similarities in the original space, one can length-normalise the documents before SVD. However, some research has shown that the additional use of the length of LSA vectors to be useful. This is because the length reflects how much was said about a concept rather than how central the discourse was to the concept. Therefore, we have made it a parameter in our experiment – document normalisation, i.e. whether or not



Normalised by length Scaled by singular value

Fig. 10.

to take into account the length of the document vector. The second variable that we have manipulated is to control whether the query vector is scaled by the singular values before calculating the cosine similarity.

7.4 Term Weights

The settings and calculation of term weights (global or local) are crucial for IR models such as VSM and LSA. In order to obtain these two weights, one needs to process the entire corpus to obtain some important measures such as the frequency of occurrences. In our prototype, we have considered the following different types of weighting schemes in Table 3. Readers can refer to [25] for a complete understanding on these schemes.

Table 3. Term weight

Local	Application
tf	The corpus spans general topics
log	$1/\sqrt{\log(1+tf)^2}$
binary	Only the presence as denoted by "1", or absence by "0" of a term is included in the vector

Global	Application
idf	Only if the corpus is relatively static
idf2	$\log(ndocs)/\log 2 - \log(df+1)/\log 2$
entropy	Consider to take out noises

We have thus carried out 10 predefined queries for each one of these nine (3×3) combinations of local-global term weight schemes. The precision is measured to examine the effectiveness of these weightings. The overall average precision under each weigh scheme is given in Figure 11, whereas the average precision for all weight schemes is shown in Figure 12. Figure 11 indicates that in our experiment "log-entropy'

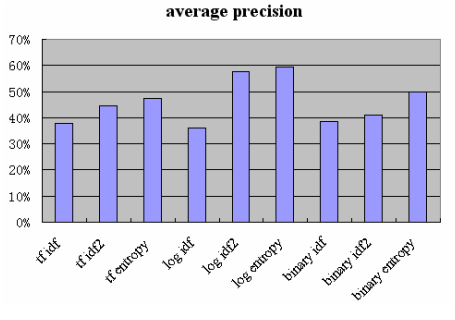


Fig. 11. Average precision for each term weight

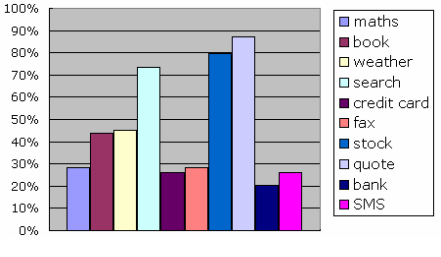


Fig. 12. Average precision for each query term

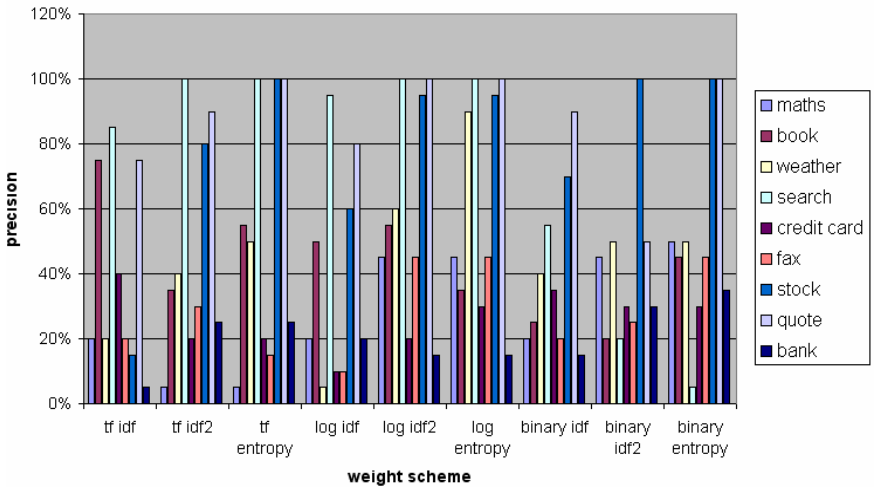


Fig. 13. Individual query precision under different term weights

term weight has obtained the best precision result. Figure 13 shows the individual query result under different term weights. It would be very interesting to explore the hidden pattern or reason that why some query terms have much better precision result than others, which might provide some useful information on the nature of the WSDL files as well as domain-specific data.

7.5 WSDL Corpus

Literature [8] has shown that the size of document corpus can affect the LSA performance. Unfortunately, there is little hard evidence on what the "ideal" size of LSA corpus might be. Intuitively, adding additional texts is unlikely to reduce performance, so a basic best practice will be "the more the better". In this experiment, the corpus contains 3577 WSDL documents with 24230 individual terms. In order to test the effect of size on the LSA performance, we have conducted three sets of experiments. In each experiment, we build LSA index on one type of data elements of WSDL files. This way, the number of terms in the WSDL corpus is manipulated as shown in Table 4.

Table 4. Manipulate the size of the Corpus

	All	Operation only	Service and Port Type
# docs	3577	3577	3577
# total terms	24230	3657	1520
Avg. #terms/docs	8.6	1.3	0.53
% non-zero	102806	22926	9847

Note: The value of Avg. #terms/docs could be less than 1 because some too frequent words are filtered out before the LSA.

The test result is shown in Figure 14and Figure 15. In the "Operation only" corpus, we found that the search performance is very different from the "All" corpus. We notice that the low precision performance for query "stock" in the "Operation only" corpus, but the high precision performance for the query "quote". Moreover, it is surprising to observe that "Operation only" corpus has the worst precision. Presumably, operation names appear to capture the nature of a Web services, and hence can semantically represent the capability of a Web service. However, our experiment does not support this assumption.

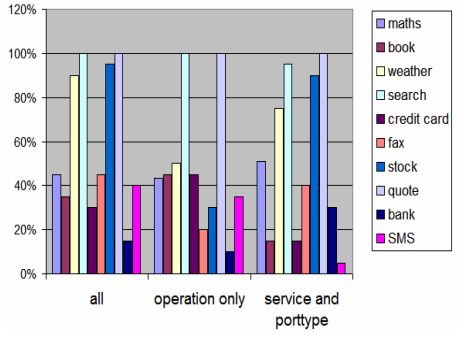


Fig. 14. Average precision for each corpus

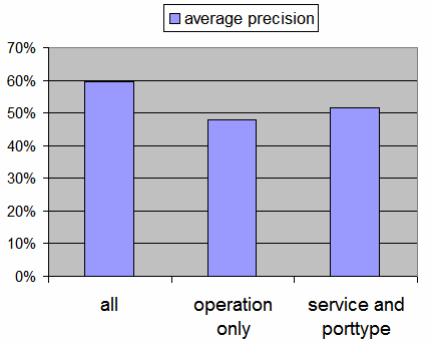


Fig. 15. Average precision for each query term

7.6 Complexity

Computational complexity is one of the important criteria in examining the performance of algorithms and models. In their original work [2] where LSA is firstly proposed, the authors have mentioned the time complexity of LSA. In particular, the two-mode factor analysis (i.e. SVD) time complexity is $O(N^2 \times k^3)$, where N is the number of terms plus documents (i.e. WSDL files), and k is the number of factors. In this section, we define the LSA complexity problem is composed of two parts – the time complexity and the space complexity. While time complexity examines the time duration needed to conduct SVD for WSDL corpus, space complexity focuses on the memory consumption required for LSA-based WSDL indexing and searching. Both of them are important for the complexity analysis. In what follows, we will elaborate studies on both time and space complexity for general LSA applications.

Theoretically, the time complexity includes design (indexing) time and run (searching) time. However, as discussed in Section 3, the LSA searching (query) is based on the classic IR *Cosine Similarity* (i.e. the dot products of two vectors). Therefore, the searching time complexity remains the same as the regular VSM time complexity, i.e. $O(q \times n)$, where q is the number of terms in the user query and n is the number of document. Note that we assume the length of each document vector ($\|v\| = \sqrt{\langle v, v \rangle} = \sqrt{\sum w_i^2}$) has been obtained during the indexing time as it is constant and is independent of any user queries. Since q is relatively much smaller than n, the

searching time complexity can thus be reasonably reduced to a linear form $O(n)$ without losing generality. Hence, conceptually, our main concern of time complexity lies in the design time, when the SVD is conducted to construct the rank-reduced matrix for WSDL indexing.

The Single-Vector Lanczos [3] -based LAS2 method is used for calculating the SVD. According to [26], this method generally yields the time complexity

$$O(I \times O(A^T Ax) + trp \times O(Ax))$$

where *I* represents the number of iterations required by a Lanczos (large scale) procedure, which is primarily used for the standard and generalised symmetric eigenvalue problem. The Lanczos (SVD manual) procedure here is used to approximate the eigensystem of $A^T A$, and the trp represents the number of singular triples – singular values in matrix ***S***, its corresponding left singular vectors in matrix ***T***, and its corresponding right singular vectors in matrix ***D***. The SVD algorithms (e.g. JAMA) that apply orthogonal transformations directly to the sparse matrix ***A*** often requires tremendous memory consumption. Previous studies show that the Lanczos procedure used in LAS2, however, have less memory consumption that has been reported 'acceptable' in [3].

7.7 Scalability

To particularly examine the time and space complexity of LSA applied in semantic Web service discovery, we have conducted the scalability experiment. Scalability is defined as the capability of the system to maintain or increase the load – large numbers of requests, or interactions among components – without degrading performance (e.g. the server response time) given reasonable resource consumptions (e.g. memory). The key of scalability experiment is thus to simulate the 'changes' and observe the 'behaviour' of the system in response to these changes in terms of time complexity and space complexity. Hence, it is our belief that the scalability measurement reflects both time and space complexity from the empirical perspective. Considering the SVD complexity introduced in Section 7.6, we provide the scalability test scheme as shown in Table 5, where four groups (G1 – G4) of scalability tests are presented. In each group, we specify the Independent Variable (IV) and the Dependent Variable (DV). For example, in G1, we manipulate the number of WSDL files in the corpus as IV, and we expect to measure the time duration for LSA indexing as DV. Likewise, in G4, the number of service consumers who simultaneously submits service requests is manipulated in order to examine the system changes in memory consumption.

Table 5. Scalability test scheme

	Indexing	**Searching**
Time Complexity DV= Time (milliseconds)	**G1** IV = # of WSDL	**G2** IV = # of Service Consumers
Space Complexity DV= RAM (megabytes)	**G3** IV = # of WSDL	**G4** IV = # of Service Consumers

Our prototype is running in one PC configured with the 2G RAM, Pentium 3.59GHz CPU. We use JVM peak committed memory as the measurement for DV RAM consumption. This value is often slightly larger than the actual consumption, but it reflects the true memory reservation for our system. The test results of four groups are illustrated in Figure 16.

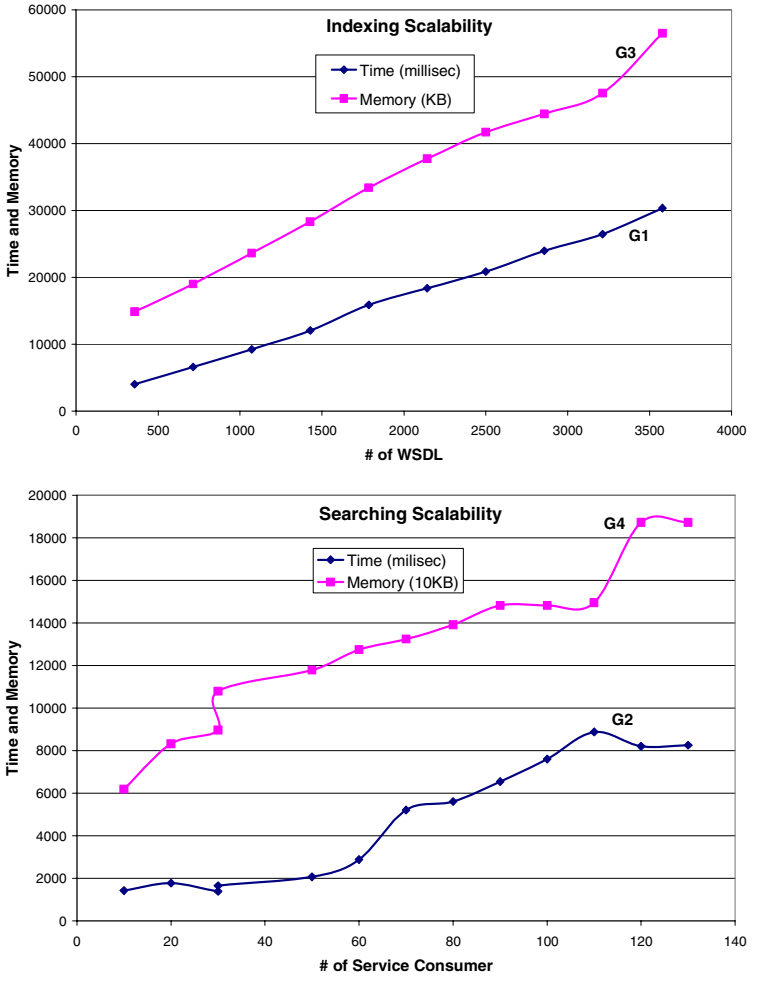


Fig. 16. Scalability test results

From Figure 16, it is clear that, in general, both time complexity and space complexity are linear. In order to quantify the scalability results, we define that

$$scalability = \frac{\Delta dv}{\Delta iv} = \frac{dv_1 / dv_0}{iv_1 / iv_0},$$

where Δ represents the change part of either dependent variable (*dv*) or independent variable (*iv*). Based on this equation, we can obtain the scalability results for four groups of experiments as follows, G1 = *75%*, G2 = *44%*, G3 = *38%*, G4 = *23%*. Since all scalability values are well less than 1, the behaviour of the system is not sensitive to changes in system loads. We can thus reasonably maintain that the LSA prototype for semantic Web service discovery is scalable.

8 Conclusions

In this chapter we provided an introduction to semantic web services, where we briefly discussed three different approaches for adding semantics to Web Services i.e. OWL based Web Services Ontology, WSMO and WSDL-S. We then explained LSA and VSM and their application in information retrieval and studied how it can be applicable to web services discovery. We found that most of current solutions and specifications for SWS rely on ontology building; a task that needs a lot of human (e.g. domain experts) involvement, and hence might not be able to scale very well in face of vast amount of web information and myriad of services providers. To address this issue we proposed a novel way to conduct SWS discovery using LSA. This technology and its existing applications motivate our solution and conceptual model in applying LSA on semantic web services. A prototype system – a service search engine – was developed based on this conceptual model and the experimental results have been evaluated in this chapter.

References

[1] Sajjanhar, A., Hou, J., Zhang, Y.: Algorithm for web services matching. In: Yu, J.X., Lin, X., Lu, H., Zhang, Y. (eds.) APWeb 2004. LNCS, vol. 3007, pp. 665–670. Springer, Heidelberg (2004)

[2] Deerwester, S., Dumais, S., Furnas, G.W., Landauer, T.K., Harshamn, R.: Indexing by Latent Semantic Analysis. Journal of the American Society for Information Science 41, 391–407 (1990)

[3] Berry, M.W.: Large scale singular value computations. International Journal of Supercomputer Applications 6, 13–49 (1992)

[4] Horst, P.: Factor Analysis of Data Matrices: Holt. Rinehart and Winston, Inc. (1965)

[5] Eckart, C., Young, G.: The approximation of one matrix by another of lower rank. Psychometrika 1, 211–218 (1936)

[6] Bartell, B.T., Cottrell, G.W., Belew, R.K.: Latent Semantic Indexing is an Optimal Special Case of Multidimensional Scaling. In: 15th Annual International SIGIR, Denmark (1992)

[7] Caron, J.: Experiments with LSA Scoring: Optimal Rank and Basis, Computer Science Department, University of Colorado at Boulder (2000)

[8] Landauer, T.K., Foltz, P.W., Laham, D.: Introduction to Latent Semantic Analysis. Discourse Processes 25, 259–284 (1998)

[9] Furnas, G.W., Deerwester, S., Dumais, S., Landauer, T.K., Harshamn, R.A., Streeter, L.A., Lochbaum, K.E.: Information Retrieval using a Singular Value Decomposition Model of Latent Semantic Structure (1988)

[10] Skoyles, J.R.: Meaning and context: the implications of LSA (latent semantic analysis) for semantics (2000)
[11] Yu, C., Cuadrado, J., Ceglowski, M., Payne, J.S.: Patterns in Unstructured Data Discovery, Aggregation, and Visualization (2005)
[12] Lin, M.Y., Amor, R., Tempero, E.: A Java reuse repository for Eclipse using LSI. In: Australian Software Engineering Conference (2006)
[13] Ye, Y.: Supporting component-based software development with active component retrieval systems. In: Computer Science, University of Colorado (2001)
[14] Landauer, T., Laham, D., Rehder, R., Schreiner, M.E.: How well can passage meaning be derived without using word order? a comparison of Latent Semantic Analysis and humans. In: 19th Annual Conference of the Cognitive Science Society, Mahwah, NJ. USA (1997)
[15] Kintsch, W.: Predication. Cognitive Science 25, 173–202 (2001)
[16] Wiemer-Hastings, P., Wiemer-Hastings, K., Graesser, A.: How latent is Latent Semantic Analysis? In: Sixteenth International Joint Congress on Artificial Intelligence, San Francisco. US (1999)
[17] Dennis, S.: Introducing word order. In: McNamara, D., Landauer, T., Dennis, S., Kintsch, W. (eds.) LSA: A Road to Meaning. Erlbaum, Mahwah (2005)
[18] Hu, X., Cai, Z., Franceschetti, D., Penumatsa, P., Graesser, A.C., Louwerse, M.M., McNamara, D.S.: LSA: The first dimension and dimensional weighting. In: 25th Annual Conference of the Cognitive Science Society (2003)
[19] Denhière, G., Lemaire, B., Bellisens, C., Jhean, S.: A semantic space for modeling a child semantic memory. In: McNamara, D., Landauer, T., Dennis, S., Kintsch, W. (eds.) A Road to Meaning, Mahwah, NJ (2005)
[20] Kittredge, R., Lehrberger, J.: Sublanguage: Studies of Language in Restricted Semantic Domains. de Gruyter (1982)
[21] Duff, I., Grimes, R., Lewis, J.: Sparse Matrix Test Problems. ACM Transactions on Mathematical Software 15, 1–14 (1989)
[22] Fan, J., Kambhampati, S.: A Snapshot of Public Web Services. ACM SIGMOD Record 34, 24–32 (2005)
[23] Berry, M.W., Drmac, Z., Jessup, E.R.: Matrices, Vector Spaces, and Information Retrieval. SIAM Review 41, 335–362 (1999)
[24] Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison Wesley, Reading (1999)
[25] Dumais, S.T.: Improving the retrieval of information from external sources. Behavior Research Methods, Instruments and Computers 23, 229–236 (1991)
[26] Berry, D.M., Do, T., O'Brien, G.W., Krishna, V., Varadhan, S.: SVDPACKC (Version 1.0) User's Guide, Computer Science Department, Univeristy of Tennessee (1993)
[27] Herrmann, M., Ahtisham Aslam, M., Dalferth, O.: Applying Semantics (WSDL, WSDL-S, OWL) in Service Oriented Architectures (SOA)
[28] Cardoso, J., Sheth, A.P.: Semantic Web Services, Processes and Applications. Springer, Heidelberg (2006)
[29] OWL-S semantic markup of web services – white paper (accessed on June 15, 2007), `http://www.daml.org/services/owl-s/1.0/owl-s.html`
[30] Marinchev, I., Agre, G.: Semantically Annotating Web Services Using WSMO Technologies. Cybernetics and Information Technologies 5(2) (2005)