Ranking and Clustering Web Services using Multi-Criteria Dominance Relationships

Dimitrios Skoutas, Dimitris Sacharidis, Alkis Simitsis, and Timos Sellis, *Senior, IEEE*

Abstract—As the Web is increasingly used not only to find answers to specific information needs but also to carry out various tasks, enhancing the capabilities of current Web search engines with effective and efficient techniques for Web service retrieval and selection becomes an important issue. Existing service matchmakers typically determine the relevance between a Web service advertisement and a service request by computing an overall score that aggregates individual matching scores among the various parameters in their descriptions. Two main 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 Information Retrieval similarity measures up to semantic logic-based inference rules. Second, the reduction of individual scores to an overall similarity leads to significant information loss. Determining appropriate weights for these intermediate scores requires knowledge of user preferences, which is often not possible or easy to acquire. Instead, using a typical aggregation function, such as the average or the minimum of the degrees of match across the service parameters, introduces undesired bias, which often reduces the accuracy of the retrieval process. Consequently, several services, e.g., those having a single unmatched parameter, may be excluded from the result set, while being potentially good candidates. In this work, we present two complementary approaches that overcome the aforementioned deficiencies. First, we propose a methodology for ranking the relevant services for a given request, introducing objective measures based on dominance relationships defined among the services. Second, we investigate methods for clustering the relevant services in a way that reveals and reflects the different trade-offs between the matched parameters. We demonstrate the effectiveness and the efficiency of our proposed techniques and algorithms through extensive experimental evaluation on both real requests and relevance sets, as well as on synthetic scenarios.

Index Terms—Web services matchmaking, ranking, clustering, skyline.

1 Introduction

Web services are software entities that have a well-defined interface and perform a specific task. Typical examples include services returning information to the user, such as news or weather forecast services, or services altering the world state, such as on-line shopping or booking services. A Web service is formally described in a standardized language (WSDL). The service description may include the names and types of input and output parameters, preconditions and effects, as well as Quality of Service (QoS) attributes, such as price, execution time, availability, and reputation.

As Web services and service providers proliferate, there will be a large number of candidate, and likely competing, services for fulfilling a desired task. Hence, effective service discovery mechanisms are required for identifying and retrieving the most appropriate services. Assume the existence of a repository that contains a large number of *advertised* service descriptions. In a typical scenario, a user provides a complete definition of the *requested* service, and issues a discovery query. The repository, then, employs a *matchmaking* algorithm to identify services relevant to the user's request. Note that, perfect matches, i.e., services with the same description as the request, are seldom found. Furthermore, even when a perfect match exists, it may not constitute the most desirable option, e.g., the service is currently unavailable. For these reasons, given a request, the matchmaking algorithm needs to consider a potentially large number of partial matches, and to select the best candidates among them.

To effectively deal with the large number of candidates, it is imperative that the identified matches are provided in a useful form. For this purpose, we examine two distinct methods: *ranking* and *clustering*. Ranking entails assigning a score to each advertisement, quantifying its suitability for the given request. Given that users typically view only a few top search results, it is important that useful results appear high in the list. Similarly, in fully automated scenarios, where a software agent automatically selects and composes services to achieve a specific task, only the few top ranked results are typically considered. On the other hand, clustering organizes advertisements so that services within a cluster provide similar matches with respect to the request. Since several parameters are involved in the matchmaking process, finding a service that provides a high degree of match for *all* parameters is difficult; instead, it is often needed to decide between different trade-offs. Clustering the search results allows the user to identify an interesting advertisement and then browse similar results, i.e., those found within the same cluster.

- *D. Skoutas is with the L3S Research Center, Hannover, Germany. skoutas@l3s.de*
- *D. Sacharidis is with the Institute for the Management of Information Systems, Greece, and the Hong Kong University of Science and Technology, Hong Kong. dsachar@imis.athena-innovation.gr*
- *A. Simitsis is with the HP Labs, Palo Alto, USA. alkis@hp.com*
- *T. Sellis is with the Institute for the Management of Information Systems, and the National Technical University of Athens, Greece. timos@imis.athena-innovation.gr*

Both ranking and clustering require as a first step the matchmaking algorithm to assign to each considered parameter a *parameter degree of match* (PDM) with respect to the requested service. Then, a *service degree of match* (SDM) can be computed as an aggregate of the individual PDMs. Various approaches for combining PDMs exist. One direction is to assign weights to individual scores [1], determined through user feedback. Appropriate weights are chosen either by assuming a-priori knowledge of the user's preferences or by applying machine learning techniques. In the lack of such information, methods are often pessimistic, computing the overall service similarity based on the lower degree of match among parameters. However, considering an aggregate SDM is inappropriate for ranking or clustering the matchmaking results, since a single criterion is a very coarse metric for evaluating composite entities, such as services. Inevitably, these approaches lead to large information loss as the PDMs of a service may vary considerably. SDMs obscure the inherent trade-offs offered by different advertisements. Consequently, SDM-based ranking may assign very low ranks, for example, to services with a single bad matching parameter. Similarly, SDM-based clustering fails to construct representative groups. In this paper we argue that *an effective Web service retrieval algorithm should explicitly take into account all PDMs*. We refer to this requirement as $\mathcal{R}_1$.

PDMs provide a finer granularity for the discovery and selection process. Still, an open issue is the lack of a single method that objectively determines the optimal degree of match between parameters. In fact, there are two general directions for assessing the match among parameters. The first treats parameter descriptions as documents and employs basic Information Retrieval techniques to extract keywords [1]. Subsequently, a string similarity measure is used to compute the PDM. The second follows the Semantic Web vision. Service descriptions are enriched by annotating their parameters with semantic concepts taken from domain ontologies [2]. Then, estimating the PDM reduces to a problem of logic inference, i.e., using a reasoner to check for equivalence or subsumption relationships between the corresponding concepts. More recently, hybrid techniques have also been proposed [3].

Both directions share their strengths and weaknesses. Keyword-based matchmaking fails to properly identify and extract semantics, since service descriptions are typically short documents with few terms. On the other hand, semantic-based matchmaking is hindered by 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 services. Depending on numerous factors, such as the application domain or the user's preferences, the most suitable method for assessing PDMs can be any of the above. Based on these observations, we advocate ranking and clustering that *simultaneously leverage multiple criteria for estimating the PDMs*. We refer to this requirement as $\mathcal{R}_2$.

In a recent work, we consider the problem of ranking Web service search results [4]. In this paper, we propose a unified framework for ranking and clustering Web services, satisfying requirements $\mathcal{R}_1$ and $\mathcal{R}_2$. The core concept in the proposed methodology is *multi-criteria service dominance*. Intuitively, an advertised service S_i *dominates* another S_j, with respect to a given request, if S_i has higher PDMs compared to S_j *in all parameters and according to all criteria*. The notion of dominance is inherently objective: if S_i dominates S_j, then S_i will have a higher overall SDM, independently of the PDM criteria and aggregation method used.

Clearly dominating services are rare to find because of the trade-offs offered by the advertisements. Rather, most services will only be good matches in some parameters while bad in others. Further, even for a particular parameter, different criteria may provide conflicting degrees of match. Table 1 illustrates this with an example. For simplicity, we assume all services have one input P_{in} and one output P_{out} parameter, and that there exist four advertisements, A, B, C, D. Furthermore, for the given request, three different matching filters (e.g., different string similarity measures), m_1, m_2, and m_3, are applied, resulting in the PDMs shown in Table 1. Observe that under any criterion, service A constitutes the best match with respect to both parameters. Hence, A *dominates* B, C, and D. However, there is no clear winner among the other three. For instance, according to m_1, B is definitely better than D. On the other hand, m_2 suggests that B has a lower match degree for the input parameter but a higher for the output.

TABLE 1
Example services with multiple criteria PDMs

Service	Parameter	m_1	m_2	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

To handle the inherent ambiguity in dominance relationships among advertised services, our methodology adapts the notion of *uncertain dominance* from [5]. Briefly, a service dominates another with a probability that depends on multiple criteria PDMs. Then, advertisements are ranked or clustered with respect to their uncertain dominance relationships. The main contributions of our work are summarized as follows.

1. We propose a method for determining dominance relationships among service advertisements that simultaneously takes into consideration multiple PDM criteria.
2. We introduce the problem of ranking Web services based on dominance relationships and discuss efficient algorithms.
3. We introduce the problem of clustering Web services based on dominance relationships and discuss efficient algorithms.
4. We extensively evaluate our approaches in terms of retrieval effectiveness, using real requests and relevance sets, and in terms of efficiency, using synthetic scenarios.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 formally defines dominance scores for comparing Web service search results. Then, Sections 4 and 5 describe, respectively, efficient algorithms for ranking and clustering the search results. Section 6 presents our experimental study. Finally, Section 7 concludes the paper.

2 Related Work

In this section we discuss related work regarding Web service discovery, skylines, data fusion, and search results clustering.

Web Service Discovery. Current industry standards for Web service description and discovery (WSDL, UDDI) focus mainly on *keyword-based* matching. Integrating multiple external matching services to a UDDI registry is proposed in [6]. A matching service is selected based on specified policies (e.g., the first available, or the most successful). If more than one are invoked, the union or the intersection of the results is returned.

To overcome the limitations of keyword-based search, several methods propose *semantic-based* approaches that use ontologies to enhance the service descriptions (WSDL-S, OWL-S, WSMO) and address the matchmaking as a logic inference task [2]. In [7] and [8], the similarity between requested and offered inputs and outputs is assessed by comparing classes in an associated domain ontology. In [9], the matching of requested and offered parameters is treated as matching bipartite graphs. Ontologies, user profiles, and query expansion or relaxation, are used in [10]. Further, hybrid matchmakers OWLS-MX [3] and WSMO-MX [11] exist for OWL-S and WSMO services, respectively. In a different direction, [12] proposes *content-based* matching and focuses on data-intensive Web services. Specifically, it probes the candidate service and measures the relevance based on the actual data returned, rather than on the metadata in the service description.

These works focus on matching pairs of parameters from the requested and offered services, while the overall SDM is typically calculated as a weighted average, assuming the existence of an appropriate weighting scheme. Furthermore, they do not consider more than one matching criteria. Based on the diversity of these approaches, it is evident that there is no single matching criterion that constitutes the silver bullet for the problem. Therefore, the approach proposed in this paper provides an efficient method that leverages multiple matching criteria (e.g., keyword, semantic, or content based) and service parameters, with no loss of information from aggregating, and without requiring a-priori knowledge of the user's preferences.

There are several works that employ clustering during the service discovery process. Clusters are used in [1] to group names of parameters into semantically meaningful concepts. Different types of similarity are then combined using a linear function, with manually assigned weights. Learning the weights from user feedback is suggested for future work. In [13], clustering quickly filters out irrelevant services for a given request. Then, the remaining services are matched at the concept level, where concepts are extracted from the request and the advertisements using probabilistic latent semantic analysis. The main difference of these works to our approach is that we do not apply clustering to identify relevant services, but to group search results according to the different trade-offs with respect to the request parameters. Users can then easily drill down to those results better capturing their needs.

Skylines. In this work we use concepts of multi-objective optimization, or *skyline queries* [14]. Given a set of points, the skyline is defined as the subset of points that are not dominated by any other, i.e., for which no objectively better point exists. The BNL algorithm [14] iterates over the data set, comparing each point with every other, and reports those not dominated. SFS [15] improves on BNL, by pre-sorting the input according to a monotone scoring function F, reducing the number of dominance checks. SaLSa [16] proposes an efficient termination condition for SFS. Other works, [17], [18], exploit index structures to speed-up the skyline computation process.

Even though our work exploits the basic techniques underlying the discussed methods, they cannot be directly applied to our problem for several reasons. First, the index-based methods work only for static data. In our setting, the underlying data depend on the matching scores and thus an index would have to be rebuilt for each query. Second, none of the above methods consider ranking or clustering. However, as we discuss shortly, some recent algorithms investigate these issues. Third, multiple matching criteria are not considered. To address this last concern, our work borrows ideas from the probabilistic skyline model for uncertain data [5], which however fails to rank or cluster the skyline.

Regarding ranking of the skyline, we note that the importance of combining top-k queries with skyline queries has been pointed out in [19]. However, some important differences to our work exist. 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. Third, it does not address the requirement for handling multiple matching criteria. To deal with large skylines, [20] relaxes the notion of dominance to k-dominance, and defines top-δ dominant skyline queries, which, return a set of *at least* δ points. Finally, [21] relies on users to specify additional preferences among points so as to control the result size.

Regarding clustering of the skyline, the k most representative skyline operator is proposed in [22]. This selects a set of k skyline points, so that the number of points dominated by them is maximized. On the other hand, [23] tries to select the k skyline points that best capture the trade-offs among the parameters. Our work builds upon the latter, as it defines a more intuitive notion of representativeness. Still none of this techniques deal with multiple criteria.

Data Fusion. Given a set of ranked lists of documents returned by different search engines, *data fusion* is the construction of a single ranked list combining the individual rankings. Data fusion techniques are classified based on whether they require knowledge of the relevance scores and whether training data is used [24]. The simplest method based solely on the documents' ranks is the Borda-fuse model [24]. It assigns as score to each document the summation of its rank (position) in each list. Training data can be used to assess the performance of each source and, hence, learn its importance. The Condorcet-fuse method [25] is based on a majoritarian voting algorithm, which specifies that a document d_1 is ranked higher in the fused list than another document d_2 if d_1 is ranked higher than d_2 more times than d_2 is ranked higher than d_1. An outranking approach was recently presented in [26]. 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.

When the relevance scores are available, other fusion techniques, including CombSUM, CombANZ and CombMNZ, can be applied [27]. In CombSUM, the fused relevance score of a document is the summation of the scores assigned by each source. In CombANZ (CombMNZ), the final score of a document is calculated as that of CombSUM divided (multiplied) by the number of lists in which the document appears. In [28], the author concludes that CombMNZ provides the best retrieval efficiency. Seen in this context, our work addresses a novel problem where in each rank a *vector of scores*, instead of a single, measures the relevance for each data item.

When training data is available, it is shown in [29] 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 [30] for the fusion of multilingual ranked lists. The optimal size of the training data that balances effectiveness and efficiency is investigated in [31].

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 [32], such a technique was shown to outperform CombMNZ.

Search Results Clustering. Most existing Web search engines return a ranked list of documents as a result to a user query. Some works propose a post-processing step that thematically groups similar documents into clusters. This facilitates quick exploring of the results and locating interesting items. Typical examples are the algorithms proposed in Grouper [33] and Lingo [34]. Commercial applications have also appeared recently, such as Vivisimo (http://www.vivisimo.com) and Grokker (http://www.grokker.com). Our idea for clustering the retrieved services to reveal trade-offs among PDMs is inspired by these works. Note, however, that this line of research is not applicable in our case, as it typically operates on text snippets returned by the search engine. To the best of our knowledge, this is the first work to discuss clustering matched services considering multiple matching criteria.

3 Service Dominance Scores

This section introduces the notion of Web service dominance and discusses related concepts used in the following for service ranking and clustering. To abstract away from a particular representation and to simplify the discussion, we model a Web service operation as a function that receives a number of inputs and returns a number of outputs. Other parameters, such as pre-conditions, effects, QoS, can be incorporated seamlessly.

Assume a service request Q with a set of input $\mathcal{P}_{in}(Q)$ and output $\mathcal{P}_{out}(Q)$ parameters. We write $Q.P_j$ to denote the j-th input parameter, where $P_j \in \mathcal{P}_{in}(Q)$. Further, let S be an advertised service with input and output parameters $\mathcal{P}_{in}(S)$ and $\mathcal{P}_{out}(S)$, respectively. Note that S can be a match to Q, even when their parameter sets' cardinalities differ, e.g., an advertisement that produces more outputs than requested.

Consider a matching function m_i assessing the PDM, i.e., the degree of match among pairs of parameters. m_i produces scores in the range $[0, 1]$, where higher values correspond to better matches. Given a request Q, the *match instance* of a service S under the m_i criterion to be a vector s_i such that

$$s_i[j]=\begin{cases}\max\limits_{P_k\in\mathcal{P}_{in}(S)}\{m_i(S.P_k,Q.P_j)\},\ \forall j: P_j\in\mathcal{P}_{in}(Q)\\ \max\limits_{P_k\in\mathcal{P}_{out}(S)}\{m_i(S.P_k,Q.P_j)\},\forall j: P_j\in\mathcal{P}_{out}(Q).\end{cases} \quad (1)$$

The vector s_i has a total of $d = |\mathcal{P}_{in}(Q)| + |\mathcal{P}_{out}(Q)|$ entries that correspond to the input and output parameters of the request. Intuitively, each entry quantifies how well the corresponding parameter of the request is matched by the advertisement S. Clearly, an input (output) parameter of Q can only match with an input (output) parameter of S.

Let $\mathcal{M}$ be a set of matching functions. Given a request Q and an advertisement S, each $m_i \in \mathcal{M}$ results in a distinct match instance. We refer to the set of instances, as the *match object* of service S. In the following, we abuse notation by representing the advertisement and its match object with the same uppercase letter (e.g., S); further, we use the terms match object and service interchangeably. We reserve lowercase letters for match instances of the corresponding service (e.g., s_1, s_2, etc.). The notation $s_i \in S$ implies that instance s_i corresponds to the match vector, i.e., service, S.

We denote as $\mathcal{I}$ the set of all match instances, given a set of advertised services $\mathcal{S}$ and a set of matching functions $\mathcal{M}$. Since a match instance can be seen as a d-dimensional point, we use the notion of dominance from [14] to describe relationships among match instances. Given two instances $u, v \in \mathcal{I}$, we say that u *dominates* v, denoted by $u \succ v$, iff u has higher or equal PDM in all parameters (or dimensions) and strictly higher PDM in at least one compared to v's, i.e.,

$$u \succ v \quad \Leftrightarrow \quad \forall i\ u[i] \geq v[i] \wedge \exists j\ u[j] > v[j]. \quad (2)$$

If u is neither dominated by nor dominates v, then u and v are incomparable. Notice that the notion of dominance leverages the PDMs of all parameters, without performing any aggregation; hence, it satisfies the requirement $\mathcal{R}_1$.

Figure 1 draws the three match instances for each service of Table 1, as a two-dimensional point in the $P_{in} \times P_{out}$ space. For example, a_1 corresponds to the PDMs of service A under criterion m_1 and, hence, has coordinates $(0.96, 0.92)$. According to the definition of dominance, instances that are in the top right corner constitute better matches. As an example, notice that d_1 dominates b_3 and c_3; similarly, it is dominated by b_1 and c_1, as well as by all the instances of A; finally, it neither dominates nor is dominated by b_2 and c_2.

Under a single matching function, i.e., a single instance, the dominance relationship among services is unambiguous. However, when multiple matching functions are applied, resulting in different PDMs, each service is represented by a set of $M = |\mathcal{M}|$ instances. Consequently, dominance relationships between services become uncertain. This work quantifies uncertainty using probabilities according to the model proposed in [5]. In the following, 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

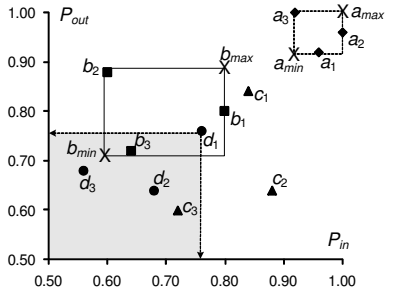


Fig. 1. Services of Table 1 in the $P_{in} \times P_{out}$ space

equal importance; however, it is straightforward to extend the approach to the case that different weights are assigned to each matching criterion. Then, a service U *dominates* another service V with a probability:

$$Pr[U \succ V] = \frac{1}{M^2} \sum_{u \in U} \sum_{v \in V} |u \succ v| \quad (3)$$

where $|u \succ v|$ is 1 if $u \succ v$ and 0 otherwise. Intuitively, the probability that U dominates V measures the number of instances of V dominated by each instance of U.

Based on the notion of uncertain service dominance, we next present three distinct definitions for assessing when a service is objectively interesting, which are then used for ranking and clustering. These definitions take into consideration multiple criteria, hence satisfy the requirement $\mathcal{R}_2$.

Service Skyline Score. Interesting services are those not dominated by any other with high probability. Given a match instance u, we define the *skyline score* of u as $u.sky = \prod_{V \neq U}(1 - \frac{1}{M} \sum_{v \in V} |v \succ u|)$. Then, the following equation measures the probability of a service U not being dominated, termed *service skyline score*, or skyline probability in [5].

$$U.sky = \frac{1}{M} \sum_{u \in U} u.sky \quad (4)$$

Service Dominated Score. Given a match instance u, we define the *dominated score* of u as $u.dds = \frac{1}{M} \sum_{V \neq U} \sum_{v \in V} |v \succ u|$. Hence, $u.dds$ considers the instances that dominate u. The dominated score of a service U is defined as the (possibly weighted) average of the dominated scores of its instances:

$$U.dds = \frac{1}{M} \sum_{u \in U} u.dds = \sum_{V \neq U} Pr[V \succ U]. \quad (5)$$

The dominated score of a service indicates the average number of services that dominate it. Hence, a *lower* dominated score indicates a better match result.

Service Dominating Score. Given a match instance u, we define the dominating score of u as $u.dgs = \frac{1}{M} \sum_{V \neq U} \sum_{v \in V} |u \succ v|$. Thus, $u.dgs$ considers the instances that u dominates. The dominating score of a match object U is defined as the (possibly weighted) average of the dominating scores of its instances:

$$U.dgs = \frac{1}{M} \sum_{u \in U} u.dgs = \sum_{V \neq U} Pr[U \succ V]. \quad (6)$$

The dominating score of a service indicates the average number of services that it dominates. Hence, a *higher* dominating score indicates a better match result.

As an example, consider the match object C with instances c_1, c_2 and c_3 depicted 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; hence, its scores are $c_1.dds = 1$ and $c_1.dgs = 5/3$. Similarly, we compute $c_2.dds = 1$, $c_2.dgs = 1/3$, and $c_3.dds = 5/3$, $c_3.dgs = 0$. Therefore, $C.dds = 11/9$ and $C.dgs = 6/9$. Further, we observe that all the instances of A dominate all the instances of C. Thus, $Pr[A \succ C] = 1$, and $C.sky = 0$.

4 Ranking Web Services

Section 4.1 introduces the problem of ranking Web services, and Sections 4.2, 4.3 and 4.4 describe our proposed solutions, based on dominance relationships.

4.1 Problem Statement

The problem of ranking Web services entails computing the scores of services and returning the top-k highest ranking ones. All the scores defined in Section 3 satisfy the requirements $\mathcal{R}_1$ and $\mathcal{R}_2$, and therefore qualify as possible ranking scores. However, the skyline score, due to the product involved in Equation 4, can be too restrictive: if a service S_i is completely dominated (i.e., with probability 1) by just a single other service S_j, then this is sufficient to completely disqualify S_i from the result set, as its skyline score would be 0 (recall the last example in Section 3). 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 using the skyline score.

Based on the above discussion, we choose to rank services based on the dominated and dominating scores. A straightforward algorithm is the following. For each instance u of a service 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 results, simply sort them according to the score in the counter. However, the applicability of this approach is limited by its large computation cost, as it needs to compute the score for all services, even those which are not in the top-k. Observe that independently of k, the algorithm exhaustively performs all possible dominance checks among instances.

In the following, we propose algorithms that address this issue by establishing lower and upper bounds for the dominated/dominating scores. This essentially allows us to (dis-)qualify objects to (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. Next, we delve into some useful properties of the dominance relationship that help us prune the search space.

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., $v.dgs \leq u.dgs$.*

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

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., $u \succ v_{max} \Rightarrow u \succ v\ \forall v \in V$.

4.2 Ranking by Dominated Score

The first algorithm, hereafter referred to as 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.

The algorithm maintains three lists, $\mathcal{I}_{min}$, $\mathcal{I}_{max}$, 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.

Since 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 $\mathcal{I}$. 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 2, proceeds in the following six steps.

Algorithm 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 $\mathcal{R}$.

```
1  begin
2    Initialize R = ∅;  ddsMax = ∞;  minValue = -1 ;
3    for U ∈ 𝒰 do
4      (u_min, u_max) ← calculate min and max bounding instances ;
5      I_min ← insert u_min ordered by F(u_min) desc. ;
6      I_max ← insert u_max ordered by F(u_max) desc. ;
7      for u∈U do  I ← insert u ordered by F(u) desc. ;
8    for u_max∈I_max do
9      if |R| = k then
10       if F(u_max) ≤ minValue then return R;
11     U.dds = 0 ;
12     for v_min ∈ I_min^F(u_max) do
13       if v_min ≻ u_max then
14         U.dds = U.dds + 1 ;
15         if (U.dds + V.dds) ≥ ddsMax then
16           for u∈U do  u.dds = U.dds ;
17           skip U;
18     U.dds = 0 ;
19     for v ∈ I^F(u_max) do
20       if v ≻ u_max then
21         U.dds = v.dds + 1/M ;
22         if (U.dds + v.dds) ≥ ddsMax then
23           for u∈U do  u.dds = U.dds ;
24           skip U;
25     U.dds = 0 ;
26     for u∈U do
27       for v_min ∈ I_min^F(u) do
28         if v_min ≻ u then
29           u.dds = u.dds + 1/M ;
30           if (U.dds + u.dds + V.dds) ≥ ddsMax then
31             U.dds = U.dds + u.dds + V.dds ;
32             skip U;
33       U.dds = U.dds + u.dds + V.dds ;
34     U.dds = 0 ;
35     for u∈U do  u.dds = 0 ;
36     for u∈U do
37       for v ∈ I^F(u) do
38         if v ≻ u then
39           u.dds = u.dds + 1/M² ;
40           if (U.dds + u.dds + v.dds) ≥ ddsMax then
41             U.dds = U.dds + u.dds + v.dds ;
42             skip U;
43       U.dds = U.dds + u.dds + v.dds ;
44     if |R| = k then remove the last result from R ;
45     R ← insert U ordered by dds asc.
46     if |R| = k then
47       U_k ← the k-th object in R ;
48       ddsMax = U_k.dds;
49       minValue = min_{i=1}^{M} (U_{k_min}[i]);
50   return R;
51 end
```

Fig. 2. Algorithm TKDD

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}.

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.

4.3 Ranking by Dominating Score

The TKDG algorithm 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 task is more time consuming compared to that of 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 $u.dds$, 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 $u.dgs$, TKDG must perform in the worst case $|\mathcal{I}|-pos(u)$ checks, i.e., those after u. 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 TKDG is significantly larger than TKDD's. Furthermore, TKDD allows for more efficient pruning, since 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.

Algorithm 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)) / M^2, 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
```

Fig. 3. Algorithm TKDG

The 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 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 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)*.

4.4 Ranking by Dominance Score

This section introduces a ranking score that achieves a balance between dominated and dominating scores.

Service Dominance Score. Given an instance u, we define the dominance score of u as $u.ds = u.dgs - \lambda \cdot u.dds$, which 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 6.1) 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:

$$U.ds = \frac{1}{M} \sum_{u \in U} u.ds \tag{7}$$

Next, we outline the TKM algorithm, that computes the top-k matches with respect to the dominance score. In particular, this algorithm is derived by the algorithm 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 to $U.dgs^- = -\lambda \cdot \sum_{u \in U} \frac{pos(u)}{M^2}$ 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. 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 Clustering Web Services

Section 5.1 introduces the problem of clustering Web service search results, based on dominance relationships. Sections 5.2 and 5.3 present two algorithms to address this problem.

5.1 Problem Statement

The purpose of clustering is fundamentally different from that of ranking. Instead of selecting the k best matches, we organize the matched services into ℓ groups that capture different trade-offs among all the considered request parameters. To better motivate and illustrate this, we refer to a simple example with two matching filters, one input P_{in} and one output P_{out} parameter, and a set of matched services, A through I. Each service comprises two match instances, represented as points in the $P_{in} \times P_{out}$ plane shown in Figure 4; for example, service A consists of instances a_1 and a_2.

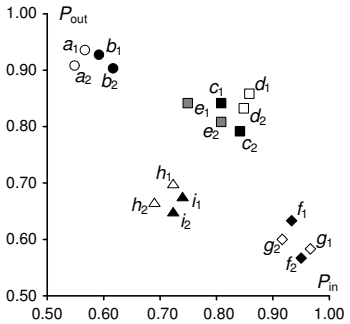


Fig. 4. Example services in the $P_{in} \times P_{out}$ space

Upon inspection of Figure 4, one can deduce that the matched services can be organized in 4 clusters: $\{A, B\}$, $\{C, D, E\}$, $\{F, G\}$, $\{H, I\}$. The first cluster, $\{A, B\}$, contains services that match the output parameter almost perfectly (with PDMs above 0.90), but fail to match the input parameter. Conversely, the third cluster, $\{F, G\}$, corresponds to services that match the input but not the output parameter. The services in the second cluster, $\{C, D, E\}$, match both parameters quite well (with PDMs between 0.75 and 0.90), but not as well as services in the first and third cluster match the output and input parameters, respectively. Finally, the fourth cluster, $\{H, I\}$, consists of services that match both parameters with average degrees of match (PDMs between 0.60 and 0.75).

Clearly, the last cluster does not contain any useful matches, as both H and I are dominated by C, D and E with probability 1 (all matching criteria agree). However, all others contain useful candidates that capture different trade-offs w.r.t. the request parameters. For example, a user interested in finding the best service might look in the second cluster; indeed, it contains the top-1 result, D, according to any algorithm of

our ranking framework (Section 4). Consider now a user that requires almost exact matches for either or both parameters, and hence is not satisfied with the services in the second cluster. That user would have to consider service composition opportunities in the first and third clusters. For example, a service, say A, in the first cluster matches the output parameter perfectly and can thus be combined with another, say X, to obtain the desired service. Therefore, to identify such an X, the user issues a new query where the requested service has as input that of the original request and as output A's input. Alternatively, the user may invoke a similar process for a service, say F, that belongs to the third cluster. S/he issues a new query where the requested service has as input F's output and as output that of the original request.

The benefits of clustering the matched services become even more apparent, when QoS parameters are considered in the matchmaking process. A typical example would be a query containing requirements for price and execution time. Then, one cluster would contain services that are expensive but have a fast response time, while another cluster contain services with low or no fee but high execution time. Then, different types of user, e.g., premium versus custom users, would focus on different clusters of results to find a good match.

Overall, the benefit of the clustering is that it diversifies the search results, allowing the users to focus and drill-down on that subset of the results that better meets their requirements.

The proposed Web service clustering framework effectively summarizes the various trade-offs (e.g., as in clusters $\{A, B\}$, $\{C, D, E\}$, $\{F, G\}$) associated with the multiple matching parameters, while eliminating irrelevant services (e.g., cluster $\{H, I\}$). The framework is based on dominance relationships, and, thus, satisfies the requirements $\mathcal{R}_1$ and $\mathcal{R}_2$. More specifically, it comprises the following high-level steps:

1. Select the services that have a skyline probability above a specified threshold p.
2. Select ℓ representative services from the above set.
3. Form the clusters by assigning each of the remaining services to its closest representative.

In the following, we focus on steps 1 and 2. Step 3 is straightforward, as it uses the distance function between two services defined already in step 2.

Regarding the first step, we choose to set the probability threshold to 0, so that the derived set will contain those matches that have a non-zero probability to belong to the skyline. Thus, intuitively, these are the "most interesting" objects with respect to the existing trade-offs, from which we can then select the ℓ representatives to be used as the "seeds" for forming the clusters. This choice has also an additional benefit in terms of efficiency, as shown by the next property.

Property 5. *A match object U has a skyline probability greater than 0 if and only if there exists at least one instance u for which there does not exist another object V such that v_{min} dominates u, i.e., $U.sky > 0$ iff $\exists u$ s.t. $\nexists V$ s.t. $v_{min} \succ u$.*

The proof can be easily derived from Equation 4, combined with Property 4. The above property also specifies the process for computing the 0-skyline, which is performed by the function Skyline Representative Candidates (SRC) shown in Figure 5. This function iterates over all the match objects, and selects those satisfying the above property.

```
Function SRC
    Input: A set of objects 𝒰, each comprising M instances.
    Output: The set of objects 𝒮 having non-zero skyline score.
 1  begin
 2      Initialize 𝒮 = ∅;
 3      for U ∈ 𝒰 do
 4          for u ∈ U do
 5              add = true;
 6              for V ∈ 𝒰 ∖ {U} do
 7                  if v.min ≻ u then
 8                      add = false;
 9                      break;
10              if add == true then
11                  Insert U in 𝒮;
12                  break;
13      return 𝒮;
14  end
```

Fig. 5. Function SRC

Returning to the example of Figure 4, the first step eliminates services H and I, as they have 0 skyline probability. To see this, we need to examine the two instances of service H. For both h_1 and h_2 there exists an object, C (either C, D or E would do) such that c_{min} dominates them. Because of this dominance relationship, H (and I) cannot be a better match than C, and is thus excluded from the next steps.

In the second step, we need to select the most representative objects among those returned by the SRC function. Two recent works, [22] and [23], address this issue, however applying only to conventional dominance relationships. The first tries to find cluster centers that have high dominating scores. For the example of Figure 4 and for $\ell = 3$, [22][1] would choose C, D and E as the three most representative services, missing important information. On the other hand, the second approach uses a distance-based metric to obtain a uniform sample of the skyline that explicitly captures all trade-offs, and therefore is suitable for the purpose of our clustering. Continuing the example of Figure 4 and for $\ell = 3$, [23][1] would choose A (or B), C (or D, E) and F (or F) as the three most representative services. Next, we extend this method to account for the existence of multiple instances per object.

The work in [23] introduced a distance-based definition for selecting the ℓ most representative skyline objects: it is a subset of the skyline $\mathcal{S}$ such that for every other skyline object there is a representative close to it. Given $\mathcal{S}$, we define the *representation error* of any subset $\mathcal{C}$ of $\mathcal{S}$ as

$$Er(\mathcal{C}) = \max_{U \in \mathcal{S} \setminus \mathcal{C}} \{\min_{V \in \mathcal{C}} dist(U, V)\}. \quad (8)$$

In our case, we define the distance $dist(U, V)$ between U and V as the median among all distances between pairs of instances of U and V. We choose the median, instead of the average, to reduce the bias of instances that are very far apart. Hence, the goal is to find the set of representatives $\mathcal{C}$ that minimizes the representation error. This problem is NP-hard, even for a single criterion [23]. In the following, we propose two algorithms that offer different trade-offs between processing cost and accuracy. Approximate Skyline Clustering

1. We apply [22] and [23] assuming only one instance per service.

Algorithm ASC

```
   Input: A set of objects 𝒰, each comprising M instances;
          The desired number of clusters ℓ.
   Output: The ℓ representative objects for clustering.
 1 begin
 2   𝒮 = SRC (𝒰);
 3   Select a random U from 𝒮;
 4   Initialize 𝒞 = {U}; Remove U from 𝒮;
 5   for i = 0 to ℓ − 1 do
 6     maxdist = 0;
 7     for U ∈ 𝒮 do
 8       mindist = ∞;
 9       for V ∈ 𝒞 do
10         i = 0;
11         for u ∈ U do
12           for v ∈ V do
13             d[i] = dist(u, v);
14             i++;
15         sort(d); median = d[size(d)/2];
16         if median < mindist then
17           mindist = median;
18       if mindist > maxdist then
19         maxdist = mindist;
20         U_max = U;
21     Insert U_max into 𝒞; Remove U_max from 𝒮;
22   return 𝒞;
23 end
```

Fig. 6. Algorithm ASC

Algorithm HSC

```
   Input: A set of objects 𝒰, each comprising M instances;
          The desired number of clusters ℓ;
          The partitioning method part to use.
   Output: The ℓ representative objects for clustering.
 1 begin
 2   𝒮 = SRC (𝒰);
 3   Initialize 𝒞 = ∅;
 4   for U ∈ 𝒮 do
 5     Compute the centroid u_ctr of all instances u ∈ U;
 6     Insert u_ctr in ℒ;
 7   Sort ℒ using a space-filling curve;
 8   if part == 0 then
 9     // equi-width partitioning;
10     Let addr(ℒ[1]), addr(ℒ[N]) be the addresses of the first and last
       objects;
11     step = (addr(ℒ[N]) − addr(ℒ[1])) /ℓ;
12     repr = addr(ℒ[1]) + step/2;
13     for u_ctr ∈ ℒ do
14       if addr(u_ctr) > repr then
15         Insert U into 𝒞;
16         repr = repr + step;
17   else
18     // equi-depth partitioning;
19     step = N/ℓ;
20     repr = step/2;
21     for i = 0 to ℓ do
22       Insert ℒ[⌊repr⌋] into 𝒞;
23       repr = repr + step;
24   return 𝒞;
25 end
```

Fig. 7. Algorithm HSC

(ASC), described in Section 5.2, is an extension of the 2-approximate algorithm of [23] and offers tight accuracy guarantees with a relatively large processing cost. On the other hand, Heuristic Skyline Clustering (HSC), described in Section 5.3, is a novel linear time heuristic, which is experimentally shown to produce good representatives.

5.2 Approximate Skyline Clustering

As mentioned above, the problem of selecting the ℓ representatives that minimize the representation error is NP-hard. The authors in [23] propose a greedy algorithm that provides a 2-approximate solution. Briefly, the algorithm selects initially a random object; then, it proceeds iteratively in $\ell - 1$ steps, selecting at each iteration the *farthest neighbor*, i.e., the object with the largest distance from its closest representative.

Figure 6 shows the adaptation of this algorithm for the case of multiple instances. First, function SRC is invoked to return an initial candidate set $\mathcal{S}$ with non-zero skyline scores (*line* 2), as discussed in the previous section. A random object is selected and inserted into the representative set $\mathcal{C}$ (*lines 3–4*). In each of the following $\ell - 1$ iterations (*lines 5–21*) the object $U \in \mathcal{S}$ (*lines 7–20*) with the largest distance from its closest representative $V \in \mathcal{C}$ (*lines 9–17*) is selected and inserted in the representative set (*line 21*). The distance between U and V is calculated as the median of the distances between pairs of instances of U and V (*lines 10–15*). Notice that the ASC algorithm produces a representative set which has a representation error at most twice as much as the optimal. The proof is analogous to that of [23], with the difference that the distance measure simultaneously takes into account the multiple instances.

5.3 Heuristic Skyline Clustering

Although ASC produces a representative set with provable guarantees, it is still a computationally expensive process, as shown by the experiments in Section 6. In the following, we introduce the Heuristic Skyline Clustering (HSC) algorithm, shown in Figure 7, which is a fast heuristic with no quality guarantees, though. The basic idea of HSC is to avoid expensive farthest neighbor searches, by first sorting the match objects and then retrieving objects at predefined positions in the sorted list, which are expected to be good representatives.

Initially, HSC invokes function SRC to obtain a candidate set of objects $\mathcal{S}$ with non-zero skyline scores; also, the representative set $\mathcal{C}$ is initialized to empty (*lines 2–3*). The key aspect of HSC lies in arranging the objects in an appropriate order for partitioning. To preserve the proximity among objects, we apply a space filling curve, such as the z-order or the Hilbert curve. The basic property of space filling curves is that they arrange multi-dimensional points in a single-dimensional space, i.e., a line, so that points that are close to each other in the original space are very likely to remain close in the line. Recall that in our case each object $U \in \mathcal{S}$ is a set of M points in the d-dimensional space. Thus, we need to represent each object by a single point. We choose the *centroid*, defined as the barycenter of all the object's instances. The centroids and the Hilbert curve are shown in Figure 8. Each centroid u_{ctr} is inserted into a list $\mathcal{L}$, and is associated with an address $addr(u_{ctr})$, which denotes its position in the space filling curve; then, all the centroids are sorted using this address as the key (*line 4–7*). In our example, this order coincides with the alphabetical.

Using the sorted list, there are two ways to extract heuris-

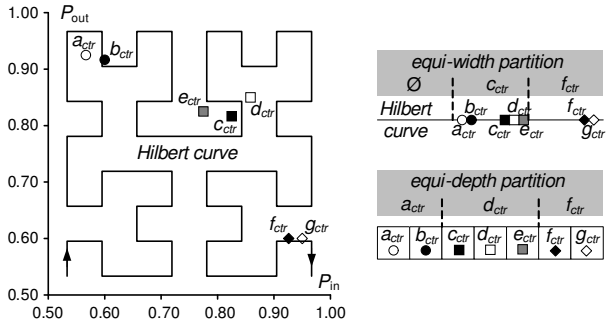


Fig. 8. Partitions based on the Hilbert curve

tically good representatives, *equi-width* and *equi-depth* partitioning. The intuition of equi-width partition is to divide the single-dimensional space into a set of ℓ partitions of equal width and select the centroid in the middle of each partition as a representative. Specifically, the first and last addresses, i.e., addr($\mathcal{L}$[1]) and addr($\mathcal{L}$[N]), are used to define the width, termed $step$, of each partition (*lines 10–11*). The address of the first desired representative is determined as that of middle of the first partition (*line 12*). Then, each centroid is examined in turn. If the address of the current centroid u_{ctr} is after the desired representative, the object U is inserted in the result set $\mathcal{C}$, and the address of the next representative is computed (*lines 13–16*). Upon conclusion of this process, a set of representatives is placed in the result set $\mathcal{C}$. However, there is no guarantee that $\mathcal{C}$ will contain ℓ objects, as there may not exist a representative in each partition, e.g., in the case of skewed or sparse data sets. Figure 8 shows the case of $\ell = 3$ for the services of Figure 4, where the first partition is empty; thus the representatives are only C, F.

Alternatively, an equi-depth partitioning can be applied. This is guaranteed to return exactly ℓ representatives, essentially by generating more representatives (i.e., more clusters) in more dense areas. The process is similar to a recently proposed heuristic for maintaining ℓ-medoid clusters in dynamic settings [35]. The selected representatives lie in positions within the list $\mathcal{L}$ that are $\lfloor N/\ell \rfloor$ entries apart (*line 19*), with the first one being in the $\lfloor N/(2\ell) \rfloor$-th position (*line 20*). Figure 8 shows that for $\ell = 3$ the representatives are A, D, F.

6 Experimental Evaluation

This section presents an extensive experimental study of our approach. First, we investigate the benefits of our methods with respect to the recall and precision of the computed results and the error in identifying the most representative services. For this purpose, we rely on a publicly available benchmark for Web service discovery, comprising real-world service descriptions, sample requests, and relevance sets. Then, we consider the computational cost of the proposed algorithms under different combinations of values for the parameters involved, using synthetic data sets. All the algorithms were implemented in Java, and the experiments were conducted on a Pentium D 2.4GHz with 2GB of RAM, running Linux.

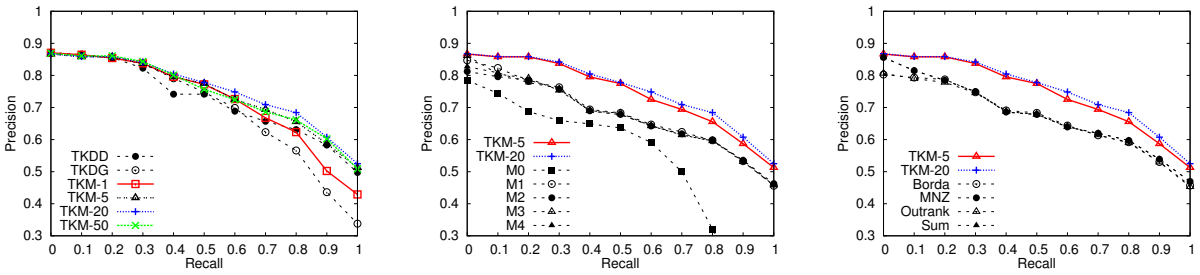
6.1 Retrieval Effectiveness

Experimental setup. We use the publicly available service retrieval test collection OWLS-TC v2 (http://www-ags.dfki.uni-sb.de/~klusch/owls-mx/), which 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.

To determine the PDMs of the services, we use the OWLS-MX matchmaker [3], which matches I/O parameters 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 provided: loss-of-information measure (M1), extended Jaccard similarity coefficient (M2), cosine similarity (M3), and Jensen-Shannon information divergence based similarity (M4).

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 to a match object, and the score vectors correspond to the object's d-dimensional match instances.

Ranking. The conducted evaluation of the algorithms described in Section 4 comprises four stages. First, we compare the three different ranking criteria introduced in our approach, in terms of the *Interpolated Recall-Precision Averages* [36], which measure precision, i.e., percentage of retrieved items that are relevant, at various recall levels, i.e., after a certain percentage of all the relevant items have been retrieved. The resulting precision-recall graphs are depicted in Figure 9(a). Regarding TKM, we study the effect of the parameter λ (see Section 3), considering 4 variations, denoted as TKM-λ, for λ=1, 5, 20, 50. As shown in Figure 9(a), 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 TKDG starts to degrade at a considerably higher rate compared to that of 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 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 TKM performs better than 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 TKM-λ_H, i.e., TKM having λ=λ_H, produces better results than the other two methods. This is illustrated by the graph TKM-5 in Figure 9(a). In addition, the experiments show that for values of λ lower than λ_H, TKM does not produce better results; i.e., the effect of dds is still not sufficient. On the other hand, we can get further



(a) TKDD, TKDG and TKM (b) TKM and individual measures M0-M4 (c) TKM and fusion approaches

Fig. 9. Ranking: Precision-Recall graphs

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 TKM for values of λ around 20, which, as demonstrated by the graph in Figure 9(a), produces slightly better precision than 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 TKDD, as illustrated by the TKM-50 graph in Figure 9(a).

Next, we examine the resulting benefit of the dominance-based ranking compared to applying either of the individual similarity measures M0-M4. The precision-recall measures are illustrated in Figure 9(b). To avoid overloading the figure, only the TKM-5 and 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 [37].

In the third stage, to better gauge the effectiveness of our methodology, we compare with more 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 [27], the simple rank-based method of Borda-fuse [24], and the Outranking approach [26]. The first three techniques are parameter-free; for the last, we employ a single outranking relation that corresponds to Pareto-optimality. The obtained precision-recall graphs are shown in Figure 9(c). Again, our approach clearly outperforms the other methods. This gain becomes even more apparent, when noticing through Figures 9(b) and 9(c) that these fusion techniques, in contrast to our approach, fail to demonstrate a significant improvement over the individual similarity measures.

Finally, we evaluate the ranking effectiveness using in addition the following standard IR evaluation measures [36]:

- *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.

Table 2 details the IR metrics for all compared methods. For each metric, the highest value is shown in bold (we treat the values of both versions of TKM uniformly), whereas the second highest in italic. In summary, TKDD and TKDG produce an average gain of 8.33% and 6.44%, respectively, with respect to the other approaches. Additionally, TKM-5 and TKM-20 improve the quality of the results by a percentage (average values) of 11.44% and 12.56%, respectively.

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

Clustering. The main challenge in the clustering problem is to select a good set of ℓ representative services, so that each one of the remaining matched services is sufficiently close (and, hence, effectively clustered) to one of these representatives. Thus, the quality of this process is quantified by the value of the representation error (see Equation 8).

Figure 10 displays the representation error for our two algorithms, ASC and HSC, averaged over the 28 requests in the collection OWLS-TC v2. As shown, ASC, which relies on farthest neighbor searches, is more effective in selecting good representatives. This is because HSC relies instead on the use of the space filling curve to order and then partition the objects. Although the space filling curve tries to preserve the proximities among objects, the dimensionality reduction from d to 1 inevitably introduces more error. Still, for a small number of clusters (up to 3), we notice that the difference in the representation error is small. This is because ASC starts with a randomly selected object as the first representative. Thus, for small values of ℓ the representation error is affected by this selection. Moreover, as will be shown in Section 6.2, HSC is typically much faster than ASC. Therefore, ASC appears to be also an appealing technique for clustering Web

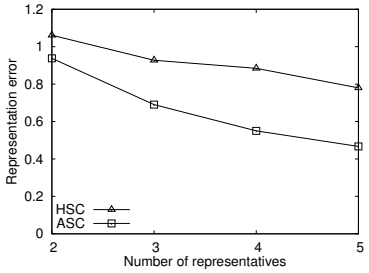


Fig. 10. Clustering: representation error

service search results.

6.2 Computational Cost

Experimental setup. We use a publicly available generator (http://randdataset.projects.postgresql.org/) to create data sets for studying the computational cost of our ranking and clustering algorithms. The examined parameters and their corresponding values are summarized in Table 3. N, k, and ℓ refer, respectively, to the number of available services, the number of top results, and the number of clusters to return. The number of parameters in the service request, i.e., the dimensionality of each match instance, is denoted with d. M corresponds to the number of distinct matching criteria employed by the service matchmaker, i.e., the number of instances per match object. Given a similarity measure, $corr$ denotes the correlation among the degrees of match for the parameters of a service and takes 3 values. 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% ($high$). In each round of the experiments, we investigate the effect of one parameter, while setting 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 or 3 clusters among a set of 5K partially matching service descriptions, using 4 different similarity measures.

TABLE 3
Parameters and examined values

Parameter	Symbol	Values
Number of services	N	[1, 10]K, **5K**
Number of top results	k	10, 20, **30**, 40, 50
Number of clusters	ℓ	2, **3**, 4, 5
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

Ranking. We first provide a theoretical analysis, and then we report our experimental findings on the synthetically generated data sets described above.

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 PDMs for all d parameters. As a result, the complexity 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 briefly discuss the computational cost of the fusion techniques considered in Section 6.1. 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 scores to a single overall. In practice, the performance of TKDG and TKM resembles that of the Outranking method, while TKDD's is similar to the other fusion approaches. For ease of presentation, in the remaining, we focus only on the three proposed methods.

Experimental Results. Overall, the experiments indicate that TKDD is the most efficient method. As already discussed in Section 4.3, 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 TKDG is significantly larger, so its delay is expected. Similarly, the performance of TKM suffers mainly due to the impact of dgs score; therefore, it is reasonable that it follows the same trend as TKDG, with a slight additional overhead for accounting for the dds score, as well. These observations are more apparent in Figure 11(a), where it can be seen that TKDD is very slightly affected, as opposed to TKDG and TKM, by the size of the data set. Another interesting observation refers to the effect of the dimensionality (Figure 11(d)), which at higher values becomes noticeable even for 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, more dominance checks need to be 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 11(e). For correlated data sets, where many successful dominance checks occur, the computational cost for all methods drops

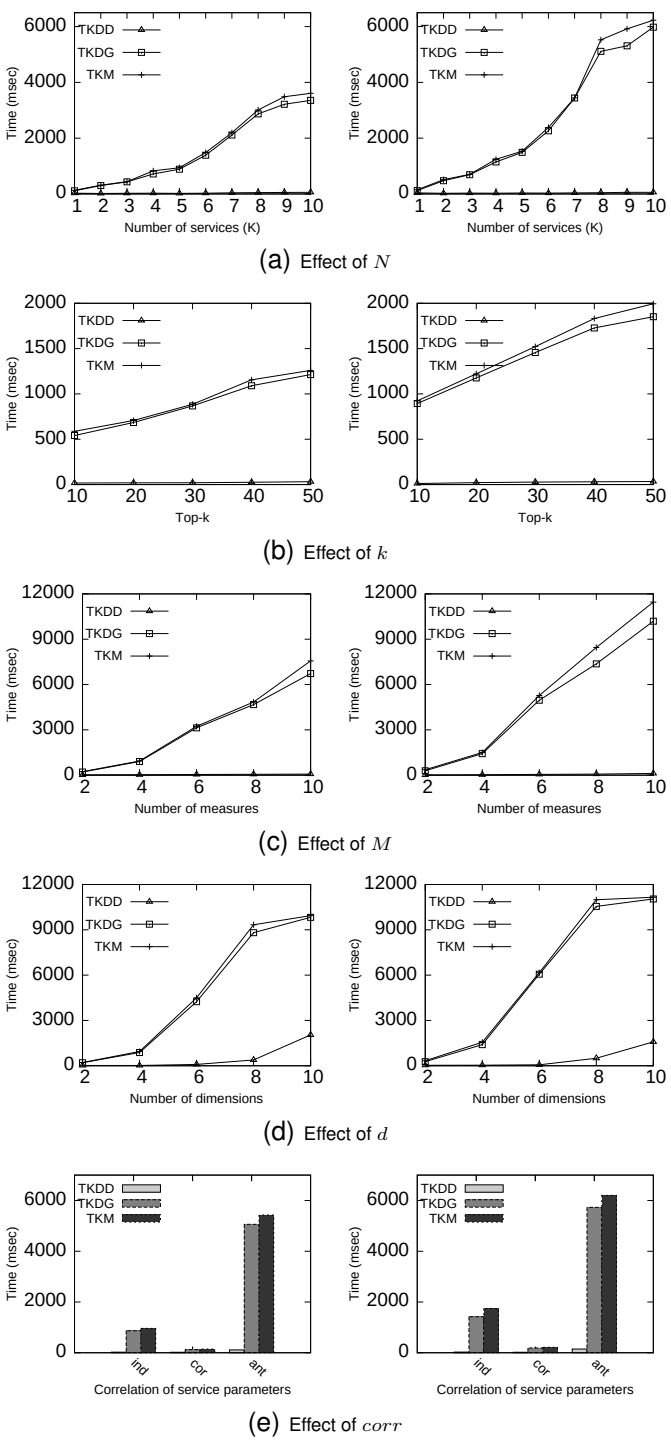


Fig. 11. Ranking: low (left) and high (right) var

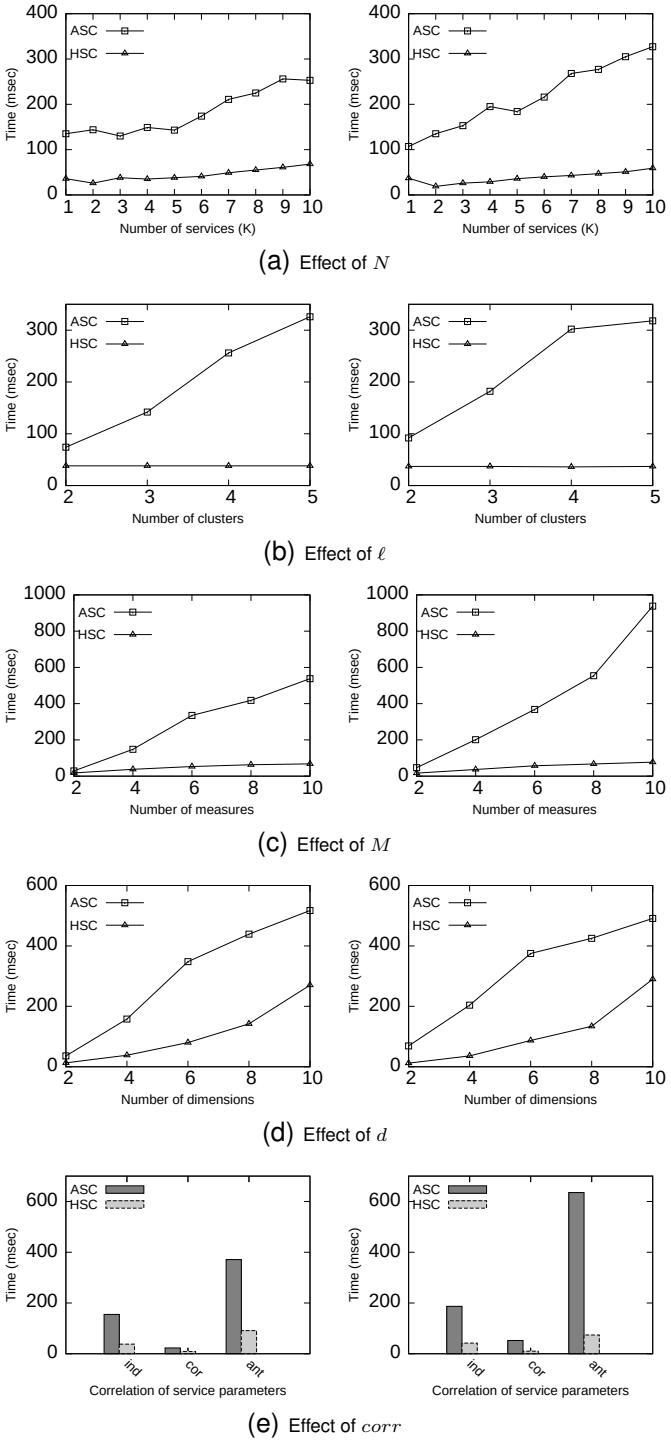


Fig. 12. Clustering: low (left) and high (right) var

close to zero. On the contrary, for anti-correlated data sets, few dominance checks are successful and the computational cost is significantly larger.

Therefore, 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 TKM seems as an excellent solution. If the time factor acts as the driving decision point, then TKDD should be favored, since it provides high quality results (see Figure 9) almost instantly (see Figure 11).

Clustering. Next, we measure the performance of the algorithms ASC and HSC on the same synthetic data sets discussed above, starting first with a brief theoretical analysis and then presenting the experimental findings.

Theoretical Analysis. Both algorithms, ASC and HSC, first rely on function SRC to compute the set of objects $\mathcal{S}$ with non-zero skyline score. Recall that this computation is based on Property 5. Thus, similar to the corresponding analysis for the ranking algorithms, the complexity of this process is $O(dN^2M)$, since for each pair of match objects U and V, all

instances u of U need to be checked against v_{min} of V. Then, ASC proceeds with selecting the ℓ representatives by means of farthest neighbor searches, where the distance between two objects is computed as the median of the distances of their instances. The cost for each search is $O(d|\mathcal{S}|M^2)$. Hence, the complexity of ASC is $O(dN^2M + \ell d|\mathcal{S}|M^2)$. On the other hand, HSC only scans $\mathcal{S}$ once, partitioning it according to the ordering imposed by the space filling curve. Hence, the complexity of HSC is $O(dN^2M + |\mathcal{S}|)$.

Experimental Results. Figure 12 presents the experimental results for the algorithms ASC and HSC, using the data sets synthetically generated according to the parameters in Table 3. In general, all experiments indicate that HSC is more efficient than ASC. The main reason for this performance is the expensive farthest neighbor searches that ASC uses, whereas HSC is accelerated by the heuristic described in Section 5.3. In particular, Figure 12(a) demonstrates that HSC is more robust with respect to the number of services for both low and high variance. On the other hand, ASC demonstrates a lower performance, which is slightly worse for high variance. Similar trends appeared when we experimented with the number of clusters (Figure 12(b)). For both low and high variance ASC performs much worse, while HSC reveals a robust performance. Figure 12(c) examines the case of varying the number of instances, where HSC scales considerably better than ASC. Figure 12(d) portrays the effect of dimensionality. Similar to the case of ranking, the processing time of clustering increases with dimensionality, which is a known challenge faced also by all skyline computation approaches. Finally, Figure 12(e) demonstrates the behavior of both techniques for different distributions of the degrees of match of service parameters. Both algorithms perform well, except for the case of anti-correlated data distributions, where ASC presents significant delays. Essentially, an anti-correlated distribution implies a larger number of objects with non-zero skyline score, and therefore an increased cost for the farthest neighbor searches.

Summarizing, the choice depends on the desired trade-off between accuracy and efficiency. ASC produces more accurate results, whereas HSC provides a more balanced solution between accuracy and efficiency.

7 Conclusions

In this paper, we have addressed ranking and clustering of Web service search results and proposed methods based on the notion of *dominance*, which apply multiple matching criteria without aggregating the match scores of individual service parameters. We have presented three algorithms for ranking the search results, and two algorithms for selecting the most representative services for clustering, so that the produced clusters reflect the trade-offs between the matched parameters. An extensive experimental evaluation validates the effectiveness and efficiency of our approach.

References

[1] X. Dong, A. Y. Halevy, J. Madhavan, E. Nemes, and J. Zhang, "Similarity Search for Web Services," in *VLDB*, 2004, pp. 372–383.

[2] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara, "Semantic Matching of Web Services Capabilities." in *ISWC*, 2002, pp. 333–347.

[3] M. Klusch, B. Fries, and K. P. Sycara, "Automated Semantic Web service discovery with OWLS-MX," in *AAMAS*, 2006, pp. 915–922.

[4] D. Skoutas, D. Sacharidis, A. Simitsis, V. Kantere, and T. K. Sellis, "Top-k Dominant Web Services under Multi-criteria Matching," in *EDBT*, 2009, pp. 898–909.

[5] J. Pei, B. Jiang, X. Lin, and Y. Yuan, "Probabilistic Skylines on Uncertain Data," in *VLDB*, 2007, pp. 15–26.

[6] J. Colgrave, R. Akkiraju, and R. Goodwin, "External Matching in UDDI," in *ICWS*, 2004, p. 226.

[7] J. Cardoso, "Discovering Semantic Web Services with and without a Common Ontology Commitment," in *IEEE SCW*, 2006, pp. 183–190.

[8] D. Skoutas, A. Simitsis, and T. Sellis, "A Ranking Mechanism for Semantic Web Service Discovery," in *IEEE SCW*, 2007, pp. 41–48.

[9] U. Bellur and R. Kulkarni, "Improved Matchmaking Algorithm for Semantic Web Services Based on Bipartite Graph Matching," in *ICWS*, 2007, pp. 86–93.

[10] W.-T. Balke and M. Wagner, "Cooperative Discovery for User-Centered Web Service Provisioning," in *ICWS*, 2003, pp. 191–197.

[11] F. Kaufer and M. Klusch, "WSMO-MX: A Logic Programming Based Hybrid Service Matchmaker," in *ECOWS*, 2006, pp. 161–170.

[12] J. Caverlee, L. Liu, and D. Rocco, "Discovering and Ranking Web Services with BASIL: A Personalized Approach with Biased Focus," in *ICSOC*, 2004, pp. 153–162.

[13] J. Ma, Y. Zhang, and J. He, "Efficiently Finding Web Services Using a Clustering Semantic Approach," in *CSSSIA*, 2008, p. 5.

[14] S. Börzsönyi, D. Kossmann, and K. Stocker, "The Skyline Operator," in *ICDE*, 2001, pp. 421–430.

[15] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with Presorting," in *ICDE*, 2003, pp. 717–816.

[16] I. Bartolini, P. Ciaccia, and M. Patella, "Efficient sort-based skyline evaluation," *ACM TODS*, vol. 33, no. 4, pp. 1–45, 2008.

[17] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive Skyline Computation in Database Systems," *ACM TODS*, vol. 30, no. 1, pp. 41–82, 2005.

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

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

[20] 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*, 2006, pp. 503–514.

[21] W.-T. Balke, U. Güntzer, and C. Lofi, "Eliciting Matters - Controlling Skyline Sizes by Incremental Integration of User Preferences," in *DASFAA*, 2007, pp. 551–562.

[22] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang, "Selecting Stars: The k Most Representative Skyline Operator," in *ICDE*, 2007, pp. 86–95.

[23] Y. Tao, L. Ding, X. Lin, and J. Pei, "Distance-based representative skyline," in *ICDE*, 2009, pp. 892–903.

[24] J. A. Aslam and M. H. Montague, "Models for Metasearch," in *SIGIR*, 2001, pp. 275–284.

[25] M. H. Montague and J. A. Aslam, "Condorcet Fusion for Improved Retrieval," in *ACM CIKM*, 2002, pp. 538–548.

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

[27] E. A. Fox and J. A. Shaw, "Combination of Multiple Searches," in *2nd TREC, NIST*, 1993, pp. 243–252.

[28] J.-H. Lee, "Analyses of Multiple Evidence Combination," in *SIGIR*, 1997, pp. 267–276.

[29] C. C. Vogt and G. W. Cottrell, "Fusion Via a Linear Combination of Scores," *Information Retrieval*, vol. 1, no. 3, pp. 151–173, 1999.

[30] 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*, 2005, pp. 121–130.

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

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

[33] O. Zamir and O. Etzioni, "Grouper: A Dynamic Clustering Interface to Web Search Results," *Computer Networks*, vol. 31, no. 11-16, pp. 1361–1374, 1999.

[34] S. Osinski, J. Stefanowski, and D. Weiss, "Lingo: Search Results Clustering Algorithm Based on Singular Value Decomposition," in *Intelligent Information Systems*, 2004, pp. 359–368.

[35] S. Papadopoulos, D. Sacharidis, and K. Mouratidis, "Continuous Medoid Queries over Moving Objects," in *SSTD*, 2007, pp. 38–56.

[36] R. A. Baeza-Yates and B. A. Ribeiro-Neto, *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.
[37] M. Klusch and B. Fries, "Hybrid OWL-S Service Retrieval with OWLS-MX: Benefits and Pitfalls," in *SMRR*, 2007.

Dimitrios Skoutas is a Postdoctoral researcher at the L3S Research Center, Germany. He received his Diploma on Electrical and Computer Engineering and his PhD in Computer Science from the National Technical University of Athens (NTUA) in 2003 and 2008, respectively. He has worked as a researcher at the Knowledge and Database Systems Lab at NTUA and at the Institute for the Management of Information Systems (IMIS). His research interests focus on semantic web services and web information retrieval.

Dimitris Sacharidis is a Marie Curie Postdoctoral Fellow at the Institute for the Management of Information Systems, Greece, and at the Hong Kong University of Science and Technology. He received his BSc degree from the National Technical University of Athens, his MSc degree from the University of Southern California, and his PhD degree in Computer Science from the National Technical University of Athens. His research interests include data streams, privacy, security, and ranking in databases.

Alkis Simitsis is a researcher in the Intelligent Information Management Lab at Hewlett Packard. He obtained his Diploma on Electrical and Computer Engineering and his PhD in Computer Science from the National Technical University of Athens (NTUA) in 2000 and 2004, respectively. He has worked as a PostDoc researcher at the Computer Science group at IBM's Almaden Research Center, and he was a research visitor at Infolab at Stanford University. His research interests include data warehouses, ETL processes, query processing, keyword search, and web services.

Timos Sellis is the Director of the Institute for the Management of Information Systems (IMIS) and a Professor at the Computer Science Division of the National Technical University of Athens (NTUA), Greece. He received his Diploma in Electrical Engineering from NTUA, his MSc degree from Harvard University, and his PhD from the University of California at Berkeley, where he was a member of the INGRES group. He was an Associate Professor at the Department of Computer Science of the University of Maryland, College Park. He has received the Presidential Young Investigator award for 1990-1995 and the VLDB 1997 10 Year Paper Award for his work on spatial databases. He was the president of the National Council for Research and Technology of Greece (2001-2003) and a member of the VLDB Endowment (1996-2000). His research interests include data streams, peer-to-peer databases, data warehouses, the integration of Web and databases, and spatio-temporal databases.