Retrieval for decision support resources by structured models

Ulrich Güntzer a, Rudolf Müller b,*, Stefan Müller c, Ralf-Dieter Schimkat d

a Wilhelm-Schickard-Institute for Computer Science, University of Tübingen, Sand 13, 72076 Tübingen, Germany
b Department of Quantitative Economics, Maastricht University; PO. Box 616, 6200 MD Maastricht, The Netherlands
c Research and Development All for One Systemhaus AG, Unixstr. 1, 88436 Oberessendorf, Germany
d Simulation, Analysis and Forecasting, SAF AG, Hightech-Center-2, Bahnstrasse 1, 8274 Tägerwilen, Switzerland

Available online 8 August 2005

Abstract

The number of available DSS within organizational Intranets will soon require efficient retrieval functionality. While Web retrieval technology performs excellent on documents, computational services need approaches that capture the semantics of resources. We present a retrieval approach that uses a variant of Structured Modeling to represent resources. It allows the use of similarity of models for retrieval. Exact similarity computation is shown to be NP-hard, and efficient heuristics for similarity computation and filter algorithms are introduced. We report an evaluation in a classroom experiment and give computational results on a benchmark library.

© 2005 Elsevier B.V. All rights reserved.

Keywords: Decision Support Systems; Information retrieval; Matching algorithms

1. Introduction

In almost every organization, the use of Web technology has changed the way in which data and computational resources are brought to the desktop of the employees. Many of these resources are intended to support their users in decision making. Since Web based solutions are easy to establish [29], the plenitude of decision support resources that can be made available in the Intranet is quickly increasing [4,21]. Full utilization of all available resources will soon require retrieval functionality.

Web retrieval technology has proven to be very successful. The currently leading search engine, Google, is now also offering its technology for use inside organizations. However, if the targets of search are computational resources, like DSS, technology designed for document retrieval is likely to be of limited use. Power and Karparthi [29] distinguish communication-driven, data-driven, document-driven, knowledge-driven, and model-driven DSS. We may assume that only a few communication-driven DSS are available within one organization, thus retrieval is not an issue. We may also assume that document-
driven DSS lend themselves extremely well towards applying Web retrieval, but the categories data-driven, knowledge-driven and model-driven DSS are likely to pose challenges for retrieval because the resources are not documents. One might create text documents that describe the resources, but these documents might use mathematical terms that do not match with the terms describing the application. Existing descriptions of these resources might be given as expressions in some modeling language. For example, a description might be an entity-relationship (ER) diagram (data-driven DSS) [7], an object-oriented design model (knowledge-driven DSS), or an algebraic model (model-driven DSS). We suggest in this paper to use labeled graphs as descriptions, and let these graphs be the basis for retrieval.

In most cases, Web retrieval works as follows. Central is a repository, called search engine, with descriptions of Web documents, for example, collections of phrases found in a document. The repository is filled either by manual registry or by software agents (crawler) who search the web for documents, and create automatically entries in the search engine. Web users are given access to a search engine by a Web-based query interface. The search engine compares user queries with repository content, and computes a result in form of a ranked list of Web resources. For an overview on web retrieval we refer to Kobayashi and Takeda [18].

The search engines proposed in this paper contain labeled graphs as representations of resources. We target on resources which lend themselves towards such representations, for example models from an algebraic modeling language or entity-relationship diagrams. More specifically, our diagrams are labeled, acyclic-directed graphs, consisting of three layers of nodes. We call these graphs Structured Service Models (SSM). In short, the retrieval system provides a user with an interface by which he can create a query graph which is compared with graphs in a repository, resulting in a list of graphs ranked by decreasing similarity. Each graph represents a resource that is available as a service within the Intranet. Other applications, like the set of models in a model library, are possible, but our focus is on online information services. That is why we use the term service instead of resource. Thus the principles are very much like that of other search engines. It is just that our approach requires to compare and rank labeled graphs rather than texts.

2. Bird’s eye view of SSM retrieval

Let us start with a remark. Within the literature on model-driven DSS the term retrieval usually stands for the retrieval of information when working with a particular model on a particular instance of a decision problem [13]. We mean by retrieval the activity to search for an appropriate model within a large model library. To the best of our knowledge, this is a new topic that has not been addressed in this form in the DSS literature. Previous work on large modeling systems focused on issues like model composition and model modification ([22,31]; see also [19]).

2.1. Retrieval architecture

Fig. 1 illustrates our envisioned retrieval architecture. The architecture resembles so-called umbrella architectures (see, e.g., [20]). On the left-hand side the retrieval functionality is sketched. It uses a repository of SSM, and a user interface by which queries to this repository are formulated, executed, and interpreted. On the right-hand side various kinds of resources and services are sketched. They could be collections of models in particular modeling languages, e.g., spreadsheets or AMPL models. The resources could also be knowledge-driven web-based DSS applications, which compute, for example,
exchange rate forecasts or help to produce customer ratings. Or they could provide pure data-driven decision support. Between the repository pillar and the resource pillar there needs to be registration that fills the repository with SSM representing the various resources and services. Three techniques could be applied. The first is crawler software that automatically visits the resources, and creates from each resource an SSM representation. The second is manual entry. The third is a converter program that converts models or other formal resource descriptions into SSM.

Generally, the quality of such a retrieval architecture will depend on the semantic accuracy of the query language and the resource representations used in the repository. Accuracy can be increased if domain knowledge is incorporated. In our approach there is room for this in various ways, indicated by the rectangle below the three pillars. We will get back to this later.

The research presented in this paper focuses on the left and middle pillar of the retrieval architecture. More precisely, we present a prototype for the left column that can also be used for service registration (Section 4). We elaborate in detail on the algorithmic aspects of this prototype (Section 5). For a specific library, we have implemented a converter (Section 7).

2.2. Retrieval objects and algorithms

SSM are a variant of Structured Models [11], a modeling approach for decision support that made a significant contribution to the theory of modeling in Management Science. Broadly speaking 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 capture the essential characteristics of a model in the field of Management Science [15].

Our retrieval algorithms, i.e., the comparison of queries with stored service models and the ranking of those, use algorithmic graph theory. 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. It is shown that computing the similarity exactly is NP-complete, but the structure of models supports the design of heuristic as well as exact algorithms (Section 5). Computationally very efficient lower bounds for similarity can be used as a filter (Section 6).

We developed our approach initially for model libraries. For such libraries we give computational results in Section 7. In order to evaluate the usefulness for further application domains, we show in Section 8 how to extend the approach to entity-relationship diagrams. With respect to this domain, Section 8.2 presents stimulating results from a small classroom experiment.

A final remark should be made before we elaborate on the details of our approach. Our Structured Service Models are a rather restricted version of Structured Models (actually, in some details they extend Structured Models, but these are minor features introduced for technical reasons). Restrictions were necessary in order to enable efficient comparison algorithms. Such restrictions could come at the cost of semantic accuracy. Our approach addresses this limitation by providing sufficient flexibility in the system design, in order to allow for future extensions. Thus our results for an “educated guess” of appropriate restrictions of general Structured Modeling remain valid when the expressiveness is extended, as long as the underlying models stay within a reasonable complexity.

In general terms, our approach is supported by findings in Jones [17] that a modeling language can be fundamentally graphical, meaning that the modeling language provides attributed graphs as only modeling features (see also [13] for a discussion of this issue).

3. 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 [11,12]. Originally designed to define decision models in Management Science, Structured Modeling has shown its potential in representing models from other fields as well, e.g., database models [5,6].
3.1. Definition of Structured Service Models

A SSM is an acyclic-directed 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 represents 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 languages, e.g. AMPL (see Section 7).

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 \(w\) represents 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, and (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 latter would also allow, e.g., dependencies between functions. As we will see in Section 5 the restriction supports the computation of similarity of models. (It does not restrict too much the expansiveness 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. Thus more freedom in modularization could even negatively influence the quality of retrieval.)

Fig. 2 shows the SSM for the Hitchcock–Koopman transportation model. 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 a 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.

As a more general example consider an optimization problem given by

\[
\max f(x) \\
\text{s.t. } g_i(x) \leq a_i \quad i = 1, \ldots, n
\]

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 attribute node and test nodes are included if an attribute contributes to the definition of a constraint. 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, SSM are therefore able to represent essential parts of the structure of the problem.

The task of SSM in a retrieval context is twofold. Firstly, they can be used as a graphical description of a service that helps a user to decide whether the service meets his requirements. Secondly, 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, given, for example, in an algebraic modeling language. We have implemented a converter for AMPL models (Section 7).

Given a repository of service models, a user submits a query to the retrieval system by creating his own SSM, describing the service he is looking for. The retrieval mechanism returns those SSM that are “close” to the query model. This approach raises the following research questions:

1. How precise can a SSM describe the semantics of a service and a user’s request?
2. What are appropriate measurements of similarity of a query and a repository entry?
3. How efficient can we do retrieval on SSM repositories with respect to a method of measurement?
4. Will retrieval based on searching for similar models provide appropriate precision and recall?

The first question is to some extent answered in Jones [17], and will be addressed in Section 8.2 for another domain of applications. We will give experimental results with students when applying SSM to entity-relationship diagrams. With respect to the second question we make an educated guess and use (structural) similarity of graphs as method of measurement. Certainly comparing graphs purely with respect to their structure is limited. However an initial choice is necessary in order to design the algorithmics and to build the system that provides the framework to experiment with other measurements in follow-up studies. In Section 3.2 we argue furthermore that our way to define similarity can easily incorporate information that is attached to nodes and edges of a graph. Questions of efficiency, precision and recall are the topics of Sections 5, 6, and 7, and form the main contributions of the present paper.

3.2. 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 $\pi$ of the nodes from $G$ to the nodes of $G'$ which only map nodes on nodes of the same type, and define $q(\pi) = |\{(u, v) \in E | (\pi(u), \pi(v)) \in E'\}|$. The function $q$ expresses the number of edges in $G$ that are implicitly mapped on 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 $\pi$ is defined as

$$d(\pi) = 2q(|E| + |E'|).$$

The similarity between two graphs is then defined by the maximum overall 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 septuple $G = (U_E, U_P, U_V, U_T, U_M, U_F, E)$, 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 $\pi$ 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 layers 1, 2, and 3 into types. For example, we might want to distinguish between integer and real valued decision variables, or we might take the dimension of the variable into account (e.g., whether it measures cost, distance, weight, or volume). The finer the partition, the better the semantics of the service is captured, the less freedom is given to mapping \( \pi \), and thus the more meaningful a high similarity. However, also the more the system depends on an agreed ontology of types.

Another important remark concerns labels of variables of the same type. Note that our similarity measure can in principle incorporate textual information, rather than being solely based on SSM structure. While we allow such labels as part of a model, up until now we do not take them into account when we compute the similarity. However, it is easy to see that most of the algorithmic results remain valid when the pure structural approach is extended. Indeed, we can change \( d(\pi) \) in the following way. Let \( \gamma(v, v') \) be a function that measures the distance between the node labels of \( v \) and \( v' \). Then define

\[
d'(\pi) = \frac{2 \left( q + \sum_{v \in V} \gamma(v, \pi(v)) \right)}{|E| + |E'| + |V| + |V'|}.
\]

Before we elaborate on the algorithmics of our retrieval system, we present in Section 4 the prototype implementation. This should give the reader a fair understanding of the context in which the algorithms are used.

## 4. The Firestorm system

In this section we describe a prototype implementation of a search engine based on SSM. The purpose of the prototype is a platform to validate and improve the choices made. On one hand validation requires benchmarking the algorithms that will be described in Sections 5 and 6, on the other hand it requires letting 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. We gave the prototype the name Firestorm (FIrst a REtrieval SysTem for Operations Research Models).

We start with an illustration of the user interface. After the user has loaded the Java applet, a Firestorm frame opens (left part of Fig. 3). Within this frame the user edits a SSM. The SSM 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 repository. The client provides all functionalities to insert, delete, and move nodes and edges. A pop-up 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 Fig. 3). It provides the interface to activate filter algorithms (see Section 6) and graph similarity algorithms (see Section 5), and to choose the number of models to be returned by these algorithms. The Similarity Checker also provides means for browsing through the retrieval results and to view additional information (e.g. the source model, if the SSM was automatically generated from, say, an AMPL model).

The retrieval server encapsulates the retrieval algorithms. Retrieval is done in main memory, all models are loaded in the beginning of a user’s session. Activating filter and retrieval algorithms reduces this list step by step. Available are the sum norm filter described in Section 6, 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 5).

A more detailed description of the software engineering aspects of the system can be found in Müller and Schimkat [24]. Let us mention that those parts of the user interface that are responsible for model editing and visualization have been separated from the rest of the prototype along a multi-tier architecture. When a new domain of models, like ER diagrams (see Section 8.1), asks for a different kind of visualization it suffices to replace this part of the implementation. Communication with the server takes place based on SSM objects. This makes the server with its reposi-
tories and algorithms a general purpose device which can easily be extended to new domains.

Let us conclude this section with a remark on customization of the system. Within a particular domain one can imagine that certain model features are always expressed by the same structural patterns. Each of the registration mechanisms can consider such domain knowledge: crawlers could use specific name conventions to translate names inside models into repository names and use a pre-compiled library of patterns. User interfaces used by providers to edit models could encourage using predefined names and patterns as well. Similarly, converters could be parameterized by domain knowledge. We will see in Section 5 that computational efficiency is also increased by adding domain knowledge, because a fine-tuned system of types of model elements reduces the level of freedom in matching queries with repository graphs.

5. Similarity algorithms

In Section 3 we introduced SSM as an approach to model decision support resources. Section 4 described a Web-based repository for resource 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 it is possible to compute in polynomial time the similarity between two SSM. The answer is unfortunately no, unless $P=NP$. This follows from the following theorem which shows a somewhat stronger result. Given two graphs $G$ and $G'$, we define the similarity $s(G, G')$ as

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

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

**Theorem 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 sub-graph (problem GT24, [10], a proof has been published in Johnson [14]) 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 exist subsets of nodes \( U_1 \subseteq U, V_1 \subseteq V \) 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 does not become easier if we restrict the mapping \( \pi \) between two bipartite graphs \( G = (U, V, E) \) and \( G' = (U', V', E') \) in the definition of similarity to mappings that map \( U \) to \( U' \) and \( V \) to \( V' \), i.e., if we introduce types of nodes as we did in the definition of SSM similarity in Section 3.2. This follows from the symmetry of the graph \( G \) used in the proof. Thus 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 1.** 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 is helpful for the design of heuristic and exact methods. The reason is the 3 layer structure of our graphs, as well as the partition of nodes into types. Let us give the layers the new names \( U_1, U_2, \) and \( U_3 \), where \( U_1 = U_{1E}, U_2 = U_P \cup U_N, U_3 = U_M \cup U_L \). Given another graph \( G' = (U'_1, U'_2, U'_3, E') \) a mapping from a subset of nodes of \( G \) on nodes of \( G' \), mapping nodes of same type, decomposes into three mappings \( \pi_i: U_i \rightarrow U'_i, i = 1, 2, 3 \). Suppose we fix two of these mappings, say \( \pi_1 \) and \( \pi_2 \), and want to change the third in order to increase the matching quality. How efficient can this be done? The good news is the following:

**Lemma 1.** Given a mapping \( \pi \) between two SSM graphs \( G \) and \( G' \), decomposing into parts \( \pi_1, \pi_2, \) and \( \pi_3 \), we can efficiently compute for every \( j \in \{1, 2, 3\} \) \( g \) a mapping with optimal matching quality among all mappings \( \pi' \) with \( \pi'_k = \pi_k, k \neq j \). Furthermore for fixed \( \pi_2 \) we can compute a mapping with optimal matching quality among all mappings \( \pi' \) with \( \pi'_2 = \pi_2 \).

**Proof.** For the first part of the theorem we observe that we can find an optimal \( \pi'_2 \) by solving a weighted bipartite matching problem in an appropriately constructed bipartite graph \( H = (V, V', E) \). Indeed, take \( V = U_j, V' = U'_j, E = \{(v, v') \mid v, v' \text{ are of the same type}\} \). The weight of an edge \( (v, v') \) is set to the number of edges that are realized if \( v \) is mapped to \( v' \). Thus a maximum weighted matching in \( H \) corresponds to a best \( \pi'_2 \) 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 the constructed matching graph problem depend only on how the middle layer is mapped. In both cases the optimal matching can be found in polynomial time (see, e.g., [28]). □

Lemma 1 provides 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 has a reasonable running time for small numbers of nodes on layer 2.

As a heuristic we can use a local search framework (see again [28]). A feasible solution is given by a mapping \( \pi \). From every feasible solution we can construct two new solutions, by fixing \( \pi \) on layer 2 and on layers 1 and 3, respectively, and optimizing according to Lemma 1 on the other layer(s). Note that in both cases we find a best solution in the exponentially large 2-exchange neighborhood of the current solution (see below). We can implement different search strategies, e.g., best neighbor, tabu search, or simulated annealing.

Instead of optimizing on the non-fixed layers, and by that considering only two neighbors, we can also work with small improvements with the help of simple exchanges. Having fixed \( \pi \) on layer 2, we change the mapping \( \pi \) on pairs of nodes on layer 1 and 3. This is done until no further exchange leads to an improvement. In other words we are using a heuristic to find a good solution of the matching problem identified in Lemma 1. In our context this makes sense, because we are not necessarily interested in
6. Filter algorithms

We have seen in the previous section that the computation of the similarity of SSM is computationally expensive as it demands to solve an NP-hard optimization problem. Even if the SSM graphs are small and algorithms terminate in a 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 distance between a query object and the library objects. It is therefore necessary to reduce in advance the number of relevant objects with a filter. This section develops such filters.

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) [2]. False rejects are usually not desirable. Assume that a user specifies a maximal accepted distance between the query object and the result objects. A filter should then be based on a lower bound of the actual distance between two objects. Objects are rejected because of a too high lower bound, excluding the possibility of a false reject.

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 a library graph.

We start by looking at the number of edges of each type. 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 $\alpha(G) = (\alpha_1, \ldots, \alpha_t)$ with these quantities. 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, $D(\alpha(G), \alpha(G')) : = \sum_{i=1}^{t} \min(\alpha_i, \alpha'_i)$ is an upper bound on the number of edges that can be matched, and thus

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

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

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

which relates $D$ to the distance of the vectors $\alpha$ and $\alpha'$ with respect to the sum norm.

The filter works then as follows. For each SSM graph stored in the library we generate its edge type vector $\alpha(G)$ once and store it in the library. Given a query graph $G'$ we compare its vector $\alpha(G')$ 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 straightforward bound can be further tightened as follows. For every edge type $j$ we look at all nodes of the middle layer which are adjacent to an
edge of type \( j \), as well as to their degrees with respect to edge type \( j \). Based on these degrees we find a better bound than \( \min(\alpha_j, \alpha_j') \) for the edges of type \( j \) that can be matched. Let these degree sequences be \( \beta = (\beta_1, \beta_2, \ldots, \beta_k) \) and \( \beta' = (\beta'_1, \ldots, \beta'_k) \). 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 \( \beta \) and \( \beta' \). Without loss of generality we may assume that \( \beta_1 \geq \beta_2 \geq \ldots \geq \beta_k \).

A mapping of node \( i \) of \( G \) to node \( \pi(i) \) of \( G' \) achieves at most \( \min(\beta_i, \beta'_{\pi(i)}) \) edges. Thus

\[
\sum_{i=1}^{k} \min(\beta_i, \beta'_{\pi(i)}),
\]

is an upper bound on the number of edges of type \( j \) that can be realized. The following lemma shows how to find a mapping \( \pi \) that maximizes this expression.

**Lemma 2.** Given two vectors \((\beta_1, \beta_2, \ldots, \beta_k)\) and \((\beta'_1, \ldots, \beta'_k)\), such that \( \beta_1 \geq \beta_2 \geq \ldots \geq \beta_k \), the expression \( \delta(\pi) = \sum_{i=1}^{k} \min(\beta_i, \beta'_{\pi(i)}) \) is maximized if \( \beta_1 \) is mapped to the largest \( \beta'_1 \), \( \beta_2 \) to the second largest, and so on.

**Proof.** Indeed suppose there is an index \( i \) such that \( \beta'_{\pi(i)} > \beta'_{\pi(i)} \). We show that in all of the following cases an exchange of the images of \( i \) and \( i+1 \) can only improve \( \delta(\pi) \). If \( \beta'_{\pi(i)} \geq \beta_i \) and \( \beta'_{\pi(i)} \geq \beta'_{\pi(i+1)} \) an exchange yields \( \beta_1 + \beta'_{\pi(i+1)} \) contribution from \( i \) and \( i+1 \) in \( \delta(\pi) \), which is an improvement compared to \( \beta_1 + \beta_{\pi(i+1)} \). If \( \beta_{\pi(i+1)} \geq \beta_i \) and \( \beta'_{\pi(i)} \geq \beta_{\pi(i)} \), exchanging the two yields \( \beta_i + \beta_{\pi(i)} \), which is an improvement compared to \( \beta_1 + \beta_{\pi(i)} \). If \( \beta_{\pi(i+1)} \geq \beta_{\pi(i)} \) and \( \beta'_{\pi(i)} \geq \beta'_{\pi(i+1)} \), an exchange yields \( \beta'_{\pi(i+1)} + \beta_{\pi(i)} \), versus \( \beta'_{\pi(i)} + \beta_{\pi(i+1)} \). Finally, if \( \beta_i > \beta'_{\pi(i)} > \beta_{\pi(i)} + \beta'_{\pi(i)} \), an exchange yields \( \beta'_{\pi(i)} + \beta_{\pi(i)} \), versus \( \beta_{\pi(i)} + \beta_{\pi(i+1)} \). \( \square \)

By Lemma 2 an upper bound on the number of mappings can be computed by sorting \((\beta'_1, \ldots, \beta'_k)\), and then calculating:

\[
\delta = \sum_{i=1}^{k} \min(\beta_i, \beta'_i).
\]

The value \( \delta \) 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 \( \beta \) are given. Calculating the vectors \( \beta \) will require counting edges and to sort, so essentially \( n \log n \) operations, if \( n \) is the number of nodes on the middle layer.

Note that instead of the middle layer nodes one might also use the upper and lower layer nodes for this filter, and thus, as a third filter, the minimum of the two other filters.

### 7. Computational results

The retrieval system *Firestorm* uses the algorithms from the previous two sections in the following way. Given a query graph \( G \) 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 are used to reduce the number of graphs to which computationally more expensive methods have to be applied.

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 [23] for details). All benchmarks have been performed on a Sun enterprise 450 with four 250 MHz UltraSparc 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 the average quality of the algorithms is measured.

Earlier computational tests have shown that the second filter is comparable in total running time with the first filter. The reason is that the overall running time of a query to the system is dominated by the time needed to compute and sort the list of output graphs. As the second filter calculates better upper bounds, computational tests presented here use only this second filter.

Section 7.1 investigates the average number of graphs that the filter can exclude, and the running time needed for the filter. Section 7.2 gives results of the precision of the filter algorithm, which is the number of relevant graphs divided by the number of returned graphs. Section 7.3 shows the overall running time when filter, heuristic, and exact algorithm are combined.
7.1. 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.

Fig. 4 contains two 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 because 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 (graphs are sorted by decreasing similarity).

7.2. Precision

The precision of a filter is calculated as follows. We apply the filter 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 we use the exact similarity algorithm to find those graphs that have a lower similarity. The remaining set are the relevant graphs. Its size 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 s were considered as intractable, and left out from the sample. Fig. 5 shows the average precision of the filter algorithm and the standard deviation according to various similarity thresholds.

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 having a similarity at least as big as the threshold.

When comparing the two filter algorithms in preparing experiments, filter 2 had always a significantly higher precision than filter 1. This provided
another reason to present here only results from filter 2.

As mentioned in Section 6, recall is not an issue because the filter algorithm returns all relevant graphs and therefore the recall is equal to 1.

7.3. 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 threshold. Firstly, the filter algorithm excludes all graphs with lower similarity. Secondly, the heuristic algorithm proves the threshold for as many graphs as possible. Thirdly, the exact algorithm is applied to the remaining undecided graphs.

The left diagram in Fig. 6 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.

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. For a similarity threshold of 0.8 the total retrieval time was always between 1 and 30 s which is acceptable for interactive usage.

With higher similarity thresholds the impact of the filter relative to that of the heuristic improves, which means that the heuristic only slightly decreases 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 reasonable running times.

8. Extensions to other domains

So far we explained and evaluated our approach when applied to model libraries. However, applications to other decision support resources are possible. The Web contains a lot of information that is not encoded as static HTML pages [20]. Rather, these pages are dynamically constructed based on user input, e.g. travel destinations, or product attributes. The set of those pages is called the deep Web. Within cooperations, deep Web technology is typically used to provide access to databases, in other words, data-driven DSS.

Usually a relational database is used to store the raw data of such a deep Web application, and entity-relationship diagrams can be used to accurately describe the content and relations of the database. Next we describe how SSM can be used for repositories of entity-relationship diagrams.
8.1. Extension to ER diagrams

We define a one-to-one mapping between ER diagrams and SSMs following the rules in Müller and Schimkat [24]: Different kinds of entities become different kinds 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. Fig. 7 shows a simple ER diagram and its corresponding SSM. The mapping rules enforce that SSMs of ER diagrams are again three-layered graphs. As for the domain of decision models, each layer consists of different types of nodes, although their semantics is now quite different. These types are used to restrict the domain of mappings \( \pi \) used in the definition of similarity of SSM.

8.2. Classroom experiment

We did a small experiment in order to evaluate how well our SSM similarity measure coincides with semantic similarity in the context of ER diagrams. In order to apply the system to ER diagrams, we implemented the one-to-one mapping of ER diagram to SSM as described in Section 8.1, and a dedicated user interface.

In a database course at the University of Tübingen (Germany), students have to design information systems with the help of ER diagrams. Firestorm is used to create and manage the ER diagrams. During the course students have to design ER diagrams for three different real world cases. The case descriptions imply the naming of entities and relationships so that we can concentrate on the structure. Staff members evaluate the solutions of the students and assign credit points. For each exercise a staff member creates a master model representing the expected solution. Generally, closeness of a student model with the master model is honored by a high grade, as it expresses that the student model matches really well with the real world case.

During such a class we performed the following evaluation. We used the master model as a query to a repository with the student models forming 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. A high coincidence of the system credit points with the credit points given by the staff member would show that the similarity computed by the system is close to the similarity observed by the staff member.

The results are encouraging. Fig. 8 relates the credit points assigned by the staff members to the credit points assigned by Firestorm. The differences in credit point assignment are marginal.

8.3. Other applications

Let us briefly mention two other applications for SSM based retrieval. The World Wide Web consor-
tion (W3C) has taken efforts for semantic-based Web retrieval by establishing a so-called semantic Web. The efforts rely partly on graphical descriptions of web services by so-called RDF graphs [8]. A mapping to RDF graphs can be expected to be as simple as for ER diagrams [25].

In Müller and Schimkat [24] it is shown that our retrieval approach has applications in software retrieval, too. 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 [27]), can easily be defined. UML models can in turn be useful for describing knowledge-driven DSS.

9. Related work

To the best of our knowledge there does not exist any retrieval system which can search repositories of decision support resources with the help of graph-based descriptions. Typically, algebraic modeling systems store their models as readable text documents, which limits retrieval to text retrieval. In order to ease navigation, models can be classified by classification indices like ACM classification (ACM [1]), or Guide to Available Mathematical Software (NIST [26]). Such indices classify based on mathematical characteristics (e.g., flow problem, quadratic optimization problem, etc.), but not with respect to applications. For domains like queuing and scheduling extremely detailed classification schemes have been developed. Because of that, search for an appropriate model requires quite some expert knowledge. Our approach compares the graphical representation of decision support resources with the graphical representation of a user’s decision problem, creating thereby a completely new type of search paradigm.

Graph structures have played a prominent role in developing modeling systems for decision support. We mention an implementation of graph-based Structured Modeling [5,6], and the graph–grammar-based approach by Jones [15,16]. Although these systems do not address the issue of model retrieval in model libraries, they support our claim that graph-based structures can capture essential parts of the semantics of a model. Their existence proves also that models in algebraic modeling languages are convertible into graph-based models.

Within the database community researchers have developed systems for automatic schema matching. Bernstein et al. [3] suggest a high-level algebra to perform operations like merging schemata. They use the graph structure of schemata together with information about the related data. The system GLUE [9] extends this idea to the Web. GLUE uses machine learning techniques to automatically create mappings between ontologies built upon RDF descriptions. Doan et al. [9] show, for example, how to map elements of two RDF graphs representing curriculum vitae data from university Web sites. As mentioned before RDF graphs can be translated into SSM by a mapping proposed in Müller and Schimkat [24], which makes the two approaches related. However, the goal of GLUE and similar systems is to find similarities between pairs of (large) graphs, while our system searches in large sets of (small) graphs for those who are similar to a query graph.

In the area of education sciences, Waugh et al. [30] have suggested a tool for automated assessment of
entity-relationship diagrams. Their approach is based on measuring both a correct model and a student's model in terms of the presence of a collection of substructures and their connectivity, and then comparing the score between both models. Since their system does not have to scale for large collections of models, more sophisticated reasoning about similarity can be applied. Despite the positive results reported in Section 8.2, our approach would not be as useful for automatic assessment.

10. Conclusions

We have described a new retrieval approach for decision support resources that is based on Structured Modeling. The retrieval approach has been implemented in a prototype system and extensively tested. The computational results are satisfactory, and a classroom 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. However, the special structure of the models can be exploited to achieve fast heuristic and exact algorithms. Furthermore, filter algorithms turn out to be very efficient in excluding many repository entries from consideration.

The system architecture is modularized in order to allow for algorithmic improvements. The similarity measure can be refined by considering the textual similarity of node labels and edge labels. A more fine-grained system of node types, arranged in type hierarchies, can be adapted. In Müller and Schimkat [24] it is shown how this improves the computational performance.

These and other improvements will be necessary if one would like to use the system for retrieval on the Web, for example if the goal is retrieval on repositories of online decision support resources. The present prototype scales only up to a size of several thousand models in the library. Extending the domain of applications from an Intranet to the Web will also require agreements on node attributes, because an approach based completely on structure would likely result in too many similar models. Still, for a limited number of models, and within a domain, where agreements on names are possible, our graph-based retrieval has proven to perform well. We see it therefore as an important contribution towards retrieval systems that use structural rather than purely textual information.

Acknowledgements

We would like to thank three anonymous referees for their comments that considerably improved the presentation and the positioning of our work. Research of the second author has been supported by the Deutsche Forschungsgemeinschaft (Sonderforschungsbereich 373) and the International Institute of Infonomics. Research of the third author has been supported by DFN (Project Matse).

References


