Web Service Retrieval by Structured Models

Web Service Retrieval by Structured Models

Ulrich Güntzer

Wilhelm-Schickard-Institute for Computer Science

University of Tübingen

Sand 13, 72076 Tübingen, Germany

Phone +49-7071-2975477,

e-mail guentzer@informatik.uni-tuebingen.de

Rudolf Müller*

Department of Quantitative Economics

Universiteit Maastricht

P.O. Box 616, 6200 MD Maastricht, The Netherlands

Phone +31-43-3883799,

e-mail r.muller@ke.unimaas.nl

Stefan Müller

Wilhelm-Schickard-Institute for Computer Science

University of Tübingen

Sand 13, 72076 Tübingen, Germany

Phone +49-7071-2975485,

e-mail muellers@informatik.uni-tuebingen.de

Ralf-Dieter Schimkat

Wilhelm-Schickard-Institute for Computer Science

University of Tübingen

Sand 13, 72076 Tübingen, Germany

Phone +49-7071-2975485,

e-mail schimkat@informatik.uni-tuebingen.de

Abstract

Much of the information available on the World Wide Web cannot effectively be found by the help of search engines because the information is dynamically generated on a user's request. This applies to online decision support services as well as Deep Web information. We present in this paper a retrieval system that uses a variant of structured modeling to describe such information services, and similarity of models for retrieval. The computational complexity of the similarity problem is discussed, and graph algorithms for retrieval on repositories of service descriptions are introduced. We show how bounds for combinatorial optimization problems can provide filter algorithms in a retrieval context. We report about an evaluation of the retrieval system in a classroom experiment and give computational results on a benchmark library.

Keywords: Information Systems: Analysis and Design, Decision Support Systems, Networks/graphs: Heuristics, Matchings.

1 Introduction

The World-Wide-Web (WWW) as a global repository for information is a big improvement for the quality of life of people around the world: Organizing a journey, buying or selling goods, or

*Corresponding author

just searching detailed and up-to-date product and service information becomes cheaper, faster, and easier. It remains however a big problem to find suitable Web pages and services. Research to solve this problem brought up a lot of different technologies, e.g. to better describe the content of Web pages by the *resource description framework* (RDF), World Wide Web Consortium (1999), and the business driven *Universal Description, Discovery and Integration of Business for the Web* (UDDI), UDDI community (2002), or elaborated index and search strategies like pagerank, Page et. al (1998), which is used by google.com.

Of particular interest for the field of Operations Research and Management Science is the growing number of sites which offer libraries of decision support models in various modeling languages (AMPL, Fourer, Gay and Kernighan (1993), see below, GAMS (1998) at www1.gams.com/-modlib/modlib.htm, Excel spreadsheets, Ragsdale (2000), at www.j-walk.com/ss/excel/links/xllinks4.htm). For example, NetLib (1995) and a Princeton Web server, Vanderbei (2000), contain about 900 AMPL models. Remotely stored models together with suitable solvers can operate as on-demand decision support services, Bhargava, Krishnan and Müller (1997). So does NEOS offer to submit and solve AMPL and GAMS instances, NEOS project (2000). These services are likely the beginning of a market of OR-based Internet services, as projected by Fourer and Goux (2001). Using those services requires again means to retrieve the appropriate model towards a given decision problem. However, not much retrieval functionality is available. This is in part due to the lack of appropriate service descriptions, and as a consequence, of appropriate retrieval algorithms for repositories of such descriptions. Retrieval on model libraries can currently only be done by directed search based on a classification index, or by full text retrieval.

In this paper we describe a completely new approach in which *Structured Service Models SSM*, a special class of acyclic directed graphs, are used for service representation. A query model from a user is checked for similarity with models in a repository and models being most similar

are reported as retrieval result. Our approach is based on *Structured Modeling*, Geoffrion (1987), a modeling approach for decision support that made a significant contribution to the theory of modeling in Management Science. Broadly spoken a structured model is an acyclic, directed graph whose nodes represent components of the model (entities, decision variables, etc.) and whose directed edges represent definitional dependencies between components. Structured Modeling has proven to have the potential to capture the essential characteristics of a model in the field of Management Science, Jones (1990*a*). This motivates to represent decision support services by structured models and to use this representation for retrieval.

The approach described in this paper turns out to be applicable to a much broader class of services than decision support. The WWW contains a lot of information that are not explicitly made persistent in a Web-accessible manner as HTML pages. Rather, these pages are dynamically constructed by scripts or servlets. Based on user input, e.g. travel destinations, or product names, these generate HTML pages from a result of a query to an underlying information system, mostly a relational database. The set of those pages is called the *Deep Web*. Information providers use dynamic pages when they have to handle a mass of related information that is well structured and rapidly changing over time.

Dynamic pages are usually not found by automatic indexing. Spiders and other indexing agents can only follow hyper-links, but not generate user input to scripts that are the entry point to deep web sites. (Typically, such scripts are *CGI scripts*, *active server pages*, or *Java server pages*). Thus they do not index all pages that result from queries to the site. (A little more is possible if the dynamic pages are created by extending URLs by queries, and if pages with such URLs exist on static pages. In this case a search engine could enter a Deep Web site from such a static page and continue to generate and index dynamically essential parts of the site as long as it finds pages with hyper-links to other dynamic pages.) The static entry points to the Deep Web contain at most a

textual description of the site, which misses to represent the types of information behind the entry point, and how this information is structured. To give an example, suppose we are interested in demographic information in order to prepare a business investment in Flint, Michigan. A query to www.google.com with this sentence gives a large list of documents in which these key words are present, but not an entry point to a deep web site which can provide the information that we are interested in.

It is interesting to see that structural information (lost when a user's question is transformed to a set of keywords) is actually used to manage the information of the Deep Web: Most of the sites offering dynamic Web pages store their data within relational databases for which Entity-Relationship diagrams (ER diagrams), Chen (1975), accurately describe the content and relations. ER diagrams are abstract enough to deliver information about the stored data without using the data itself. Imagine a search engine that indexes the internal ER diagrams of Deep Web information systems and provides a front end to formulate queries in a structural manner like ER modeling. We will see that Structured Service Models can provide such a search engine due to the fact that the entity relationship diagrams can be converted into structured service models, and our retrieval techniques can be applied to these models. Furthermore, current efforts by the World Wide Web consortium (W3C) to establish a so-called *semantic web* rely partly on graphical descriptions of web services by so-called RDF graphs, Decker (2001). We will see in Section 3 that our system of Structured Service Model repositories and retrieval algorithms can provide the kernel for any retrieval system for graphical structures as long as these structures can be mapped into our models. While we did not define such a mapping yet for RDF graphs, it can be expected to be as simple as for ER diagrams, Müller, Schimkat and Müller (2002).

The rest of the paper uses the term *(Web) service* to summarize services like online model repositories, online decision support, and Deep Web services. Müller and Schimkat, (2001), have

shown that our retrieval approach has applications in software retrieval, too. Furthermore, other kind of information services that can be represented by structured service models may benefit of our retrieval approach. For simplicity, we decided however to restrict the motivation for this paper to the two applications discussed above. In section 2 we will introduce our modeling paradigm in the context of decision support models, and at the end of the section we briefly mention how the extension to entity relationship diagrams has been done.

Our retrieval techniques use Operations Research methods. We define the similarity of two models as a property on pairs of graphs. Computing the similarity is a combinatorial optimization problem that is related to graph isomorphism. Computing the similarity exactly is shown to be NP-complete, but the structure of models supports the design of exact or heuristic algorithms, as will be shown in Section 4. Computationally very efficient lower bounds on the similarity can be used for filter algorithms, which is the topic of Section 5. But before we elaborate the mathematical part of our paper, Section 3 presents the prototype implementation. This should give the reader a fair understanding of the context in which the algorithms are used. Also, Section 3.1 further motivates our approach by stimulating results from a small class room experiment with entity relationship diagrams. Section 6 is devoted to computational results. Section 7 gives an overview of other approaches to search for Deep Web information and other Web services, before Section 8 summarizes and concludes.

Parts of this paper have been presented at the Workshop of Information Technology and Systems, Müller and Müller (2000), and the 2nd International Workshop on Web Dynamics, Müller, Schimkat and Müller (2002). Publication in the proceedings of these workshops did not exclude journal publication. Furthermore we extend these papers by giving detailed proofs in this paper.

2 Models for Web Services

2.1 Structured Service Models

A Structured Service Model (SSM) describes a service by the type of information that is input to the service, the type of information that is computed by the service, and relations between input and output. The approach is based on *Structured Modeling* by Geoffrion (1987, 1989), which was designed to define decision models in Management Science, but which has shown its potential in representing models from other fields as well, e.g., data base models, Chari and Sen (1997, 1998). Our Structured Service Modeling can be viewed as a restricted version of Structured Modeling in order to facilitate the use of the later for service retrieval. Restrictions simplify usage and open the door towards efficient retrieval algorithms on model libraries (see Section 4).

A SSM is a directed, acyclic graph with textual node labels describing node semantics. Every node represents an item, every arc represents a definitional dependency between items. A SSM distinguishes 6 types of nodes.

An *entity* node represents a primitive item whose definition does not depend on other items. A *parameter* node represents an attribute describing an entity, or combinations of entities, and whose value is an input to the service. A *variable* node stands for an attribute whose value is computed by the service. In a decision model we can think of it as a decision variable. The definition of parameters and variables are dependent on the definition of the entities they describe. A *function* represents a rule to compute a value from variables and parameters, in a decision model it represents for example the objective. A *test* is a function that evaluates with true or false. It is an expression that defines a constraint on a combination of parameters and variables. A *multi-test* is a collection of tests. It represents the fact that a constraint has to be valid for a range of combinations of parameters and variables. We illustrate this in more detail in an example below. The distinction between parameters and variables, as well as between tests and multi-tests extends concepts of

Structured Modeling. These extensions have been introduced to ease the automated construction of SSM for models written in other widely used modeling language, e.g. AMPL (see Section 6).

The SSM graph contains directed edges (v, w) for all nodes v and w for which the definition of the item represented by node w depends on the definition of the item represented by node v. For example, if a node w stands for a parameter that is associated with an entity, represented by node v, an arc (v, w) is added. Or, if w represents a test, we add directed edges pointing to w from all variable and parameter nodes that are input to that test.

Edges are only allowed between specific types of nodes: (1) from entities to variables and parameters, (2) from variables and parameters to functions, tests and multi-tests. Thus, SSM are acyclic directed graphs with three layers of nodes: a layer of entities, a layer of parameters, and variables, and a layer of functions, tests, and multi-tests. This restricts structured modeling, as the later would allow, e.g., dependencies between functions, too. As we will see in Section 4 the restriction supports the computation of similarity of models. (It does not restrict too much the expressiveness of SSM. Indeed, edges between functions represent intermediate steps which help to modularize the model, rather than capturing semantics. Choices of modularization might be rather arbitrary. This could even negatively influence the quality of retrieval.)

Figure 2.1 shows the SSM for the Hitchcock-Koopman transportation model. The decision model describes the situation where we have a collection of plants and customers, represented by the entities PLANT and CUST. Each plant has a supply and each customer has a demand, modeled by parameters SUP and DEM. For each combination of plant and customer we observe per unit transportation cost from plant to customer, modeled as parameter COST. Decision variables are amounts of shipment between every plant and every customer, given by the variable FLOW. Total shipment to a customer has to satisfy demand, and total shipment from plant may not exceed supply. These constraints are modeled by multi-tests T-SUP and T-DEM. The objective is to

maximize revenue. It is represented by the function REV.

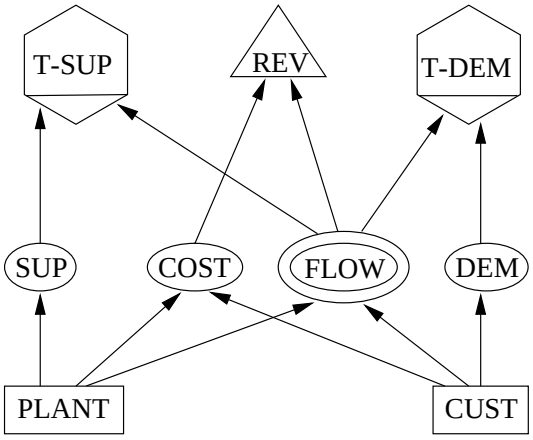

Figure 1: Structured Service Model for transportation problem

As a more general example consider an optimization problem given by

$$\begin{aligned} &\max\ f(x) \\ &\text{s.th. } g_i(x) \leq a_i \qquad\qquad i = 1, \ldots, n \end{aligned}$$

where x is a vector of decision variables, and g_i are functions that model constraints. Assume that constraints in the model can be grouped into several types, then each type will be represented by a test or multi-test in the third layer of the SSM. If we further assume that parameters and variables can be divided into different types, each of these types will have a node on layer 2. Edges between these attribute nodes and test nodes are included, if an attribute of the first type contributes to the definition of a constraint of the later type. If the optimization problem is a linear model, edges between variables and tests indicate areas of non-zero entries in the underlying constraint matrix. And edges between parameters and tests cluster the coefficients into categories. In particular in a restricted domain like linear programming models, SSM are therefore able to represent essential parts of the structure of the problem.

The task of SSM in a retrieval context is two-fold. First, they can be used as a graphical representation of a service that helps a user to decide whether the service meets her requirements. Second, providers of services can register them in repositories and users can search the repositories for fitting services. Furthermore robots might automatically create SSM from other service descriptions, like a description in an algebraic modeling language in case of a decision model. Retrieval for Web services can make use of SSM in the following way: A user submits a query to the retrieval system by creating his own SSM, describing the service she is looking for. The retrieval mechanism returns those SSM that are "close" to the query model. This approach faces us with the following research questions:

1. How precise can a SSM describe the semantics of a Web service and a user's request for such a service? What are appropriate measures for problem similarity?

2. How efficient can we do retrieval on SSM repositories with respect to a similarity measure?

3. Will retrieval based on searching for similar models provide appropriate precision and recall?

The first question will be addressed in Section 3.1 after we have described the prototype. We will give experimental results with students when applying SSM to entity relationship diagrams. For the application domain of OR/MS models we have done similar experiments with knapsack models, but with a smaller number of users. Our choice of similarity that led to this positive answer is discussed in the next section. Questions of efficiency, precision and recall are the topics of Section 4, 5, and 6.

2.2 Similarity of SSM

The similarity exploits the adjacency structure of SSM graphs. Given two graphs $G = (V, E)$ and $G' = (V', E')$, we look at partial mappings π of the nodes from G onto the nodes of G' which only

map nodes onto nodes of the same type, and define $q(\pi) = |\{(u,v) \in E \mid (\pi(u), \pi(v) \in E'\}|$. The function q expresses the number of edges in G that are implicitly mapped onto edges in G', i.e. edges $(v,w) \in E$ such that $(\pi(v), \pi(w)) \in E'$. The *matching quality* realized by the mapping π is defined as $d(\pi) = 2q/(|E| + |E'|)$. The similarity between two graphs is then defined by the maximum over all matching qualities of mappings from G onto G'. Mappings are always one to one. The matching quality is a rational number between 0 and 1 with a value of 1 indicating that there exists a mapping between the library graph and the query graph that matches exactly all edges. As we can assume w.l.o.g. that there are no isolated nodes in a SSM, this is the case if and only if both graphs are isomorphic.

Recall that SSM consist of three layers, with two types of nodes on the second layer and three types of nodes on the third layer. Thus an SSM is a 7-tuple $G = (U_E, U_P, U_V, U_T, U_M, U_F, E)$, where U_E are the entity nodes, U_P the parameter nodes, U_V the variable nodes, U_T the test nodes, U_M the multi-test nodes, U_F the function nodes, and E the edges. Given two SSM G and G', we further require that mappings π of the nodes from G on the nodes of G' map nodes on nodes of the same type, e.g. nodes in U_E on nodes in U'_E. The similarity of SSM G and G' is then defined as the maximum matching quality achievable by such restricted mappings.

One should mention at this point that the similarity can be defined with respect to any partition of nodes on layer 1, 2, and 3 into types. The finer the partition can be done in a certain application domain, the better will the structure represent the semantics of the service. Furthermore, it should be obvious that identifiers of nodes, like COST or DISTANCE in the transportation model can be used to further measure the quality of a mapping of a node from the query model on a node of the repository model. A string comparison of the node labels can be used in this context. This is a matter of implementation, rather than a conceptual extension of our approach.

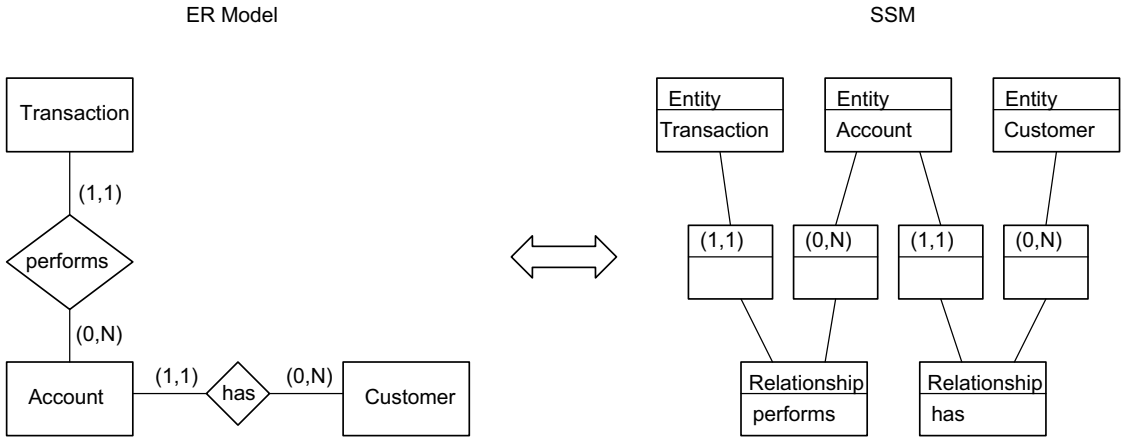

Figure 2: Mapping between ER diagram and SSM

2.3 Extension to ER diagrams

The rest of this section we devote to illustrate how SSM can be used for repositories of entity relationship diagrams. Towards this end we define a one-to-one mapping between ER diagrams and SSMs following the rules of (Müller & Schimkat 2001): Different kind of entities become different kind of nodes of the first layer. Relationships become nodes of the third layer and the cardinalities inside relationships, e.g., 1 to n or m to n, as well as aggregations become nodes on the second layer. Figure 2 shows a simple ER diagram and its corresponding SSM. The mapping rules enforce that SSMs of ER diagrams are again 3-layered graphs. As for the domain of decision models, each layer consists of different types of nodes, although their semantics is now quite different. As in the case of SSM, these types are used to restrict the domain of mappings π used in the definition of the similarity.

A mapping between SSM and UML (unified modeling language) class diagrams, which are a part of the de-facto standard for object oriented systems modeling, Object Management Group (2000), can easily be defined as well, Müller and Schimkat (2001). The same applies to RDF graphs. Whenever a model translation for a new domain of models has been defined and added to

the implementation of the system described in the next section, our approach provides the kernel for a retrieval system for this domain.

3 The FIRESTORM system

Having described our proposals for representing Web services by SSM, we turn now to a description of our prototype implementation. The purpose of the prototype is to validate our proposals, and to improve them. On the one hand this requires to benchmark the algorithms that will be described in Section 4 and 5 on large model repositories, on the other hand it requires to make real users test the system and report their experiences. We therefore implemented a Web-based client-server system. It consists of a Java applet at the client side that implements the user interface, a retrieval server with retrieval algorithms, to which the Java applet sends retrieval requests, and a database with collections of SSM models. FIRESTORM (FIrst a REtrieval SysTem for Operations Research Models) is the name of our prototype.

We start with an illustration of the user interface as it best describes the functionality of the system so far realized. After the user has loaded the Java applet from the site

http://firestorm.informatik.uni-tuebingen.de

a FIRESTORM frame opens (left part of Figure 3).

Within this frame the user can edit a SSM. The SSM currently represents, as described in the previous section, the query to the system. In a future release providers of services can use the same frame to submit models and thereby extend the collection of models in the FIRESTORM database. The client provides all funtionalities to insert, delete and move nodes and edges. A second window (Model Components) lets the user choose which types of nodes she wants to add, delete or rename. After having created a query model, the user opens the similarity checker for retrieval (right part of Figure 3). This frame offers activation of the filter algorithms from section

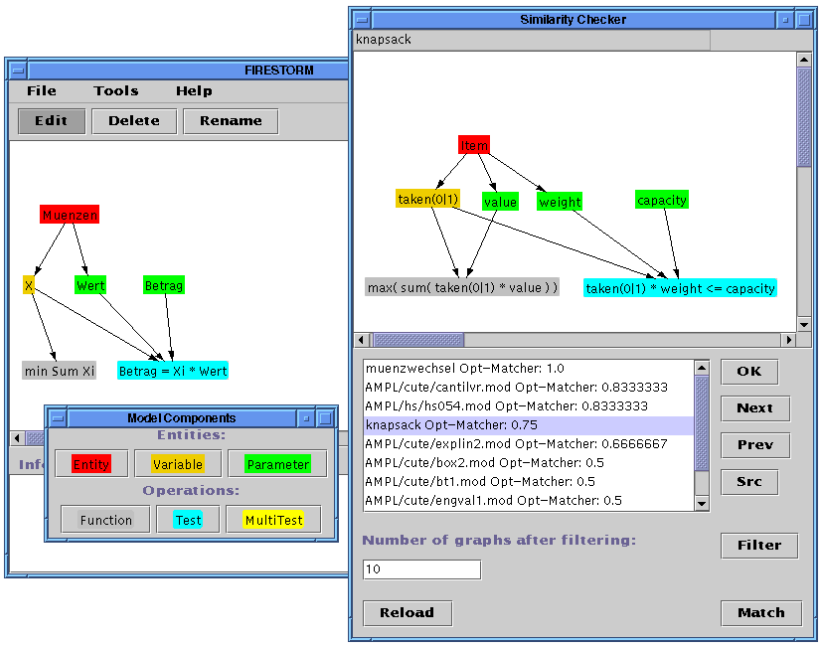

Figure 3: The FIRESTORM user interfaces

5 and the graph similarity algorithms from section 4 with a possibility to restrict the size of the query result. It contains functionalities for browsing through the retrieval results and viewing additional information (e.g. the source model, if the SSM was automatically generated from an AMPL model).

The retrieval server encapsulates the retrieval algorithms. Retrieval is done in main memory, all models are currently loaded in the beginning of a user's session. Through filter and retrieval algorithms this list is continuously reduced. Available algorithms are the sum norm filter algorithms described in Section 5, a local search for graph similarity, using two-exchanges to define the neighborhood structure with a best neighbor strategy as search strategy, and an exact method that enumerates over all assignments of the middle layer nodes (see Section 4).

A more detailed description of the software engineering aspects of the system can be found in Müller and Schimkat (2001). Important to mention at this point is that the part of the user interface

for model editing and visualization has been separated from the server along a multi-tier architecture. A mapping from a domain of models, like ER diagrams, to SSM is added to the system by adding a new model editor and visualizer and connecting this to the server. Communication with the server will take place on the basis of the SSM representation. This makes the server with its repositories and algorithms a general purpose device, that easily be extended to new domains.

3.1 Classroom experiment

In order to apply the system for ER diagram repositories, we implemented one-to-one mappings of ER diagram to SSM as described in Section 2.3. The graphical user interface for ER diagrams lets a user state queries and visualize the stored SSMs as natural ER diagrams, hiding the underlying SSM representation completely. With this extension of the FIRESTORM prototype we run a small classroom experiment that addresses question 1 from the end of section 2.1: how well does the structure of SSMs capture the semantics of the corresponding application?

Modeling is an art and one may argue that the same real world situation can be modeled in very different ways leading to different SSMs. The structural retrieval approach reduces the detection of similarity between different ER diagrams to the graph similarity between their SSMs but the retrieval quality depends on semantic similarity between ER diagrams. The restrictiveness in structure and available means to create ER diagrams somehow limits the modeling freedom, but that does not guarantee by itself that the approach is meaningful.

The focus of our working group at the University of Tübingen (Germany) is on database and information systems. Part of a database course is to learn how to create ER diagrams that describe an aspect of the real world to be managed with a database. In this course *Firestorm* is used by students to create and manage their ER diagrams. This gave us the opportunity to perform the following classroom experiment.

The course contains three exercises that require students to design ER diagrams for three

different real world cases. These cases are formulated as natural language text. The text implies the naming of entities and relationships so that we can concentrate only on the structure. Some staff members evaluated the solutions of the students and assigned credit points according to how well the ER diagrams represent the real world cases. For each exercise a staff member created a master model representing the expected solution. This made it possible to evaluate whether high similarity between the students ER diagrams and the master model coincides with a high grade, which we assume to reflect a certain similarity with the semantics of the modeled real world case. A positive answer would support our claim that the structural retrieval approach is meaningful for retrieval, at least for the domain of ER diagrams.

In our evaluation the master models were used as queries to the system with the corresponding student models building the library. The system computed the (graph) similarity between each student model and the master model. The similarities were converted into credit points by simply multiplying the similarities with maximal credit points. The results are encouraging. Figure 4 relates the credit points assigned by the staff members to the credit points assigned by *Firestorm.* The differences in credit point assignment are marginal so it seems that graph similarity between SSMs can be used to determine semantic similarity between ER models (as long as the context is clear).

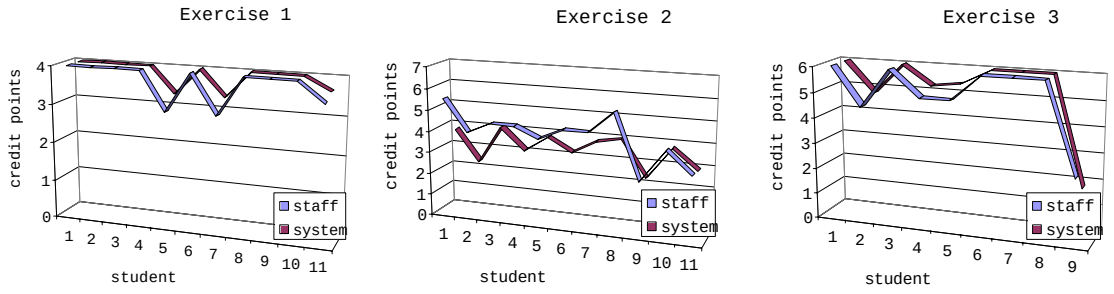

Figure 4: Credit point assignment (staff vs. *Firestorm*)

4 Similarity Algorithms

In Section 2 we introduced SSM as an approach to model Web services, in particular decision support services and deep web services. Section 3 described a Web-enabled repository for service descriptions with retrieval functionality. In this section we present the algorithmic aspects of the similarity measure used in the retrieval system.

An immediate question is whether we are able to compute in polynomial time the similarity between two SSM. The answer is unfortunately no. It follows from the following theorem which shows an even stronger result. Given two graphs G and G', we define the similarity $s(G, G')$ as

$$\max_{\pi} \frac{2|\{(u,v) \in E \,|(\pi(u), \pi(v)) \in E'\}|}{|E| + |E'|}$$

with respect to all partial one-to-one mappings $\pi : V \to V'$. Note that we do not require three layer graphs here, and accordingly do not have types of nodes.

Theorem 4.1 *Given two bipartite graphs G and G' and a number $s \in [0, 1]$ the problem to decide whether the similarity of G and G' is greater than or equal to s is NP-complete.*

Proof. The NP-complete problem MAXIMUM BALANCED COMPLETE BIPARTITE SUBGRAPH (problem GT24, Garey and Johnson (1987), a proof has been published in Johnson, (1987))) can be reduced to this problem. In this problem we are given a bipartite graph $H = (U, V, E)$ and an integer K, and we ask whether there exists subsets of nodes $U_1 \subset U$, $V_1 \subset V$ that induce a complete bipartite subgraph and for which $|U_1| = |V_1| = K$. Given an instance of this problem we construct an instance of the similarity problem as follows. We let G be a balanced complete bipartite subgraph with $2K$ vertices, and $G' = H$. We set $s = 2K^2/(K^2 + |E'|)$. Then it is obvious that H is a yes-instance of maximum balanced complete bipartite subgraph if and only if the similarity of G and G' is greater or equal to s. □

Note that the problem becomes not easier if we restrict the mapping π between two bipartite graphs $G = (U, V, E)$ and $G' = (U', V', E')$ in the definition of similarity to mappings that map U into U' and V into V', i.e., if we introduce a simple form of types of nodes as we did in the definition of SSM similarity in section 2.2. This because of the symmetry of the graph G in our reduction. From that it follows that checking the similarity of two SSM is also NP-hard, because we have shown that it is NP-hard even if there are only two types of nodes.

Corollary 4.2 *Given two SSM graphs G and G' and a number $s \in [0, 1]$ the problem to decide whether the similarity of G and G' is greater than or equal to s is NP-complete.*

Although the problem of finding a mapping of optimal quality is NP-hard, the special structure of our graphs supports us in designing heuristic and exact methods. This is due to the 3 layer structure of our graphs, and the partition of nodes into types. Let us give the layers the new names U_1, U_2, and U_3, where $U_1 = U_E$, $U_2 = U_P \cup U_V$, $U_3 = U_M \cup U_F$. Given another graph $G' = (U_1', U_2', U_3', A')$ a mapping from a subset of nodes of G to nodes of G', mapping nodes of same type, decomposes into three mappings $\pi_i : U_i \to U_i'$, $i = 1, 2, 3$. Suppose we fix two of these mappings, π_1 and π_2, say, and want to change the third in order to increase the matching quality. How efficient can that be done? The good news are the following

Lemma 4.3 *Given a mapping π between two SSM graphs G and G', decomposing into parts π_1, π_2, and π_3, we can efficiently compute for every $j \in \{1, 2, 3\}$ a mapping with optimal matching quality among all mappings π' with $\pi_k' = \pi_k, k \neq j$. Furthermore for fixed π_2 we can compute a mapping with optimal matching quality among all mappings π' with $\pi_2' = \pi_2$.*

Proof. For the first part of the theorem we observe that we can find an optimal π_j' by solving a weighted bipartite matching problem in an appropriately constructed bipartite graph $H = (V, V', E)$. Indeed, we take $V = U_j$, $V' = U_j'$, $E = \{(v, v') |\ v, v'$ belong to the same type. The

weight of an arc $\{v, v'\}$ is set to the number of edges that are realized if v is mapped to v'. A maximum weighted matching in H thus corresponds to a best π'_j with respect to fixed $\pi'_k = \pi_k, k \neq j$. For the second part of the lemma observe that for layers 1 and 3 the arc weights in our matching graph only depend on how the middle layer is mapped. In both cases the optimal matching can be found in polynomial time (see, e.g., Papadimitriou and Steiglitz (1982)). □

Lemma 4.3 is a good base for exact or heuristic search. For an exact approach we can enumerate all feasible mappings of nodes from the second layer and solve for each of them the matching problems on layers 1 and 3 to optimality. This algorithm is polynomial for a fixed number of nodes on layer 2, and a has a reasonable running time for small numbers of nodes on layer 2.

As a heuristic we can use a local search framework (see, e.g., Papadimitriou and Steiglitz (1982)). A feasible solution is given by a mapping π, given by two partial fixes of π: layer 2, layer 1 and 3. From every feasible solution we can construct two new solutions, by fixing π on layer 2 and on layer 1 and 3, respectively. In both cases a weighted bipartite matching algorithms finds a best solution in an exponentially large neighborhood. We can implement different search strategies on this neighborhood structure, best neighbor, tabu search, or simulated annealing.

Instead of optimizing on the non-fixed layers, and by that generating only two neighbors, one might be less strict. Having fixed π on layer 2, we might just change the mapping π on pairs of nodes on layer 1 and 3. This can be done until no further exchange leads to an improvement. In other words we are using a heuristic to find an approximate solution of the matching problem identified in Lemma 4.3. In our context this might make sense, because we are not necessarily interested in the exact similarity, but a lower bound.

At the time being we have implemented the later version of a local search in our prototype. It first builds the adjacency matrices of the query graph G and the library graph G', thereby extending each graph with dummy nodes (nodes with no edges) to ensure that the matrices have the same

number of nodes for each different category. Then it starts with an arbitrary mapping π of nodes and calculates the number of realized edges. While there is a pair of nodes in G of the same category for which an exchange of images under π increases the number of realized edges, this exchange is done. If no such exchange is found the algorithm reports the reached solution as a local optimum.

5 Filter Algorithms

We have seen in the previous section that the computation of the similarity of SSM is computationally expensive as it requires to solve an NP-hard optimization problem. Even if the SSM graphs are small and exact algorithms terminate in reasonable time for a single comparison, the mass of SSM graphs stored within a library leads to unacceptable response times because the algorithm must compare each library graph with the query graph. This situation is typical for multimedia information retrieval systems: A large amount of library objects and a complex distance function computing the similarity between a query object and the library objects. It is therefore necessary to reduce in a first step the amount of relevant objects with a fast filter. Such filters are proposed in this section.

Applying a filter may lead to two types of faults: objects that are relevant are not returned (*false rejects*), and objects that are not relevant are returned (*false accepts*), Baeza-Yates and Ribeiro-Neto (1999). False rejects are usually not desirable. A filter should therefore be based on a lower-bound of the actual distance between two objects. The user then specifies the maximum distance between the query object and the result objects. Objects are rejected because of a too high lower bound, excluding the possibility of a false reject. A filter with a tight lower bound is able to significantly reduce the amount of possibly relevant library objects to which the computationally expensive algorithm has to be applied.

In our case the "distance" is smallest if the coefficient s is equal to 1. We want to maximize the similarity, therefore the filter has to compute an upper bound on this similarity. Given the formula for s this is the same as computing an upper bound on the number of edges that can be matched by a mapping between the query graph and library graphs.

The structure of the SSM graphs only permits edges between the first and the second, and between the second and the third layer, respectively. Every edge connects nodes of two different types. This defines $O(m^2)$ edge types, where m is the maximum number of different node types on a layer. We expect the number m to be small (in our case m is even only 2) because a large number of node types would make the modeling task too complex. Let t be the number of edge types.

For every graph G we can count the edges of each type and define a vector $v(G) = (v_1, \ldots, v_t)$ with these numbers. Note that different SSM graphs may have the same vector representation. Every mapping between two graphs has to map edges of the same type. Therefore, D(v(G),v(G')) := $\sum_{i=1}^{t} min(v_i, v_i')$ is an upper bound on the number of edges that can be matched, and thus

$$c(G, G') := \frac{2D(v(G), v(G'))}{|E| + |E'|}$$

is an upper bound on $d(G, G')$. Note that for two numbers a and b it is $min(a, b) = a + b - |a - b|$. Therefore we get

$$\sum_{i=1}^{t} min(v_i, v_i') = |E| + |E'| - \sum_{i=1}^{t} |v_i - v_i'|,$$

which relates our distance to the distance of the vectors v and v' with respect to the sum norm.

For each SSM graph stored in the library we generate its edge type vector $v(G)$ once and store it in the library. We do the same for a query graph and then compare its vector with all in the library. Considering the number of edge types t as a constant the filter algorithm can determine $c(G, G')$ in constant time for every pair of graphs. Applied to all library graphs the performance is linear in the number of stored graphs.

This straight forward bound can be further tightened as follows. For every edge type we might look at all nodes of the middle layer which are adjacent to an edge of this type, as well as to their degrees with respect to this edge type. We use this information to compute a better upper bound than $\min(v_i, v_i')$ on how many edges of this type can be matched. We construct two vectors with the degree sequences, $(w_1, w_2, \ldots, w_k)$, $(w_1', \ldots, w_k')$. This means that there are k nodes in G adjacent to edges of this type, and (w.l.o.g.) the same number of nodes in G', with degrees as given by the components of the vectors w and w'. Without loss of generality we may assume that $w_1 \geq w_2 \geq \cdots \geq w_k$.

A mapping of node i of G to node $\pi(i)$ of G' achieves at most $min(w_i, w'_{\pi(i)})$ edges. The following lemma shows how to find a mapping π that maximizes the sum over these values.

Lemma 5.1 *Given two vectors $(w_1, w_2, \ldots, w_k)$ and $(w_1', \ldots, w_k')$, such that $w_1 \geq w_2 \geq \cdots \geq w_k$, the expression $\delta(\pi) := \sum_{i=1}^{k} min(w_i, w'_{\pi(i)})$ is maximized if w_1 is mapped to the largest w_j', w_2 to the second largest, and so on.*

Proof. Indeed suppose there is an index i such that $w'_{\pi(i+1)} > w'_{\pi(i)}$. If $w'_{\pi(i)} \geq w_i$ or $w_{i+1} \geq w'_{\pi(i+1)}$ an exchange of the images of i and $i+1$ would not change $\delta(\pi)$. If $w'_{\pi(i+1)} \geq w_i$ and $w_i > w'_{\pi(i)} \geq w_{i+1}$ an exchange yields $w_i + w_{i+1}$ as contribution from i and $i+1$ in $\delta(\pi)$, which is an improvement compared to $w'_{\pi(i)} + w_{i+1}$. If $w'_{\pi(i+1)} \geq w_i$ and $w_{i+1} \geq w'_{\pi(i)}$, exchanging the two yields $w_i + w'_{\pi(i)}$, which is an improvement compared to $w_{i+1} + w'_{\pi(i)}$. If $w_i > w'_{\pi(i+1)}$ and $w'_{\pi(i)} \geq w_{i+1}$, an exchange yields $w'_{\pi(i+1)} + w_{i+1}$, versus $w'_{\pi(i)} + w_{i+1}$. Finally, if $w_i > w'_{\pi(i+1)} > w_{i+1}$ and $w_{i+1} > w'_{\pi(i)}$, an exchange yields $w'_{\pi(i+1)} + w'_{\pi(i)}$, versus $w_{i+1} + w'_{\pi(i)}$. Thus in every case an exchange of the images of i and $i+1$ can only improve $\delta(\pi)$.

□

By Lemma 5.1 an upper bound on the number of mappings can be computed by sorting

$(w'_1, \ldots, w'_k)$, and then calculate:

$$\delta := \sum_{i=1}^{k} min(w_i, w'_i).$$

The value δ has to be computed for every edge type, and all these values have to be added. This can be done in linear time in terms of the number of nodes on the middle layer, if the number of edge types is considered to be a constant, and if the vectors w are given. Calculating the vectors w will require to count edges and to sort, so essentially $n \; log \; n$, if n is the number of nodes on the middle layer.

Note that instead of using the middle layer nodes one might also use the upper and lower layer nodes for the bound.

6 Computational Results

The retrieval system *Firestorm* uses the algorithms from the previous two sections in the following way. Given a query model G and a similarity level s it searches for all graphs G' whose similarity to G is larger than or equal to s. The filter algorithms can be used to reduce the number of graphs on which computationally more expensive methods have to be applied. The filters used are fast algorithms to compute upper bounds on the similarity. These upper bounds would probably not be very helpful if we want to solve a similarity problem exactly by branch and bound, say. In our context however bounds prove to be very efficient, as they exclude very fast highly dissimilar graphs.

In order to verify this a library of 1000 randomly generated benchmark graphs has been created. The key characteristics of the benchmark graphs (e.g. the distribution of node types) have been determined from an existing AMPL model library (see Müller and Müller, (2000), for details). All benchmarks have been performed on a Sun enterprise 450 with four 250 MHz Ultra-

Sparc II processors and 1 GB of main memory. In our computational tests all 1000 graphs are used as queries to the other 999 graphs, and average quality of the algorithms is measured.

Computational tests that we did in preparation of the tests about which we report in this paper have shown that the second filter is comparable in total running time with the first filter. The reason is that the overall running time is dominated by the time needed to compute the list of output graphs[1]. As the second filter calculates better upper bounds, all our later computational tests have therefore used this second filter.

Section 6.1 investigates the average number of graphs that the filter can exclude, and the running time needed for the filter. Section 6.2 gives our results of the precision of the filter algorithm, which is the number of relevant graphs divided by the number of returned graphs. Section 6.3 shows the overall running time when filter, heuristic, and exact algorithm are combined.

6.1 Effectiveness and Efficiency

For each of the 1000 graphs the *effectiveness* of the filter algorithm is measured by the amount of graphs the filter returns for a required minimum similarity. The measure of computational efficiency is the running time that the filter needs to produce this set of graphs.

Figure 5 contains 2 diagrams. The left diagram shows the average amount of graphs returned by the filter and the standard deviation of this number for different similarity thresholds. The right diagram shows the average running time (in ms) and its standard deviation.

It is obvious that the running time depends mostly on the number of graphs returned. This is due to the fact that for each library graph the necessary data structures can be instantiated at system boot. The time needed to evaluate the filter on a pair of graphs is therefore much less than the time needed to create and sort the result[2].

[1] Graphs are sorted by decreasing similarity

[2] Graphs are sorted by decreasing similarity

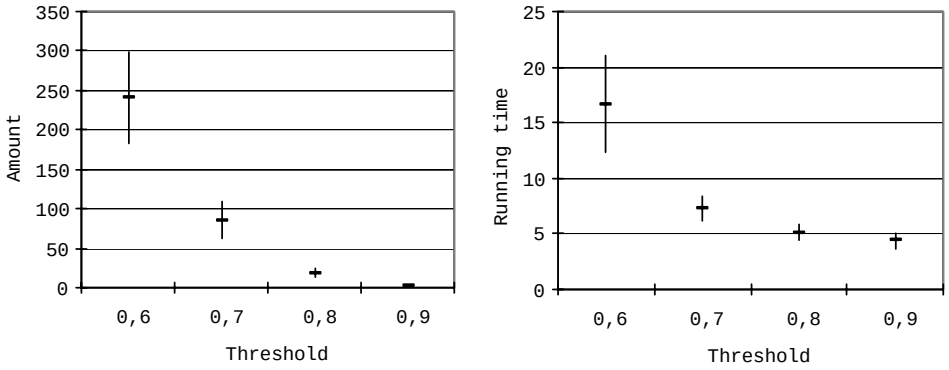

Figure 5: Effectiveness and Efficiency of Filter

6.2 Precision

The precision of a filter algorithm is calculated as follows: Apply the filter algorithm with a chosen minimal value of similarity. This gives the set of graphs for which the filter algorithm cannot exclude that the similarity is higher than the chosen value. Then use the exact similarity algorithm to eliminate those graphs that have a lower similarity. The rest defines the set of relevant graphs. Their amount divided by the number of graphs returned by the filter gives the precision of the filter. The running time of the exact algorithm can be very long. Cases where it did not terminate within 10 seconds were considered as in-tractable, and left out from the sample. As mentioned in Section 5, recall is not an issue because the filter algorithm returns all relevant graphs and therefore the recall is 1. Figure 6 shows the average precision of the filter algorithm and the standard deviation according to various similarity threshold.

With decreasing required similarity the precision generally decreases, until a minimal precision is reached. After that point the precision of the filter increases again because the quotient between irrelevant and relevant graphs decreases rapidly, and thus almost all graphs satisfy the filter as well as have a similarity at least as large as the threshold.

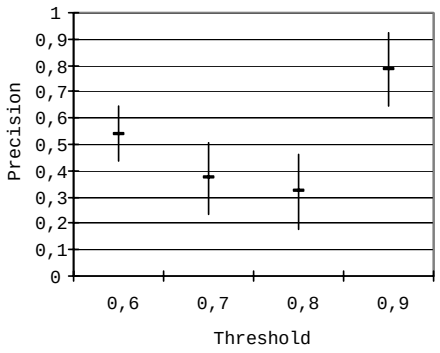

Figure 6: Precision of Filter

When comparing the two filter algorithms we see that filter 2 has always a significantly higher precision than filter 1. Since we saw earlier that it has almost the same running time, we use only this filter in the combination of filter, heuristic and exact algorithm that is presented in the next section.

6.3 Combining Filter, Heuristic, and Exact Algorithm

A user is mostly interested in the overall time the system needs for retrieval. In this section we combine all three algorithms to achieve a good response time for a given similarity thresholds. First, the filter algorithm excludes all graphs with lower similarity. Second, the heuristic algorithm proves the threshold for as much graphs as possible. Third, the exact algorithm is applied to the remaining undecided graphs.

Figure 7 contains 2 diagrams. The left diagram shows the average total running time for various similarity thresholds. Each pillar is divided into segments that show the average fraction of the running time that is needed by the filter, the heuristic, and the exact algorithm, respectively. The right diagram shows the amount of undecided graphs after applying filter and heuristic for different similarity thresholds.

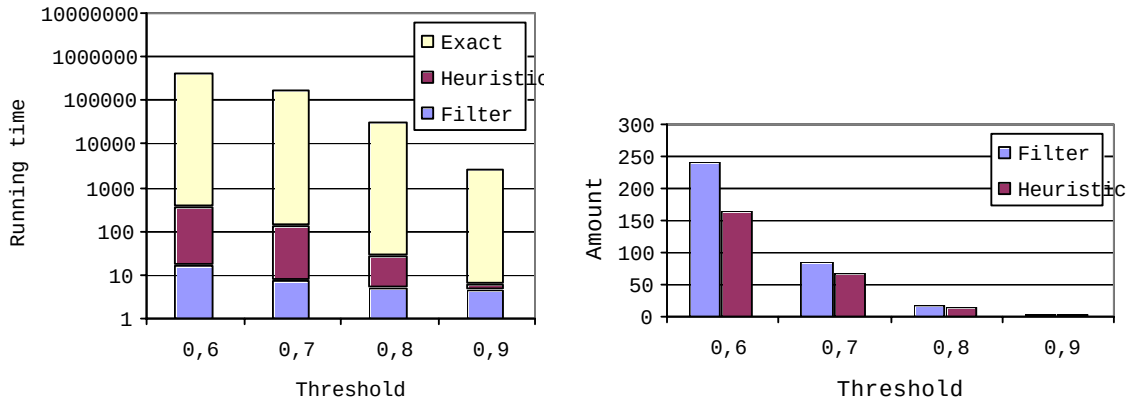

Figure 7: Filter Combination

It is evident that with an increasing similarity threshold the filter algorithm reduces the overall running time dramatically. The logarithmic scaling for the running time visualizes this fact. With a similarity threshold of 0.8 total retrieval time was always between 1 and 30 seconds, which is acceptable for interactive usage.

With higher similarity thresholds the effect of the filter relative to that of the heuristic improves, which means that the heuristic can only slightly reduce further the number of graphs that have to be considered by the exact algorithm. On the other hand the running time of the heuristic decreases as well so that using it as an intermediate step does not negatively influence the overall running time.

We may conclude from this section that the combination of the three algorithms gives precise and complete retrieval results at very reasonable running times.

7 Related work

Searching for information that is hidden in the Deep Web is not a new task. Sites providing information generated by scripts out of data stored in databases exist nearly as long as the Web

exists. There are a number of search engines that index those sites and query them according to some user input. We look at some of those engines and relate them to the structural retrieval approach.

One of the most popular commercial Deep Web search engines is *LexiBot* from BrightPlanet, (2001). It searches among 2200 databases and provides a query interface for simple text or boolean queries. The users can adjust the search strategies and result presentation to their own preferences.

Also from BrightPlanet is the free search engine *CompletePlanet*, BrightPLanet (2000). It contains the addresses of about 103000 databases organized in a directory structure with categories and subcategories. The search interface allows simple text input.

The *LII*, Library of California (2001), is a free search engine with an annotated, searchable subject directory of about 9000 Web resources. The resources are selected and evaluated by librarians for their usefulness to users of public libraries. Simple text and boolean operators are the means to state queries.

The search engines *ProFusion*, IntelliSeek (2001*b*) and *InvisibleWeb*, IntelliSeek (2001*a*), resemble much the other mentioned directory-based search engines.

Neither of the presented search engines use the database structure of Deep Web information systems. They describe the content of those systems with keywords and organize them (by hand or automated) within a directory. Query interfaces require simple text and boolean queries. They are easier to use than our query interface that is based on ER diagrams. But especially the structural dependencies between different information units captures much more semantics and therefore allows a more precise search.

To the best of our knowledge there is no retrieval functionality available that uses the graph structure of decision support models. In most of the algebraic modelling systems models are stored as readable text documents, which makes search for keywords and phrases in these documents

can be applied. On a semantic level models can be classified by classification indices like ACM classification, ACM (2002), or *Guide to Available Mathematical Software*, NIST (2002). These indices classify on the basis of mathematical structure (e.g., flow problem, quadratic optimization, etc.), but not with respect to applications. Retrieval requires therefore very good mathematical knowledge, and can only be done by search through the classification hierarchy. For some domains like queuing and scheduling more detailed classification schemes have been developed. Again they require a precise knowledge of the mathematical theory, and they do not provide a framework for the comparison of models.

Graph structures have however played a prominent role in developing modeling systems for decision support. Besides structured modeling one should mention an implementation of graph based structured modeling, Chari and Sen (1997, 1998), and the graph-grammar based approach by Jones (1990*b*). But these systems do not address the issue of model retrieval in model libraries.

Within the database community there is ongoing work to develop systems for automatic schema matching. Bernstein, Halevy and Pottinger, (2000), have suggested a high level algebra to perform operations like merging schemata. In fact they use the graph structure of schemata together with information about the related data to fulfill the task. An application of this idea for the Web is the system GLUE, Doan et. al (2002). It uses machine learning techniques to automatically create mappings between ontologies built upon RDF descriptions. In an example they map elements of two RDF graphs representing curriculum vitae data from university web sites.

Although our models are not intended to represent RDF graphs, these could be translated into SSM based upon the rules proposed in Müller and Schimkat, (2001). In contrast to GLUE our approach would compare two RDF graphs/schemas only by structure and not use the underlying data to create a mapping between schema elements. The intention of our approach is also different. Retrieval on RDF repositories along the lines from our paper would allow to find most similar RDF

schemata in large repositories. This could either support RDF graph design (similar to what we can support in the context of UML), or it could provide the kernel of an RDF schema based search engine (similar to our application in the context of decision support). While details cannot be given here yet, our contribution to this type of retrieval technology should be clear. We suggest an internal representation, namely SSM, because it provides the basis for robust and efficient retrieval algorithms. Furthermore, as mentioned before, the presented prototype is easy to extend to a new domain like RDF graphs.

8 Conclusions

We have described a new retrieval approach for Web services that is based on structured modeling. The approach has applications for retrieval of online decision support services, Deep Web information and other services that can be represented by directed graphs. The retrieval approach has been implemented in a prototype system and extensively tested. The computational results are satisfactory, and a class room experiment has shown that similarity reported by the retrieval system is related to semantic similarity, at least when the context is restricted.

The algorithmic problem of computing the similarity of two structured service models is shown to be NP-complete. But we also showed how to exploit the structure of the models for fast heuristic and exact algorithms. Furthermore, fast filter algorithms turn out to be very efficient in our benchmark.

Yet, the approach leaves many opportunities for improvements. New algorithmic ideas for heuristic, exact, and filter algorithms can be easily integrated into the retrieval system in order to improve the running time. A finer system of node types, arranged in type hierarchies, can be adapted. Such an extension of the system has been suggested in Müller and Schimkat, (2001). It reduces the running time of the algorithms, and refines the similarity measure. Finally, differences

of node labels in the model can be measured by string comparison algorithms, and used to refine the definition of matching quality, and thus the definition of similarity. These extensions are the topic of forthcoming papers.

References

ACM (2002), 'ACM Computer Classification Systems'. www.acm.org/class (April 2002).

Baeza-Yates, R. & Ribeiro-Neto, B. (1999), *Modern Information Retrieval*, Addison Wesley. ISBN 0-201-39829-X.

Bernstein, P. A., Halevy, A. Y. & Pottinger, R. (2000), 'A vision of management of complex models', *SIGMOD Record* **29**(4), 55–63.
URL: *citeseer.nj.nec.com/bernstein00vision.html*

Bhargava, H., Krishnan, R. & Müller, R. (1997), 'Decision support on demand: Emerging electronic markets for decision technologies', *Decision Support Systems* **19**(3), 193–214.

BrightPlanet (2000), *CompletePlanet*. www.completeplanet.com (April 2002).

BrightPlanet (2001), *LexiBot*. www.lexibot.com (April 2002).

Brooke, A., Kendrick, D., Meeraus, A. & Raman, R. (1998), 'GAMS - A user's guide'. www.gams.com (April 2002).

Chari, K. & Sen, T. K. (1997), 'An integrated modeling system for structured modeling using model graphs', *INFORMS Journal on Computing* **9**(4), 397–416.

Chari, K. & Sen, T. K. (1998), 'An implementation of a graph-based modeling system for structured modeling (GBMS/SM)', *Decision Support Systems* **22**(2), 103–120.

Chen, P. P. S. (1975), 'The entity-relationship model — toward a unified view of data', *Proceedings of the 1th Conference on Very Large Databases, Morgan Kaufman pubs. (Los Altos CA), Kerr (ed), pp.173* .

community, U. (2002), *Universal Description, Discovery and Integration of Business for the Web (UDDI)*. www.uddi.org/.

Decker, S. (2001), *The Semantic Web Community Portal*. www.semanticweb.org.

Doan, A., Madhavan, J., Domingos, P. & Halevy, A. (2002), Learning to map between ontologies on the semantic web, *in* 'Proceedings of the 11th International World Wide Web Conference', pp. 662–673.

Fourer, R., Gay, D. M. & Kernighan, B. W. (1993), *AMPL: A Modeling Language for Mathematical Programming*, Duxbury Press.

Fourer, R. & Goux, J.-P. (2001), 'Optimization as an internet resource', *Interfaces* **31**(2), 130–150.

Garey, M. R. & Johnson, D. S. (1987), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco.

Geoffrion, A. M. (1987), 'An introduction to structured modeling', *Management Science* **33**(5), 547–588.

Geoffrion, A. M. (1989), 'The formal aspects of structured modeling', *Operations Research* **37**(1), 30–51.

IntelliSeek (2001*a*), *InvisibleWeb – The Search Engine of Search Engines*. www.invisibleweb.com (April 2002).

IntelliSeek (2001*b*), *ProFusion*. www.profusion.com (April 2002).

Johnson, D. S. (1987), 'The np-completenes column: an ongoing guide', *Journal of Algorithms* **8**, 438–448.

Jones, C. V. (1990*a*), 'An introduction to graph-based modeling systems, part i: Overview', *ORSA Journal on Computing* **2**(2), 136–151.

Jones, C. V. (1990*b*), 'An introduction to graph-based modeling systems, part i: Overview', *ORSA Journal on Computing* **2**(2), 136–151.

Library of California (2001), *Librarians' Index to the Internet*. www.lii.org (April 2002).

Müller, S. & Müller, R. (2000), Retrieval of service descriptions using structured service models, *in* 'Proceedings of the 10th Annual Workshop on Information Technologies and System', pp. 55–60.

Müller, S. & Schimkat, R. (2001), A general, web-enabled model retrieval approach, *in* 'Proceedings of the International Conference for Object-Oriented Information Systems (OOIS 2001)'.

Müller, S., Schimkat, R.-D. & Müller, R. (2002), Query language for structural retrieval of deep web information, *in* 'Proceedings of the 2nd International Workshop on Web Dynamics'.

NetLib (1995), 'Netlib Repository of mathematical software, papers, and databases'. www.netlib.org/ (April 2002).

NIST (2002), 'GAMS -guide to available mathematical software'. gams.nist.gov (April 2002).

Object Management Group (2000), 'Unified Modeling Language - UML'. Available at http://www.omg.org/technology/documents/formal/uml.htm.

Page, L., Brin, S., Motwani, R. & Winograd, T. (1998), *The PageRank Citation Ranking: Bringing Order to the Web*. citeseer.nj.nec.com/page98pagerank.html (APRIL 2002).

Papadimitriou, C. H. & Steiglitz, K. (1982), *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall.

project, N. (2000), 'Neos - server for optimization'. www-neos.mcs.anl.gov (April 2002).

Ragsdale, C. (2000), *Spreadsheet Modeling and Decision Analysis*, South-Western College Publishing.

Vanderbei, R. (2000), 'Nonlinear Optimization Models – Sample AMPL Examples'. www.princeton.edu/~rvdb/ampl/nlmodels/index.html (April 2002).

World Wide Web Consortium (W3C) (1999), *Resource Description Framework*. www.w3.org/-TR/REC-rdf-syntax/.