Top-*k* Dominant Web Services Under Multi-Criteria Matching

Dimitrios Skoutas[1,2] Dimitris Sacharidis[1] Alkis Simitsis[3] Verena Kantere[4] Timos Sellis[2,1]

[1]National Technical University of Athens, Greece
{dskoutas,dsachar}@dblab.ntua.gr
[2]Institute for the Management of Information Systems, R.C. "Athena", Greece
timos@imis.athena-innovation.gr
[3]HP Labs, Palo Alto, USA
alkis@hp.com
[4]Ecole Polytechnique Fédérale de Lausanne, Switzerland
verena.kantere@epfl.ch

ABSTRACT

As we move from a Web of data to a Web of services, enhancing the capabilities of the current Web search engines with effective and efficient techniques for Web services retrieval and selection becomes an important issue. Traditionally, the relevance of a Web service advertisement to a service request is determined by computing an overall score that aggregates individual matching scores among the various parameters in their descriptions. Two drawbacks characterize such approaches. First, there is no single matching criterion that is optimal for determining the similarity between parameters. Instead, there are numerous approaches ranging from using Information Retrieval similarity metrics up to semantic logic-based inference rules. Second, the reduction of individual scores to an overall similarity leads to significant information loss. Since there is no consensus on how to weight these scores, existing methods are typically pessimistic, adopting a worst-case scenario. As a consequence, several services, e.g., those having a single unrelated parameter, can be excluded from the result set, even though they are potentially good alternatives. In this work, we present a methodology that overcomes both deficiencies. Given a request, we introduce an objective measure that assigns a dominance score to each advertised Web service. This score takes into consideration all the available criteria for each parameter in the request. We investigate three distinct definitions of dominance score, and we devise efficient algorithms that retrieve the top-k most dominant Web services in each case. Extensive experimental evaluation on real requests and relevance sets, as well as on synthetically generated scenarios, demonstrates both the effectiveness of the proposed technique and the efficiency of the algorithms.

EDBT 2009, March 24–26, 2009, Saint Petersburg, Russia.

1. INTRODUCTION

Web services are software entities that are accessible over the Web and are designed to perform a specific task, which essentially comprises either returning some information to the user (e.g., a weather forecast service or a news service) or altering the world state (e.g., an on-line shopping service or an on-line booking service). Web services, in the traditional sense, are described by a well-defined interface, which provides the number, the names, and the types of the service input and output parameters, and is expressed in a standardized language (WSDL). They constitute a key technology for realizing Service Oriented Architectures, enabling loose coupling and interoperability among heterogeneous systems and platforms. By standardizing the interfaces and the exchanged messages between communicating systems, they can significantly reduce the development and, even more importantly, the maintenance cost for large-scale, distributed, heterogeneous applications.

Typical application scenarios for Web services cover a broad area of software engineering [2]. In a broader sense, any dynamic Web site can be thought of as a (collection of) Web service(s), and thus, Web services provide a means for querying the hidden Web. In addition, Web services are often used as wrappers for databases allowing data access through firewalls, hiding details regarding the underlying data and enforcing data access policies. Another use of Web services is met in mashups applications; e.g., DAMIA [38] and Yahoo Pipes[1]. These constitute a recently emerging trend, where users select and combine building blocks (essentially, services) to create applications that integrate information from several Web sources. Consequently, it becomes apparent that the Web services paradigm rapidly gains popularity constituting an integral part of many "hot" real-world applications. For these reasons, several techniques for retrieving and ranking Web services have been recently proposed.

Consider the following typical Web service discovery scenario. The user provides a complete definition of the *desired* service and poses a query on a system maintaining a repository of *advertised* service descriptions. Alternatively, the user could specify a desirable service, e.g., among previous results, and request similar services. Then, the search engine employs a *matchmaking* algorithm identifying advertisements relevant to the user's request. A lot of recent work has focused on defining objectively good similar-

[1]`http://pipes.yahoo.com/pipes/`

ity measures capturing the *degree of match* between a requested and an advertised service. Typically, this process involves two steps: (i) selecting a criterion for assessing the similarity of service parameters, and (ii) aggregating individual parameter scores to obtain the overall degree of match between a request and an advertisement.

The first step involves the estimation of the degree of match between parameters of the request and the advertisement. There are two paradigms for assessing the match among parameters. The first, treats parameter descriptions as documents and employs basic Information Retrieval techniques to extract keywords; e.g., [16]. Subsequently, a string similarity measure is used to compute the degree of match. The second paradigm follows the Semantic Web vision. Services are enriched by annotating their parameters with semantic concepts taken from domain ontologies; e.g., [32, 28]. Then, estimating the degree of parameter match reduces to a problem of logic inference: a reasoner is employed to check for equivalence or subsumption relationships between concepts.

Both paradigms share their weaknesses and strengths. Regarding the former techniques, keyword-based matchmaking fails to properly identify and extract semantics since service descriptions are essentially very short documents with few terms. On the other hand, the latter techniques face common Semantic Web obstacles, e.g., the lack of available ontologies, the difficulty in achieving consensus among a large number of involved parties, and the considerable overhead in developing, maintaining an ontology and semantically annotating the available data and services. More recently, hybrid techniques for estimating the degree of parameter match have appeared, e.g., [23], taking into account both paradigms. Still, the common issue with all approaches is that there is no *single* matching criterion that optimally determines the similarity between parameters. In fact, different similarity measures may be more suitable, depending on the particular domain or the particular pair of request and offer. Therefore, we advocate an approach that simultaneously employs *multiple matching criteria*.

The second step in matchmaking deals with the computation of the *overall* degree of match for a pair of requested and advertised services taking into consideration the individual scores of corresponding parameters. Various approaches for aggregating parameter match scores exist. One direction is to assign weights, determined through user feedback, to individual scores [16]. Appropriate weights are chosen either by assuming a-priori knowledge about the user's preferences or by applying expensive machine learning techniques. Both alternatives face serious drawbacks and raise a series of other issues to be solved. More often, methods are pessimistic adopting a worst-case scenario. The overall service similarity is derived from the worst degree of match among parameters. However, this leads to information loss that significantly affects the retrieved results accuracy (see Section 5). For example, services having only one bad matching parameter may be excluded from the result set, even though they are potentially good alternatives.

A significant, yet often neglected, aspect of the matchmaking process is the *ranking* of the advertised services based on their degree of match. It is important that useful results appear high on the list. A recent survey [20] shows that users view the top-1 search result in about 80% of the queries, whereas results ranked below 3 were viewed in less than 50% of the queries. In addition, Web service discovery plays an important role in fully automated scenarios, where a software agent, e.g., for travel planning, acting on behalf of a human user, automatically selects and composes services to achieve a specific task. Typically, the agent will only try a few top ranked results ignoring the rest.

The following example portrays a typical discovery scenario, showing the challenges in identifying the top matching services.

Example. Consider a user searching for a Web service providing weather information for a specific location. For simplicity, we assume only one input P_{in} and one output P_{out} parameter. There are four available services, A, B, C, D. Furthermore, three different matching filters (e.g., different string similarity measures), $f_{m_1}, f_{m_2}, f_{m_3}$, have been applied resulting in the degrees of match shown in Table 1. Observe that under any criterion, service A constitutes a better match with respect to both parameters, than any other service. However, there is no clear winner among the other three services. For instance, consider services B and D. If the first matching criterion is the one that more closely reflects the actual relevance of B to the given request, then B is definitely a better match than D. On the other hand, if the second measure is chosen, then B has a lower match degree for the input parameter but a higher degree for the output. Even for such a simple scenario, specifying an appropriate ranking for the candidate matches is not straightforward.

Table 1: Example services and matching criteria scores

Service	Parameter	f_{m_1}	f_{m_2}	f_{m_3}
A	P_{in}	0.96	1.00	0.92
	P_{out}	0.92	0.96	1.00
B	P_{in}	0.80	0.60	0.64
	P_{out}	0.80	0.88	0.72
C	P_{in}	0.84	0.88	0.72
	P_{out}	0.84	0.64	0.60
D	P_{in}	0.76	0.68	0.56
	P_{out}	0.76	0.64	0.68

Contributions. Summarizing, we can identify the following main challenges for Web services search: ($\mathcal{R}_1$) how to combine the degrees of match for the different parameters in the matched descriptions; ($\mathcal{R}_2$) how to combine the match results from the individual similarity measures; and ($\mathcal{R}_3$) how to rank the results. Our approach is based on the notion of *dominance* to address the problem of ranking available Web service descriptions with respect to a given service request, in the presence of multiple parameters and matching criteria. Given two objects U and V, each described by a set of d parameters, U is said to dominate V iff U is better than or equal to V in all the parameters, and strictly better in at least one parameter. This concept allows us to define three ranking criteria, presented in Section 3, which address the aforementioned challenges. Our contributions are summarized as follows:

1. We introduce the notion of top-k dominant Web services, specifying three ranking criteria for matching Web service descriptions with service requests using multiple similarity measures.
2. Based on this specification, we present efficient algorithms for selecting the top-k matches for a service request.
3. We experimentally evaluate our approach both in terms of retrieval effectiveness, using real requests and relevance sets, as well as in terms of efficiency, using synthetically generated scenarios.

Outline. The rest of the paper is structured as follows. Section 2 reviews related work. Section 3 formally introduces the problem of ranking Web service descriptions. Section 4 describes efficient algorithms for retrieving the top-k matches for a Web service request. Section 5 presents our experimental study. Finally, Section 6 concludes the paper.

2. RELATED WORK

In this section we discuss related work in the areas of Web service discovery, skylines, and data fusion.

Web Service Discovery. Current industry standards for the description and the discovery of Web services (WSDL, UDDI) provide limited search capabilities, focusing on describing the structure of the service interfaces and of the exchanged messages, and addressing the discovery process as keyword-based search. In [15], the need for employing many types of matching is identified, and the integration of multiple external matching services to a UDDI registry is proposed. Selecting the external matching service is based on specified policies (e.g., the first available, or the most successful). If more than one matching services are invoked, the policy specifies whether the union or the intersection of the results should be returned. The work in [16] focuses on similarity search for Web service operations, combining multiple sources of evidence. A clustering algorithm groups names of parameters into semantically meaningful concepts, used to determine the similarity between I/O parameters. Different types of similarity are combined using a linear function, with weights being assigned manually, based on analysis of the results from different trials. Learning the weights from user feedback is mentioned as for future work.

Following the Semantic Web vision, several approaches have been proposed exploiting ontologies to semantically enhance the service descriptions (WSDL-S [1], OWL-S [10], WSMO [19]). Web services matchmaking is then treated as a logic inference task [32, 28]. The matching algorithms in [11] and [39] assess the similarity between requested and offered inputs and outputs by comparing classes in an associated domain ontology. In [8] the matching of requested and offered parameters is treated as matching bipartite graphs. The work presented in [6] employs ontologies and user profiles and uses techniques like query expansion or relaxation to try to satisfy user requests. Finally, OWLS-MX [23] and WSMO-MX [21] are hybrid matchmakers for OWL-S and WSMO services, respectively.

These works focus on matching pairs of parameters from the requested and offered services, while the overall match is typically calculated as a weighted average, assuming the existence of an appropriate weighting scheme. Furthermore, none of these approaches considers more than one matching criteria simultaneously. However, from the diversity of these approaches, it is evident that there is no single matching criterion that constitutes the silver bullet for the problem. On the other hand, the approach proposed in this paper addresses this issue and provides a generic and efficient way to accommodate and leverage multiple matching criteria and service parameters, without loss of information from aggregating the individual results and without requiring a-priori knowledge concerning the user's preferences.

Skylines. Our case resembles concepts of multi-objective optimization, which has been studied in the literature, initially as *maximum vector* problem [25, 36], and more recently, as *skyline queries* [9]. Given a set of points in a d-dimensional space, the skyline is defined as the subset containing those points that are not dominated by any other point. Thus, the best answers for such a query exist in the skyline.

Skyline queries have received a lot of attention over the recent years, and several algorithms have been proposed. BNL [9] is a straightforward, generic skyline algorithm. It iterates over the data set, comparing each point with every other point, and reports the points that are not dominated by any other point. SFS [14] improves the efficiency of BNL, by pre-sorting the input according to a monotone scoring function F, reducing the number of dominance checks required. SaLSa [7] proposes an additional modification, so that the computation may terminate before scanning the whole data set.

Even though our work exploits the basic techniques underlying these methods (see Section 4), these algorithms are not directly applicable to our problem, as they do not deal with ranking issues (the objects comprising the skyline are incomparable to each other) or with the requirement for multiple matching criteria. Also, the size of the skyline is not known a-priori, and, depending on the data dimensionality and distribution may often be either too large or too small. In addition, our work borrows some ideas from the probabilistic skyline model for uncertain data introduced in [34], which however also does not provide any ranking of the data.

Other works exploit appropriate indexes, such as B^+-tree or R-tree, to speed-up the skyline computation process [40, 24, 33, 27]. Note that these techniques apply only on static data, where the overhead of building the index is amortized across multiple queries. In our setting, the underlying data depend on the matching scores and thus an index would have to be rebuilt for each query.

The importance of combining top-k queries with skyline queries has been pointed out in [42]. However, there are some important differences to our work. First, this approach also relies on the use of an index, in particular an aggregate R-tree. Second, it considers only one of the ranking criteria proposed in this paper (see Section 3). Third, it does not address the requirement for handling multiple matching criteria. The works in [13, 5] deal with the problem that the skyline in high dimensional spaces is too large. For this purpose, [13] relaxes the notion of dominance to k-dominance, so that more points are dominated. It also considers top-δ dominant skyline queries, which, in contrast to our case, return a set of at least (i.e., not exactly) δ points. Also, the selection criterion is different to ours; in fact, it resembles subspace skyline analysis [35]. On the other hand, [5] relies on users to specify additional preferences among points so as to decrease the result size. Finally, the k most representative skyline operator is proposed in [30]. This selects a set of k skyline points, so that the number of points dominated by at least one of them is maximized. Again, this differs from our ranking criteria, and it does not consider any extensions to meet the requirement for multiple similarity scores in our case.

Data Fusion. Given a set of ranked lists of documents returned from multiple methods – e.g., from different search engines, different databases, and so on – in response to a given query, *data fusion* (a.k.a. results merging, metasearch or rank aggregation) is the construction of a single ranked list combining the individual rankings. The data fusion techniques can be classified [3] based on whether they require knowledge of the relevance scores and whether training data is used. The simplest method based solely on the documents' ranks is the Borda-fuse model introduced in [3]. In its non-training flavor, it assigns as score to each document the summation of its rank (position) in each list. The documents in the fused list are ranked by increasing order of their score, solving ties arbitrarily. Training data can be used to assess the performance of each source and, hence, learn its importance. In this case, the sources are assigned weights relative to their importance and a document's score is the weighted summation of its ranks.

The Condorcet-fuse method [31] is another rank-based fusion approach. It is based on a majoritarian voting algorithm, which specifies that a document should be ranked higher in the fused list than another document if the former is ranked higher than the latter more times than the latter is ranked higher than the former. Condorcet-fuse proceeds iteratively: it identifies the winner(s), i.e., the highest ranked document(s), removes it/them from the lists and then repeats the process until there are no more documents to rank.

For the case where the relevance scores are given/known, several

fusion techniques, including CombSUM, CombANZ and CombMNZ, were discussed in [18]. In CombSUM, the final (fused) relevance score of a document is given by the summation of the relevance scores assigned by each source; if a document does not appear in a list, its relevance score is considered 0 for that list. In CombANZ (CombMNZ), the final score of a document is calculated as the score of CombSUM divided (multiplied) by the number of lists in which the document appears. In [26], the author concludes that CombMNZ provides the best retrieval efficiency.

When training data is available, it is shown in [41] that a linear (weighted) combination of scores works well when the various rank engines return similar sets of relevant documents and dissimilar sets of non-relevant documents. For example, a weighted variant of CombSUM is successfully used in [37] for the fusion of multilingual ranked lists. The optimal size of the training data that balances effectiveness and efficiency is investigated in [12].

Probabilistic fusion techniques, which rank documents based on their probability of relevance to the given query, have also appeared. The relevance probability is calculated in the training phase, and depends on which rank engine returned the document amongst its results and the document's position in the result set. In [29], such a technique was shown to outperform CombMNZ.

An outranking approach was recently presented in [17]. According to this, a document is ranked better than another if the majority of input rankings is in concordance with this fact and at the same time only a few input rankings refute it.

Seen in the context of data fusion, our work addresses the novel problem where in each ranking a *vector of scores*, instead of a single score, is used to measure the relevance for each data item.

3. PROBLEM DEFINITION

In this section, we formally describe the problem of multi-criteria Web services matching, we introduce our terminology and notation, and we formalize our notion for *top-k dominant Web services*. Also, to motivate and justify our formulation, we discuss some related notions, namely *p-skyline* [34] and *K-skyband* [33], showing that these concepts are inadequate in capturing the requirements of the problem at hand.

To abstract away from a particular Web service representation, we model a Web service operation as a function that receives a number of inputs and returns a number of outputs. Other types of parameters, such as pre-conditions and effects or QoS parameters, can be handled similarly. Hence, in the following, the description of a Web service operation corresponds to a vector S containing its I/O parameters. A request R is viewed as the description of a desired service operation, and is therefore represented in the same way.

Given a Web service request, the search engine matches registered services against the desired description. For this purpose, it uses a similarity measure f_m to assess the similarity between the parameters in these descriptions. If more than one offered parameters match a requested parameter, the closest match is considered. Thus, the result of matching a pair $\langle R, S\rangle$ is specified by a vector $U_{R,S}$, such that

$$\forall i \in [0, |R|] \;\; U_{R,S}[i] = \max_{j=0}^{|S|} f_m(R[i], S[j]). \qquad (1)$$

As discussed in Section 1, achieving better retrieval accuracy requires employing more than one matching criteria. Then, for each different similarity measure f_{m_i}, a match vector $U^{m_i}_{R,S}$ is produced. Hereafter, we refer to each such individual vector as *match instance*, denoted by lowercase letters (e.g., u, v), whereas to the set of such vectors for a specific pair $\langle R, S\rangle$ as *match object*, denoted by uppercase letters (e.g., U, V).

To address the challenges $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$ discussed in Section 1, we propose an approach based on the notion of *dominance*, also used in skyline queries [9]; i.e., queries returning those objects in a data set that are not dominated by any other object. Assume a set of match instances $\mathcal{I}$ in a d-dimensional space. Given two instances $u, v \in \mathcal{I}$, we say that u dominates v, denoted by $u \succ v$, iff u is better than or equal to v in all dimensions, and strictly better in at least one dimension, i.e.

$$u \succ v \Leftrightarrow \forall i \in [0, d)\, u[i] \geq v[i] \wedge \exists j \in [0, d)\, u[j] > v[j] \quad (2)$$

If u is neither dominated by nor dominates v, then u and v are incomparable. The notion of dominance addresses requirement ($\mathcal{R}_1$), since comparing matched services takes into consideration the degrees of match in all parameters, instead of calculating and using a single, overall score.

Example (Cont'd). Consider the example of Table 1. Let $S^{m_i}_{R,S} = (s_i.P_{in}, s_i.P_{out})$ denote the match vector under criterion f_{m_i} for the input and output parameters of service S. Figure 1 draws the degrees of match s_i as an instance in the $P_{in} \times P_{out}$ space for all services and criteria. For example, a_1 corresponds to the degrees of match of service A under f_{m_1} and, hence, has coordinates $(0.96, 0.92)$. Clearly, instances that are in the top right corner constitute better matches. Notice that instance d_1 dominates instances b_3 and c_3. Similarly, it is dominated by instances b_1 and c_1, as well as by all instances of A, but it neither dominates nor is dominated by b_2 and c_2.

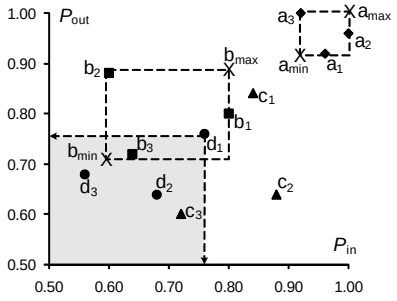


Figure 1: Services of Table 1 in the $P_{in} \times P_{out}$ space

To address the requirement of multiple matching criteria ($\mathcal{R}_2$), i.e., having a set of match instances per service, we use a model similar to the probabilistic skylines proposed in [34]. The dominance relationship between two instances u and v is defined as previously. Then, the dominance relationship between two objects U and V is determined by comparing each instance $u \in U$ to each instance $v \in V$. This may result in a partial dominance of U by V, in other words a probability under which U is dominated by V. Note that, without loss of generality, all instances of an object are considered of equal probability, i.e., all the different matching criteria employed are considered of equal importance; it is straightforward to extend the approach to the case that different weights are assigned to each matching criterion. Based on this notion, the work in [34] defines the concept of *p-skyline*, which comprises all the objects belonging to the skyline with probability at least p.

Although $\mathcal{R}_2$ is satisfied by the assumption of multiple instances per object, $\mathcal{R}_3$ is not fulfilled by the concept of p-skyline. The notion of skyline, and consequently that of p-skyline, is too restrictive:

only objects not dominated by any other object are returned. However, for Web services retrieval, given that the similarity measures provide only an indication of the actual relevance of the considered service to the given request, interesting services may be missed.

A possible work-around would be to consider a *K-skyband* query, which is a relaxed variation of a skyline query, returning those objects that are dominated by at most K other objects [33]. In particular, we could extend the p-skyline to the *p-K-skyband*, comprising those objects that are dominated by at most K other objects with probability at least p. Relaxing (restricting) the value of K, increases (reduces) the number of results to be returned. Still, such an approach faces two serious drawbacks. The first is how to determine the right values for the parameters p and K. A typical user may specify the number of results that he/she would like to be returned (e.g., top 10), but he/she cannot be expected to understand the semantics or tune such parameters neither it is possible to determine automatically the values of p and K from the number of desired results. Second, the required computational cost is prohibitive. Indeed, in contrast to the p-skyline where only one case for each object needs to be tested (i.e., the case that this object is not dominated by any other object), the p-K-skyband requires to consider, for each object, the cases that it is dominated by exactly $0, 1, 2, \ldots, K$ other objects, i.e., a number of $\sum_{j=0}^{K} \frac{N!}{j!(N-j)!}$ cases, where N is the total number of matches.

In the following, we formulate three ranking criteria that meet requirements $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$ without facing the aforementioned limitations. The first two are based, respectively, on the following intuitions: (a) a match is good if it is *dominated* by as *few* other matches as possible, and (b) a match is good if it *dominates* as *many* other matches as possible; the third *combines* both.

Dominated Score. Given an instance u, we define the dominated score of u, denoted by dds, as:

$$u.dds = \sum_{V \neq U} \frac{|\{v \in V \mid v \succ u\}|}{|V|} \tag{3}$$

Hence, $u.dds$ considers the instances that dominate u. The dominated score of an object U is defined as the (possibly weighted) average of the dominated scores of its instances:

$$U.dds = \sum_{u \in U} \frac{u.dds}{|U|} \tag{4}$$

The dominated score of an object indicates the average number of objects that dominate it. Hence, a *lower* dominated score indicates a better match.

Dominating Score. Given an instance u, we define the dominating score of u, denoted by dgs, as:

$$u.dgs = \sum_{V \neq U} \frac{|\{v \in V \mid u \succ v\}|}{|V|} \tag{5}$$

Thus, $u.dgs$ considers the instances that u dominates. The dominating score of an object U is defined as the (possibly weighted) average of the dominating scores of its instances:

$$U.dgs = \sum_{u \in U} \frac{u.dgs}{|U|} \tag{6}$$

The dominating score of an object indicates the average number of objects that it dominates. Hence, a *higher* dominating score indicates a better match.

Dominance Score. Given an instance u, we define the dominance score of u, denoted by ds, as:

$$u.ds = u.dgs - \lambda \cdot u.dds \tag{7}$$

The dominance score of u promotes u for each instance it dominates, while penalizing it for each instance that dominates it. The parameter λ is a scaling factor explained in the following. Consider an instance u corresponding to a good match. Then, it is expected that u will dominate a large number of other instances, while there will be only few instances dominating u. In other words, the dgs and dds scores of u will differ, typically, by orders of magnitude. Hence, the factor λ scales dds so that it becomes sufficient to affect the ranking obtained by dgs. Consequently, the value of λ depends on the size of the data set and the distribution of the data. A heuristic for selecting the λ value that works well in practice (see Section 5.1) is $\Delta dgs/\Delta dds$, where Δdgs and Δdds are the differences in the scores of the first and second result obtained by each respective criterion.

The dominance score of an object U is defined as the (possibly weighted) average of the dominance scores of its instances:

$$U.ds = \sum_{u \in U} \frac{u.ds}{|U|} \tag{8}$$

Example (Cont'd). Consider object C with instances c_1, c_2 and c_3 shown in Figure 1. Instance c_1 is dominated by a_1, a_2 and a_3, whereas it dominates b_1, b_3, d_1, d_2 and d_3. Thus, its scores are: $dds = 1$, $dgs = 5/3$ and $ds = 2/3$ (for $\lambda = 1$).

We now provide the formal definition for the top-k dominant Web services selection problem.

Formal Statement of the Problem. Given a Web service request R, a set of available Web services $\mathcal{S}$, and a set of similarity measures $\mathcal{F}_m$, return the top-k matches, according to the aforementioned ranking criteria.

4. RANKING WEB SERVICES

We first introduce some important observations pertaining to the problem at hand. The algorithms for selecting the top-k services according to dds, dgs and ds are presented in Sections 4.1, 4.2 and 4.3, respectively.

A straightforward algorithm for calculating the dominated (resp., dominating) score is the following. For each instance u of object U iterate over the instances of all other objects and increase a counter associated with U, if u dominates (resp., is dominated by) the instance examined. Then, to produce the top-k list of services, simply sort them according to the score in the counter. However, the applicability of this approach is limited by its large computation cost independent of k. Observe that no matter the value of k, it exhaustively performs all possible dominance checks among instances.

On the other hand, our algorithms address this issue by establishing lower and upper bounds for the dominated/dominating scores. This essentially allows us to (dis-)qualify objects to or from the results set, without computing their exact score. Let U be the current k-th object. For another object V to qualify for the result set, the score of V, as determined by its bounds, should be at least as good (i.e., lower, for dds, or higher, for dgs) as that of U. In the following, we delve into some useful properties of the dominance relationship (see Equation 2), in order to prune the search space.

Observe that the dominance relationship is transitive, i.e., given three instances u, v and w, if $u \succ v$ and $v \succ w$, then $u \succ w$. An important consequence for obtaining upper and lower bounds is the following.

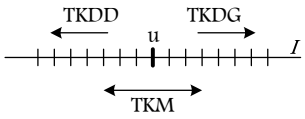


Figure 2: Search space for $\mathcal{TKDD}$, $\mathcal{TKDG}$, and $\mathcal{TKM}$

Property 1. *If $u \succ v$, then v is dominated by at least as many instances as u, i.e., $v.dds \geq u.dds$, and it dominates at most as many instances as u, i.e., $v.dgs \leq u.dgs$.*

Presorting the instances according to a monotone function, e.g., $F(u) = \sum_i u[i]$, can help reduce unnecessary checks.

Property 2. *Let $F(u)$ be a function that is monotone in all dimensions. If $u \succ v$, then $F(u) > F(v)$.*

To exploit this property, we place the instances in a list sorted in descending order of the sum of their values. Then, given an instance u, searching for instances by which u is dominated (resp., it dominates) can be limited to the part of the list before (resp., after) u. Furthermore, if $F(u)$ is also symmetric in its dimensions [7], e.g., $F(u) = \sum_i u[i]$, the following property holds, providing a termination condition.

Property 3. *Let $F(u)$ be a function that is monotone and symmetric in all dimensions. If $\min_i u[i] \geq F(v)$ for two instances u and v, then u dominates v as well as all instances with $F()$ value smaller than v's.*

Given an object U, let u_{min} be a virtual instance of U whose value in each dimension is the minimum of the values of the actual instances of U in that dimension, i.e., $u_{min}[i] = \min_{j=0}^{|U|} u_j[i],\ \forall i \in [0, d)$. Similarly, let $u_{max}[i] = \max_{j=0}^{|U|} u_j[i],\ \forall i \in [0, d)$. Viewed in a 2-$d$ space, these virtual instances, u_{min} and u_{max}, form, respectively, the lower-left and the upper-right corners of a virtual minimum bounding box containing the actual instances of U (see Figure 1). The following property holds.

Property 4. *For each instance $u \in U$, it holds that $u_{max} \succeq u$, and $u \succeq u_{min}$.*

Combined with the transitivity of the dominance relationship, this allows us to avoid an exhaustive pairwise comparison of all instances of two objects, by first comparing their corresponding minimum and maximum virtual instances. More specifically, given two objects U and V, (a) if u_{min} dominates v_{max}, then all instances of U dominate all instances of V, i.e., $u_{min} \succ v_{max} \Rightarrow u \succ v\ \forall u \in U, v \in V$; (b) if u_{min} dominates an instance of V, then all instances of U dominate this instance of V, i.e., $u_{min} \succ v \Rightarrow u \succ v\ \forall u \in U$; (c) if an instance of U dominates v_{max}, then this instance of U dominates all instances of V, i.e., $u \succ v_{max} \Rightarrow u \succ v\ \forall v \in V$.

4.1 Ranking by dominated score

The first algorithm, hereafter referred to as $\mathcal{TKDD}$, computes top-k Web services according to the dominated score criterion, dds. The goal is to quickly find, for each object, other objects dominating it, avoiding an exhaustive comparison of each instance to all other instances.

Algorithm $\mathcal{TKDD}$

Input: A set of objects $\mathcal{U}$, each comprising M instances;
The number k of results to return.
Output: The top-k objects w.r.t. *dds* in a sorted set $\mathcal{R}$.

begin
  Initialize $\mathcal{R} = \emptyset$; $ddsMax = \infty$; $minValue$ = -1 ;
  for $U \in \mathcal{U}$ **do**
    $(u_{min}, u_{max}) \leftarrow$ calculate min and max bounding instances ;
    $\mathcal{I}_{min} \leftarrow$ insert u_{min} ordered by $F(u_{min})$ desc. ;
    $\mathcal{I}_{max} \leftarrow$ insert u_{max} ordered by $F(u_{max})$ desc. ;
    for $u \in U$ **do** $\mathcal{I} \leftarrow$ insert u ordered by $F(u)$ desc. ;
  for $u_{max} \in \mathcal{I}_{max}$ **do**
    if $|\mathcal{R}| = k$ **then**
      if $F(u_{max}) \leq minValue$ **then return** $\mathcal{R}$;
      $U.dds = 0$;
      for $v_{min} \in \mathcal{I}_{min}^{F(u_{max})}$ **do**
        if $v_{min} \succ u_{max}$ **then**
          $U.dds = U.dds + 1$;
          if $(U.dds + V.dds) \geq ddsMax$ **then**
            for $u \in U$ **do** $u.dds = U.dds$;
            skip U;
      $U.dds = 0$;
      for $v \in \mathcal{I}^{F(u_{max})}$ **do**
        if $v \succ u_{max}$ **then**
          $U.dds = v.dds + 1/M$;
          if $(U.dds + v.dds) \geq ddsMax$ **then**
            for $u \in U$ **do** $u.dds = U.dds$;
            skip U;
      $U.dds = 0$;
      for $u \in U$ **do**
        for $v_{min} \in \mathcal{I}_{min}^{F}(u)$ **do**
          if $v_{min} \succ u$ **then**
            $u.dds = u.dds + 1/M$;
            if $(U.dds + u.dds + V.dds) \geq ddsMax$ **then**
              $U.dds = U.dds + u.dds + V.dds$;
              skip U;
      $U.dds = U.dds + u.dds + V.dds$;
    $U.dds = 0$;
    for $u \in U$ **do** $u.dds = 0$;
    for $u \in U$ **do**
      for $v \in \mathcal{I}^{F}(u)$ **do**
        if $v \succ u$ **then**
          $u.dds = u.dds + 1/M^2$;
          if $(U.dds + u.dds + v.dds) \geq ddsMax$ **then**
            $U.dds = U.dds + u.dds + v.dds$;
            skip U;
      $U.dds = U.dds + u.dds + v.dds$;
    if $|\mathcal{R}|$ = k **then** remove the last result from $\mathcal{R}$;
    $\mathcal{R} \leftarrow$ insert U ordered by dds asc.
    if $|\mathcal{R}| = k$ **then**
      $U_k \leftarrow$ the k-th object in $\mathcal{R}$;
      $ddsMax = U_k.dds$;
      $minValue = \min_{i=1}^{M}(U_{k_{min}}[i])$;
  return $\mathcal{R}$;
end

Figure 3: Algorithm $\mathcal{TKDD}$

The algorithm maintains three list, $\mathcal{I}_{min}$, $\mathcal{I}_{max}$, and $\mathcal{I}$, containing, respectively, the minimum bounding instances, the maximum bounding instances, and the actual instances of the objects. The instances inside these lists are sorted by $F(u) = \sum_i u[i]$ and are examined in descending order. The results are maintained in a list $\mathcal{R}$ sorted in ascending order of dds. The algorithm uses two variables,

$ddsMax$ and $minValue$, which correspond to an upper bound for dds, and to the minimum value of the current k-th object, respectively.

Given that, for an object U, we are interested in objects that *dominate* it, we search only for instances that are prior to those of U in $\mathcal{I}$ (see Figure 2). Since, the top matches are expected to appear in the beginning of $\mathcal{I}$, this significantly reduces the search space. The basic idea is to use the bounding boxes of the objects to avoid as many dominance checks between individual instances as possible. After k results have been acquired, we use the score of the k-th object as a maximum threshold. Objects whose score exceeds the threshold are pruned. In addition, if at some point, it is guaranteed that the score of all the remaining objects exceeds the threshold, the search terminates.

More specifically, the algorithm, shown in Figure 3, proceeds in the following six steps.

Step 1. *Initializations (lines 2–7).* The result set $\mathcal{R}$ and the variables $ddsMax$ and $minValue$ are initialized. The lists $\mathcal{I}_{min}$, $\mathcal{I}_{max}$, and $\mathcal{I}$ are initialized, and sorted by $F(u)$. Then the algorithm iterates over the objects, according to their maximum bounding instance.

Step 2. *Termination condition (line 10).* If the $F()$ value of the current u_{max} does not exceed the minimum value of the current k-th object, the result set $\mathcal{R}$ is returned and the algorithm terminates (see Property 3).

Step 3. *Dominance check object-to-object (lines 12–17).* For the current object U, the algorithm first searches for objects that fully dominate it. For example, in the case of the data set of Figure 1, with a single dominance check between b_{max} and a_{min}, we can conclude that all instances b_1, b_2 and b_3 are dominated by a_1, a_2 and a_3. According to property 2, only objects with $F(v_{min}) > F(u_{max})$ need to be checked. If a v_{min} is found to dominate u_{max}, then the score of U is increased by 1, and the sum of the new score and the score of V (see Property 1) is compared to the current threshold, $ddsMax$. If it exceeds the threshold, the object is pruned and the iteration continues with the next object. In this case, the score of the object is propagated to its instances for later use. Otherwise, the score of the object is reset, to avoid duplicates, and the search continues in the next step.

Step 4. *Dominance check object-to-instance (lines 19–24).* This step searches for individual instances v that dominate U. For example, in Figure 1, a dominance check between d_{max} (which coincides with d_1) and c_1 shows that all instances d_1, d_2, and d_3 are dominated by c_1. As before, only instances with $F(v) > F(u_{max})$ are considered. If an instance v is found to dominate u_{max}, then the score of U is increased by $1/M$, where M is the number of instances per object, and the sum of the new score and that of v is compared to the current threshold, $ddsMax$.

Step 5. *Dominance check instance-to-object (lines 26–33).* If the object U has not been pruned in the previous two steps, its individual instances are considered. Each instance u is compared to instances v_{min}, with $F(v_{min}) > F(u)$. If it is dominated, the score of u is again increased by $1/M$, and the threshold is checked. In Figure 1, this is the case with d_3 and b_{min}.

Algorithm $\mathcal{TKDG}$

Input: A list $\mathcal{I}$ containing all instances u, in descending order of $F(u)$; The number k of results to return.
Output: The top-k objects w.r.t. dgs in a sorted set $\mathcal{R}$.

```
1  begin
2    Initialize R = ∅, L = ∅;
3    U ← the set of objects in descending order of F(u_max) ;
4    for every object U ∈ U do
5      if ( (|I| − pos(u_max)) / M < R_{k−1}.dgs⁻ ) then return R;
6      if ( |R| = 0 ) then add U in R;
7      if (∃V ∈ L ∪ R_{k−1} s.t. V fully dominates U then skip U;
8      set U.dgs⁻ = 0, U.dgs⁺ = Σ_{u∈U} (|I| − pos(u)) / M²,
       U_i = pos(u_max);
9      for j = |R| − 1 to 0 do
10       while ( not ( U.dgs⁺ < R_j.dgs⁻ or U.dgs⁻ > R_j.dgs⁺ ) ) do
11         refineBounds (U, R_j);
12       if ( U.dgs⁺ < R_j.dgs⁻ ) then
13         if ( j = k − 1 ) then add U in L, and continue with the next object;
14         else move R_{k−1} to L, add U in R after R_j, and continue with the
           next object;
15     move R_{k−1} to L, and add U at the beginning of R;
16   return R;
17 end
```

Figure 4: Algorithm $\mathcal{TKDG}$

Step 6. *Dominance check instance-to-instance (lines 35–42).* If all previous steps failed to prune the object, a comparison between individual instances takes place where each successful dominance check contributes to the object's score by $1/M^2$.

Step 7. *Result set update (lines 44–49).* If U has not been pruned in any of the previous steps, it is inserted in the result set $\mathcal{R}$. If k results exist, the last is removed. After inserting the new object, if the size of $\mathcal{R}$ is k, the thresholds $ddsMax$ and $minValue$ are set accordingly.

4.2 Ranking by dominating score

The $\mathcal{TKDG}$ algorithm, shown in Figure 4, computes the top-k dominant Web services with respect to the dominating score, i.e., it retrieves the k match objects that dominate the larger number of other objects. This is a more challenging task compared to that of $\mathcal{TKDD}$, for the following reason. Let $pos(u)$ denote the position of the currently considered instance u in the sorted, decreasing by F, list $\mathcal{I}$ of instances. To calculate $u.dds$, $\mathcal{TKDD}$ performs in the worst case $pos(u)$ dominance checks, i.e., with those before u in the list. On the other hand to calculate $u.dgs$, $\mathcal{TKDG}$ must perform in the worst case $|\mathcal{I}| - pos(u)$ checks, i.e., those after u (see Figure 2). Since the most dominating and less dominated objects are located close to the beginning of $\mathcal{I}$, execution will terminate when $pos(u)$ is small relative to $|\mathcal{I}|$. As a result, the search space for $\mathcal{TKDG}$ is significantly larger than $\mathcal{TKDD}$'s. Furthermore, $\mathcal{TKDD}$ allows for efficient pruning as it searches among objects and/or instances that have already been examined in a previous iteration, and therefore (the bounds of) their scores are known.

The $\mathcal{TKDG}$ algorithm maintains three structures: (1) the $\mathcal{I}$ list; (2) a list $\mathcal{R}$ of at most k objects (current results), ordered by dominating score descending; (3) a list $\mathcal{L}$ containing objects that have been disqualified from $\mathcal{R}$, used to prune other objects. The lists $\mathcal{R}$ and $\mathcal{L}$ are initially empty.

Similar to $\mathcal{TKDD}$, the algorithm iterates over the objects, in descending order of their maximum bounding instance *(lines 3–4)*.

Let U be the currently examined object. U can dominate at most $|\mathcal{I}| - pos(u_{max})$ instances. If this amount, divided by the number of instances per object, is lower than T, where T is the lower bound for the score of the k-th object in $\mathcal{R}$, the whole process terminates, and the result set $\mathcal{R}$ is returned *(line 5)*. On the other hand, if the result set is empty, then U is added as the first result *(line 6)*.

Next, if U is dominated by the k-th object in $\mathcal{R}$ or by any object in $\mathcal{L}$, it is pruned *(line 7)*. Otherwise, we need to check whether U qualifies for $\mathcal{R}$. For an examined object U it is straightforward to calculate its dominating score, by examining all instances in $\mathcal{I}$, starting from the position of its best instance. However, we avoid unnecessary computations by following a lazy approach, which examines instances in $\mathcal{I}$ until a position that is sufficient to qualify (disqualify) U for (from) the current result set $\mathcal{R}$. For this purpose, we maintain for each examined object U a lower and an upper bound for its dominating score, $U.dgs^-$ and $U.dgs^+$ respectively, as well as the last examined position in $\mathcal{I}$, denoted by U_i. We initialize the lower and upper bounds for $U.dgs$ to

$$U.dgs^- = 0 \text{ and } U.dgs^+ = \sum_{u \in U} \frac{|\mathcal{I}| - pos(u)}{M^2},$$

respectively. Also, the last examined position for U is initialized to $U_i = pos(u_{max})$ *(line 8)*.

Let V be the k-th result in $\mathcal{R}$. We start by comparing U with V. Three cases may occur: (1) if $U.dgs^+ < V.dgs^-$, then U does not qualify for $\mathcal{R}$, and it is inserted in $\mathcal{L}$; (2) if $U.dgs^- > V.dgs^+$, U is inserted in R before V, and it is recursively compared to the preceding elements of V in $\mathcal{R}$; if V was the k-th object in $\mathcal{R}$, it is removed from R and it is inserted in $\mathcal{L}$; (3) otherwise, the lower and upper bounds of U and V need to be refined, until one of the conditions (1) or (2) is satisfied. This refinement is performed by searching in $\mathcal{I}$ for instances dominated by an instance of U, starting from the position U_i. At each step of this search, the instance at this position, v, is compared to the instances of U preceding it. For each instance u of U that dominates (does not dominate) v, the lower (upper) bound of the dominating score of U is increased (decreased) by $1/M^2$. Also, the last examined position for U is incremented by 1. Notice that, as in $\mathcal{TKDD}$, if $F(v)$ does not exceed the minimum value of u, then u dominates v and all its susequent instances, hence, the lower bound of the score of u is updated accordingly, without performing dominance checks with those instances *(lines 9–15)*.

4.3 Ranking by dominance score

The previously presented algorithms take into consideration either one of the dds or dgs scores. In the following, we present an algorithm, referred to as $\mathcal{TKM}$, that computes the top-k matches with respect to the third criterion introduced in Section 3, which combines both measures. In particular, this algorithm is derived by the algorithm $\mathcal{TKDG}$, with an appropriate modification to account also for the dominated score. More specifically, this modification concerns the computation of the lower and upper bounds of the scores. First, the lower bound for the score of an object is now initialized as:

$$U.dgs^- = -\lambda \cdot \sum_{u \in U} \frac{pos(u)}{M^2} \quad (9)$$

instead of 0. Second, the bounds refinement process now needs to consider two searches, one for instances dominated by the current object, and one for instances that dominate the current object (see Figure 2). These searches proceed interchangeably, and the bounds are updated accordingly. Consequently, two separate cursors need to be maintained for each object, to keep track of the progress of each search in the list containing the instances.

5. EXPERIMENTAL EVALUATION

In this section we present an extensive experimental study of our approach. In particular, we conduct two sets of experiments. First, we investigate the benefits resulting from the use of the proposed ranking criteria with respect to the recall and precision of the computed results. For this purpose, we rely on a publicly available, well-known benchmark for Web service discovery, comprising real-world service descriptions, sample requests, and relevance sets. In particular, the use of the latter, which are manually identified, allows to compare the results of our methods against human judgement. In the second set of experiments, we consider the computational cost of the proposed algorithms under different combinations of values for the parameters involved, using synthetic data sets.

Our approach has been implemented in Java and all the experiments were conducted on a Pentium D 2.4GHz with 2GB of RAM, running Linux.

5.1 Retrieval Effectiveness

To evaluate the quality of the results returned by the three proposed criteria, we have used the publicly available service retrieval test collection OWLS-TC v2 [2]. This collection contains real-world Web service descriptions, retrieved mainly from public IBM UDDI registries. More specifically, it comprises: (a) 576 service descriptions, (b) 28 sample requests, and (c) a manually identified relevance set for each request.

Our prototype comprises two basic components: (a) a matchmaker, based on the OWLS-MX service matchmaker [23], and (b) a component implementing the algorithms presented in Section 4 for processing the degrees of match computed by the various matching criteria and determining the final ranking of the retrieved services.

OWLS-MX matches I/O parameters extracted from the service descriptions, exploiting either purely logic-based reasoning (M0) or combined with some content-based, IR similarity measure. In particular, the following measures are considered: loss-of-information measure (M1), extended Jaccard similarity coefficient (M2), cosine similarity (M3), and Jensen-Shannon information divergence based similarity (M4). Given a request R and a similarity measure (M0–M4), the degrees of match among its parameters and those of a service S are calculated and then aggregated to produce the relevance score of S. Therefore, given a request, a ranked list of services is computed for each similarity criterion. Note that in OWLS-MX no attempt to combine rankings from different measures is made.

We have adapted the matching engine of OWLS-MX as follows. For a pair $\langle R, S \rangle$, instead of a single aggregated relevance score, we retrieve a score vector containing the degrees of match for each parameter. Furthermore, for any such pair, all similarity criteria (M0–M4) are applied, resulting in five score vectors. Hence, for a request having in total d I/O parameters, each matched service corresponds essentially to an object, and the score vectors correspond to the object's d-dimensional instances. Then, the algorithms $\mathcal{TKDD}$, $\mathcal{TKDG}$, and $\mathcal{TKM}$, described in Section 4, are applied to determine the ranked list of services for each criterion.

To evaluate the quality of the results, we apply the following standard IR evaluation measures [4]:

- *Interpolated Recall-Precision Averages*: measures precision, i.e., percent of retrieved items that are relevant, at various

[2] `http://www-ags.dfki.uni-sb.de/~klusch/owls-mx/`

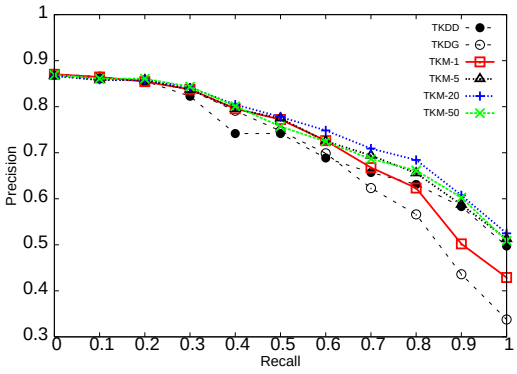


Figure 5: Recall-Precision graphs: $\mathcal{TKDD}$, $\mathcal{TKDG}$ and $\mathcal{TKM}$

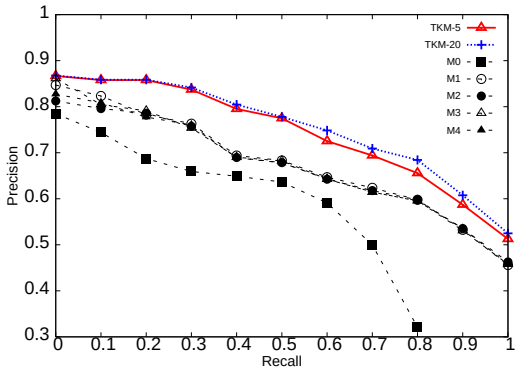


Figure 6: Recall-Precision graphs: $\mathcal{TKM}$ and individual measures M0-M4

recall levels, i.e., after a certain percentage of all the relevant items have been retrieved.

- *Mean Average Precision (MAP)*: average of precision values calculated after each relevant item is retrieved.
- *R-Precision (R-prec)*: measures precision after all relevant items have been retrieved.
- *bpref*: measures the number of times judged non-relevant items are retrieved before relevant ones.
- *Reciprocal Rank (R-rank)*: measures (the inverse of) the rank of the top relevant item.
- *Precision at N (P@N)*: measures the precision after N items have been retrieved.

The conducted evaluation comprises three stages.

First, we compare the three different ranking criteria considered in our approach. The resulting recall-precision graphs are depicted in Figure 5. Regarding $\mathcal{TKM}$, we study the effect of the parameter λ (see Section 3), considering 4 variations, denoted as $\mathcal{TKM}$-λ, for λ=1, 5, 20, 50. As shown in Figure 5, for a recall level up to 30%, the performance of all methods is practically the same. Differences start to become more noticeable after a recall level of around 60%, where the precision of $\mathcal{TKDG}$ starts to degrade at a considerably higher rate compared to that of $\mathcal{TKDD}$. This means that several services, even though dominating a large number of other matches, were not identified as relevant in the provided relevance sets. On the other hand, as expected, the behavior of $\mathcal{TKM}$ is dependent on the value of λ. Without considering any scaling factor, i.e., for λ=1, the effect of the dds criterion is low, and, hence, although $\mathcal{TKM}$ performs better than $\mathcal{TKDG}$, it still follows its trend. However, significant gains are achieved by values of λ that strike a good balance between the two criteria, dds and dgs. The heuristic presented in Section 3, provides us with a starting value λ_H, which is equal to 5 for the data set into consideration. All our experiments with the real data show that $\mathcal{TKM}$-λ_H, i.e., $\mathcal{TKM}$ having λ=λ_H, produces better results than the other two methods. This is illustrated by the graph $\mathcal{TKM}$-5 in Figure 5. In addition, the experiments show that for values of λ lower than λ_H, $\mathcal{TKM}$ does not produce better results; i.e., the effect of dds is still not sufficient. On the other hand, we can get further improved results (by a factor of around 1% in our experimental data set), by tuning λ into a range of values belonging to the same order of magnitude as λ_H. (Obviously, the tuning of λ is required only once per data set.) In our experiments, we got the best performance of $\mathcal{TKM}$ for values of λ around 20, which, as demonstrated by the graph in Figure 5, produces slightly better precision than $\mathcal{TKM}$-5. Further increasing the factor λ, i.e., the effect of the dds criterion, fails to provide better results, and, as expected, it eventually converges back to $\mathcal{TKDD}$, as illustrated by the $\mathcal{TKM}$-50 graph in Figure 5.

Next, we examine the resulting benefit of the dominance-based ranking compared to applying either of the individual similarity measures M0-M4. The recall-precision measures are illustrated in Figure 6. To avoid overloading the figure, only the $\mathcal{TKM}$-5 and $\mathcal{TKM}$-20 have been plotted. As shown, the dominance-based ranking clearly outperforms all the individual similarity measures. This can be attributed to the fact that the combination of all the matching criteria constitutes the matchmaker more tolerant to the false positive or false negative results returned by the individual measures [22].

As this is not very surprising, to better gauge the effectiveness of our methodology, we finally compare it to better informed approaches, as well. When multiple rankings exist, a common practice for boosting accuracy is to combine, or *fuse*, the individual results. Several methods, reviewed in Section 2, exist for this task. We compare our method to four popular fusion techniques: the score-based approaches CombSum and CombMNZ [18], the simple rank-based method of Borda-fuse [3], and the Outranking approach [17]. The first three techniques are parameter-free. On the other hand, the latter requires a family of outranking relations, where each relation is defined by four threshold values $(s_p, s_u, c_{min}, d_{max})$. We chose to employ a single outranking relation, setting the parameters to $(0, 0, M, M-1)$, satisfying, thus, Pareto-optimality (M denotes the number of ranking lists, or criteria, which is 5 in our case). The obtained recall-precision graphs are shown in Figure 7. Again, our approach clearly outperforms the other methods. This gain becomes even more apparent, when noticing through Figures 6 and 7 that these fusion techniques, in contrast to our approach, fail to demonstrate a significant improvement over the individual similarity measures.

In addition to the recall-precision graphs discussed above, Table 2 details the results of all the compared methods for all the aforementioned IR metrics. For each metric, the highest value is shown in bold (we treat the values of both versions of $\mathcal{TKM}$ uniformly), whereas the second highest in italic. In summary, $\mathcal{TKDD}$ and $\mathcal{TKDG}$ produce an average gain of 8.33% and 6.44%, respectively, with respect to the other approaches. Additionally, $\mathcal{TKM}$-5

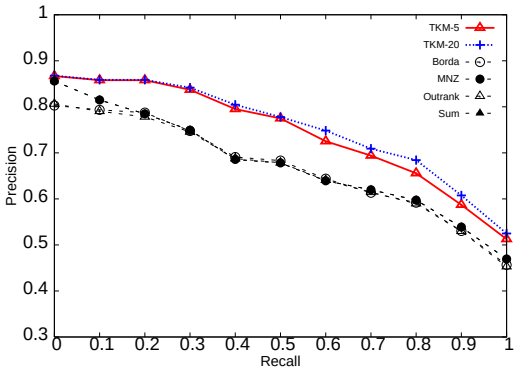


Figure 7: $\mathcal{TKM}$ and fusion approaches

Table 2: IR metrics for all methods

Method	MAP	R-prec	bpref	R-rank	P@5	P@10	P@15	P@20
TKDD	*0.7050*	*0.6266*	*0.6711*	0.8333	*0.8071*	0.6893	0.6143	*0.5446*
TKDG	0.6750	0.6233	0.6334	0.8333	**0.8143**	*0.7143*	*0.6238*	0.5089
TKM-5	**0.7249**	**0.6618**	**0.7098**	*0.8393*	0.8000	0.7036	**0.6738**	**0.5714**
TKM-20	**0.7375**	**0.6808**	**0.7243**	*0.8393*	0.8000	**0.7250**	**0.6857**	**0.5750**
M0	0.5097	0.5128	0.5138	0.7217	0.6357	0.6071	0.5357	0.4464
M1	0.6609	0.5966	0.6313	0.8155	0.7571	0.6679	0.5738	0.5268
M2	0.6537	0.5903	0.6260	0.7708	0.7357	0.6536	0.5762	0.5232
M3	0.6595	0.5924	0.6254	**0.8482**	0.7357	0.6571	0.5762	0.5161
M4	0.6585	0.5822	0.6234	0.8127	0.7429	0.6571	0.5690	0.5250
Borda	0.6509	0.5778	0.6210	0.7577	0.7357	0.6464	0.5667	0.5179
MNZ	0.6588	0.5903	0.6274	0.8214	0.7357	0.6536	0.5738	0.5286
Outrank	0.6477	0.5811	0.6164	0.7575	0.7214	0.6500	0.5643	0.5179
Sum	0.6588	0.5903	0.6274	0.8214	0.7357	0.6536	0.5738	0.5286

and $\mathcal{TKM}$-20 improve the quality of the results by a percentage (average values) of 11.44% and 12.56%, respectively.

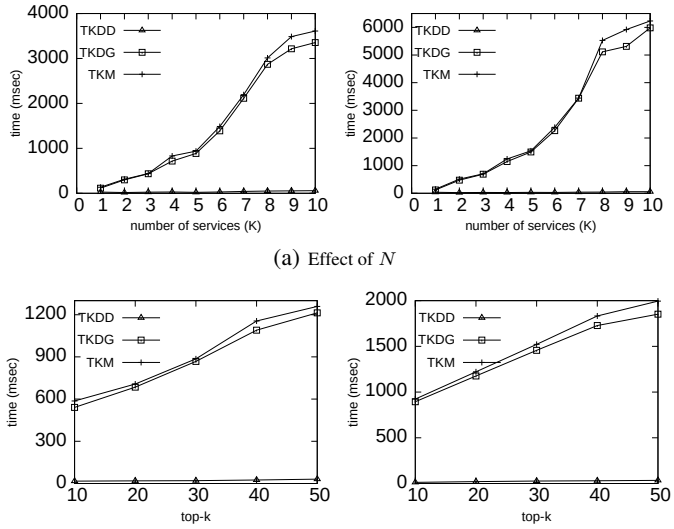
5.2 Computational Cost

In this section, we consider the computational cost of $\mathcal{TKDD}$, $\mathcal{TKDG}$, and $\mathcal{TKM}$, for different values of the involved parameters. These parameters and their examined values are summarized in Table 3. Parameters N and k refer to the number of available services and the number of results to return, respectively. Parameter d corresponds to the number of parameters in the service request, i.e., the dimensionality of the match objects. Parameter M denotes the number of distinct matching criteria employed by the service matchmaker.

We first provide a theoretical analysis, and then report our experimental findings on synthetically generated data sets.

Theoretical Analysis. To determine the dominated and dominating scores, our methods need to compare the instances of all services with each other, in the worst case. In total, there are $N \cdot M$ instances (i.e., M match instances per service), hence we perform $O(N^2M^2)$ dominance checks. For any pair of instances, a dominance check needs to examine the degrees of match for all d parameters. As a result, the complexity of our methods is $O(dN^2M^2)$. Clearly, this is a worst-case bound, as our algorithms need only find the top-k dominant services and employ various optimizations for reducing the number of dominance checks.

For the sake of comparison, we also discuss briefly the computational cost of the fusion techniques considered in Section 5.1. These take as input M lists, one for each criterion, containing the N advertised services ranked in decreasing order of their *overall* degree of match with the request. Therefore, an aggregation of the individual parameter-wise scores is required. CombSum, CombMNZ and Borda-fuse scan the lists, compute a *fused* score for each service and output the results sorted by this score. This procedure costs $O(NM + N \log N)$, where the first (second) summand corresponds to scanning (sorting). The Outranking method computes the fused score in a different manner: for each pair of services it counts agreements and disagreements as to which is better in the ranked lists. Therefore, its complexity is $O(N^2M)$. Note that all fusion techniques are independent of d due to the reduction of the individual parameter scores to a single overall score. In practice, the performance of $\mathcal{TKDG}$ and $\mathcal{TKM}$ resembles that of the Outranking method, while $\mathcal{TKDD}$ performs as well as the other fusion approaches. Therefore, for clarity of the presentation, in the rest of our analysis, we focus only on the three proposed methods.



(a) Effect of N

(b) Effect of k

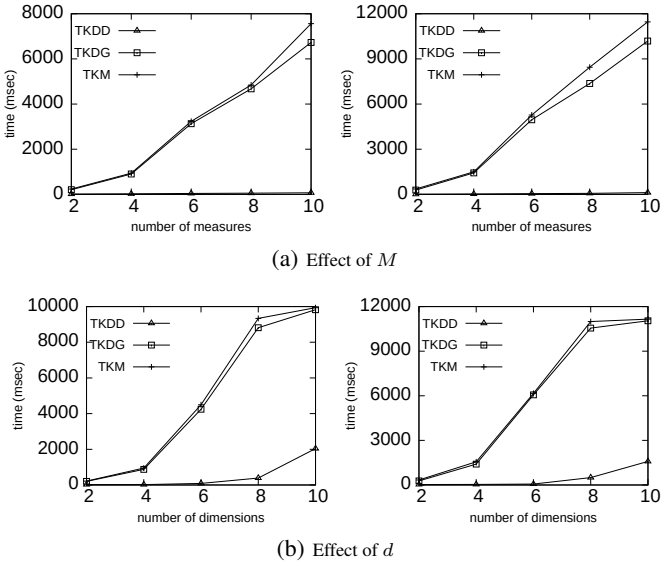
Figure 8: Effect of parameters under low (left graph of each pair) and high (right graph of each pair) variance var

Experimental Analysis. We used a publicly available synthetic generator [3] to obtain different types of data sets, with varying distributions represented by the parameters $corr$ and var, shown in Table 3. Given a similarity metric, parameter $corr$ denotes the correlation among the degrees of match for the parameters of a service. We consider three distributions: in independent (ind), degrees of match are assigned independently to each parameter; in correlated (cor), the values in the match instance are positively correlated, i.e., a good match in some service parameters increases the possibility of a good match in the others; in anti-correlated (ant) the values are negatively correlated, i.e., good matches (or bad matches) in all parameters are less likely to occur. Parameter var controls the variance of results among similarity metrics. When var is low, matching scores from different criteria are similar. Hence, the instances of the same match object are close to each other in the d-dimensional space. On the other hand, when var is high, the matching scores from different criteria are dissimilar and, consequently, instances are far apart. We report our measures for variance around 10% (low) and 20% ($high$).

[3] `http://randdataset.projects.postgresql.org/`

Table 3: Parameters and examined values

Parameter	Symbol	Values
Number of services	N	[1, 10]K, **5K**
Number of results	k	10, 20, **30**, 40, 50
Number of dimensions	d	2, **4**, 6, 8, 10
Number of instances	M	2, **4**, 6, 8, 10
Parameter correlation	$corr$	**ind**, cor, ant
Instance variance	var	low, high



(a) Effect of M

(b) Effect of d

Figure 9: Effect of parameters under low (left graph of each pair) and high (right graph of each pair) variance var

In all experimental setups, we investigate the effect of one parameter, while we set the remaining ones to their default values, shown bold in Table 3. As a default scenario, we consider a request with 4 parameters, asking for the top-30 matches of a set of 5K partially matching service descriptions, using 4 different similarity measures. For the factor λ in $\mathcal{TKCM}$, we used the value λ_H appropriately estimated for each corresponding data set. The results are presented in Figure 9.

In general, all experiments indicate that $\mathcal{TKDD}$ is the most efficient method. As already discussed, $\mathcal{TKDD}$ is interested in objects that dominate the top match objects; hence, it searches a relatively small portion of the data set. On the contrary, the search space for $\mathcal{TKDG}$ is significantly larger, so its delay is expected. Similarly, $\mathcal{TKCM}$ performance suffers mainly due to the impact of dgs score; therefore, it is reasonable that it follows the same trend as $\mathcal{TKDG}$, with a slight additional overhead for accounting for dds score, as well. These observations are more apparent in Figure 8(a), where it can be seen that $\mathcal{TKDD}$ is very slightly affected, as opposed to $\mathcal{TKDG}$ and $\mathcal{TKCM}$, by the size of the data set. Another interesting observation refers to the effect of the dimensionality (Figure 9(b)), which at higher values becomes noticeable even for $\mathcal{TKDD}$. This, in fact, is a known problem faced by the skyline computation approaches as well. As the dimensionality increases, it becomes increasingly more difficult to find instances dominating other instances; hence, many unnecessary dominance

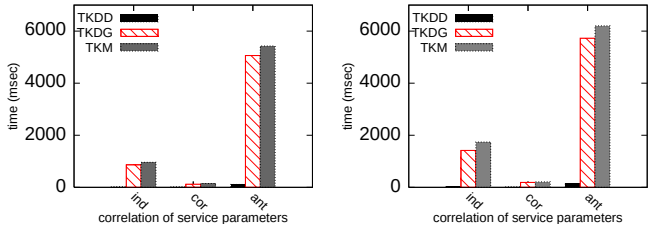


Figure 10: Effect of $corr$ under low (left) and high (right) variance var

checks are performed. A possible work-around is to group together related service parameters so as to decrease the dimensionality of the match objects. For the same reasons, a similar effect is observed in Figure 10. For correlated data sets, where many successful dominance checks occur, the computational cost for all methods drops close to zero. On the contrary, for anti-correlated data sets, where very few dominance checks are successful, the computational cost is significantly larger.

Summarizing, the final choice of the appropriate ranking method depends on the application. All three proposed measures produce significantly more effective results than the previously known approaches. If an application favors more accurate results, then $\mathcal{TKCM}$ seems as an excellent solution. If the time factor acts as the driving decision point, then $\mathcal{TKDD}$ should be favored, since it provides high quality results (see Table 2) almost instantly (see Figures 9 and 10).

6. CONCLUSIONS

In this paper, we have addressed the issue of top-k retrieval of Web services, with multiple parameters and under different matching criteria. We have presented three suitable criteria for ranking the match results, based on the notion of *dominance*. We have provided three different algorithms, $\mathcal{TKDD}$, $\mathcal{TKDG}$, and $\mathcal{TKCM}$, for matching Web service descriptions with service requests according to these criteria. Our extensive experimental evaluation shows that our approach compared to the state of the art significantly improves the effectiveness of the search by an average factor of 12%. Since our methods combine more than one similarity measures, their execution time is affected. However, $\mathcal{TKDD}$ runs in almost instant time, while still producing results that outperform the traditional approaches by an average extent of 8.33%.

7. REFERENCES

[1] R. Akkiraju and et. al. Web Service Semantics - WSDL-S. In *W3C Member Submission*, November 2005.

[2] G. Alonso, F. Casati, H. A. Kuno, and V. Machiraju. *Web Services - Concepts, Architectures and Applications*. Data-Centric Systems and Applications. Springer, 2004.

[3] J. A. Aslam and M. H. Montague. Models for metasearch. In *SIGIR*, pages 275–284, 2001.

[4] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[5] W.-T. Balke, U. Güntzer, and C. Lofi. Eliciting matters - controlling skyline sizes by incremental integration of user preferences. In *DASFAA*, pages 551–562, 2007.

[6] W.-T. Balke and M. Wagner. Cooperative Discovery for User-Centered Web Service Provisioning. In *ICWS*, pages 191–197, 2003.

[7] I. Bartolini, P. Ciaccia, and M. Patella. Efficient sort-based skyline evaluation. *ACM TODS*, 33(4):1–45, 2008.

[8] U. Bellur and R. Kulkarni. Improved Matchmaking Algorithm for Semantic Web Services Based on Bipartite Graph Matching. In *ICWS*, pages 86–93, 2007.

[9] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *ICDE*, pages 421–430, 2001.

[10] M. Burstein and et. al. OWL-S: Semantic Markup for Web Services. In *W3C Member Submission*, November 2004.

[11] J. Cardoso. Discovering Semantic Web Services with and without a Common Ontology Commitment. In *IEEE SCW*, pages 183–190, 2006.

[12] S. Cetintas and L. Si. Exploration of the Tradeoff Between Effectiveness and Efficiency for Results Merging in Federated Search. In *SIGIR*, pages 707–708, 2007.

[13] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. Finding k-dominant Skylines in High Dimensional Space. In *SIGMOD*, pages 503–514, 2006.

[14] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with Presorting. In *ICDE*, pages 717–816, 2003.

[15] J. Colgrave, R. Akkiraju, and R. Goodwin. External Matching in UDDI. In *ICWS*, page 226, 2004.

[16] X. Dong, A. Y. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity Search for Web Services. In *VLDB*, pages 372–383, 2004.

[17] M. Farah and D. Vanderpooten. An Outranking Approach for Rank Aggregation in Information Retrieval. In *SIGIR*, pages 591–598, 2007.

[18] E. A. Fox and J. A. Shaw. Combination of Multiple Searches. In *2nd TREC, NIST*, pages 243–252, 1993.

[19] H. Lausen, A. Polleres, and D. Roman (eds.). Web Service Modeling Ontology (WSMO). In *W3C Member Submission*, June 2005.

[20] T. Joachims and F. Radlinski. Search engines that learn from implicit feedback. *IEEE Computer*, 40(8):34–40, 2007.

[21] F. Kaufer and M. Klusch. WSMO-MX: A Logic Programming Based Hybrid Service Matchmaker. In *ECOWS*, pages 161–170, 2006.

[22] M. Klusch and B. Fries. Hybrid OWL-S Service Retrieval with OWLS-MX: Benefits and Pitfalls. In *SMRR*, 2007.

[23] M. Klusch, B. Fries, and K. P. Sycara. Automated Semantic Web service discovery with OWLS-MX. In *AAMAS*, pages 915–922, 2006.

[24] D. Kossmann, F. Ramsak, and S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *VLDB*, pages 275–286, 2002.

[25] H. T. Kung, F. Luccio, and F. P. Preparata. On Finding the Maxima of a Set of Vectors. *J. ACM*, 22(4):469–476, 1975.

[26] J.-H. Lee. Analyses of Multiple Evidence Combination. In *SIGIR*, pages 267–276, 1997.

[27] K. C. K. Lee, B. Zheng, H. Li, and W.-C. Lee. Approaching the Skyline in Z Order. In *VLDB*, pages 279–290, 2007.

[28] L. Li and I. Horrocks. A Software Framework for Matchmaking based on Semantic Web Technology. In *WWW*, pages 331–339, 2003.

[29] D. Lillis, F. Toolan, R. W. Collier, and J. Dunnion. ProbFuse: A Probabilistic Approach to Data Fusion. In *SIGIR*, pages 139–146, 2006.

[30] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting Stars: The k Most Representative Skyline Operator. In *ICDE*, pages 86–95, 2007.

[31] M. H. Montague and J. A. Aslam. Condorcet Fusion for Improved Retrieval. In *ACM CIKM*, pages 538–548, 2002.

[32] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara. Semantic Matching of Web Services Capabilities. In *ISWC*, pages 333–347, 2002.

[33] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive Skyline Computation in Database Systems. *ACM TODS*, 30(1):41–82, 2005.

[34] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic Skylines on Uncertain Data. In *VLDB*, pages 15–26, 2007.

[35] J. Pei, Y. Yuan, X. Lin, W. Jin, M. Ester, Q. Liu, W. Wang, Y. Tao, J. X. Yu, and Q. Zhang. Towards Multidimensional Subspace Skyline Analysis. *ACM TODS*, 31(4):1335–1381, 2006.

[36] F. P. Preparata and M. I. Shamos. *Computational geometry: An introduction*. Springer-Verlag New York, Inc., 1985.

[37] L. Si and J. Callan. CLEF 2005: Multilingual Retrieval by Combining Multiple Multilingual Ranked Lists. In *Proceedings of the 6th Workshop of the Cross-Language Evalution Forum*, pages 121–130, 2005.

[38] D. E. Simmen, M. Altinel, V. Markl, S. Padmanabhan, and A. Singh. Damia: data mashups for intranet applications. In *SIGMOD*, pages 1171–1182, 2008.

[39] D. Skoutas, A. Simitsis, and T. K. Sellis. A Ranking Mechanism for Semantic Web Service Discovery. In *IEEE SCW*, pages 41–48, 2007.

[40] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient Progressive Skyline Computation. In *VLDB*, pages 301–310, 2001.

[41] C. C. Vogt and G. W. Cottrell. Fusion Via a Linear Combination of Scores. *Information Retrieval*, 1(3):151–173, 1999.

[42] M. L. Yiu and N. Mamoulis. Efficient Processing of Top-k Dominating Queries on Multi-Dimensional Data. In *VLDB*, pages 483–494, 2007.