International Journal of Global Engineering (IJGE) E- ISSN: 2456-3099

VOL 3 ISSUE 2 (2018) PAGES 175 - 185

Received: 20/02/2018. Published: 22/03/2018

COMPUTATION OF SKYLINE FOR AMALGAMATED SERVICE SELECTION USING BOTTOM UP ALGORITHM

S. Sashikumar, S. Srilekaa, A. Rachel Roselin
Department of Computer Science and Engineering
Dhanalakshmi Srinivasan Institute of Technology, Samayapuram, T.N, India

Abstract—Service composition is emerging as an effective vehicle for integrating existing Web services to create value-added and personalized composite services. As Web services with similar functionality are expected to be provided by competing providers, a key challenge is to find the "best" Web services to participate in the composition. When multiple quality aspects (e.g., response time, fee, etc) are considered, a weighting mechanism is usually adopted by most existing approaches, which requires users to specify their preferences as numeric values. We propose to exploit the dominance relationship among service providers to find a set of "best" possible composite services, referred to as a composite service skyline. We develop efficient algorithms that allow us to find the composite service skyline from a significantly reduced searching space instead of considering all possible service compositions. We propose a novel bottom-up computation framework that enables the skyline algorithm to scale well with the number of services in a composition. We conduct a comprehensive analytical and experimental study to evaluate the effectiveness, efficiency, and scalability of the composite skyline computation approaches.

Index Terms—Service composition, skyline, bottom-up, dominance analysis, quality of service.

1 INTRODUCTION

One of the fundamental objectives of service computing is to enable interoperability amongst different software and data applications running on a variety of platforms. The introduction of Web services has been key for the paradigm shift in business structures allowing them to outsource required functionality from third-party Web-based providers through service composition [21], [35]. The fast-growing number of Web services will result in a significant number of Web services with similar functionalities. This also poses a new challenge for service composition: selecting proper service providers that achieve a composition with the best user desired Quality of Web Service (QoWS).

A number of service selection approaches have been developed that depend on the computation of a predefined objective function [34], [31], [30], [23], [35]. A weighting mechanism is leveraged where users express their preference over different (and sometimes conflicting) quality parameters as numeric weights. The composition gaining the highest value from the objective

International Journal of Global Engineering (IJGE) E- ISSN: 2456-3099
VOL 3 ISSUE 2 (2018) PAGES 175 - 185
Received: 20/02/2018. Published: 22/03/2018

function will be selected and returned to the user. There are two major limitations to these approaches. First, it is a rather demanding task for users to transform their personal preferences into numeric weights. Without a detailed knowledge about the QoWS from all possible compositions, users may not have the ability to make a precise tradeoff decision between different quality aspects using numbers. Second, whenever the weights are changed, a completely new search needs to be performed. An exhaustive search would be computationally expensive because the possible compositions will increase exponentially with the number of services involved [31].

1.1 Motivation

We propose to exploit the dominance relationship among service providers to find a set of "best" possible service compositions, referred to as a composite service skyline. We will use an example to motivate key ideas. Example 1: (Composing Services). Consider the development of a composite Web service, TravelAssistant, which provides travel assistance services for users. Typical Web services that would need to be accessed include TripPlanner, Map, and Weather. TripPlanner provides basic trip information, such as airlines, hotels, and local attractions. Other than this, users may also be interested to consult the city map and local transportations by accessing the Map service. The weather condition during the travel days is an important factor that makes the Weather service relevant. The developer may face a number of options for each of these services as there are multiple software vendors competing to offer similar functionalities. For example, Table 1 shows the five possible Map service providers and four possible Trip-Planner providers.

Accessing a Web service typically includes the invocation of a set of operations. For example, accessing a Map service requires to invoke two operations: Geocode and GetMap. There may be dependency constraints between these operations (e.g., GetMap depends on Geocode). Thus, these operations can be arranged into a sequence with respect to the dependency constraints, which is referred to as a Service Execution Plan (SEP). The QoWS of a SEP are computed by aggregating those of its member service operations using a set of predefined aggregation functions [35], [31].

International Journal of Global Engineering (IJGE) E- ISSN: 2456-3099
VOL 3 ISSUE 2 (2018) PAGES 175 - 185
Received: 20/02/2018. Published: 22/03/2018

TABLE 1
Map and TripPlanner Providers

sid	Operation	Latency	Fee	Reputation
Map Providers				
A	GeoCode	0.5	0.8	2
	GetMap	1	0	2
B	GeoCode	0.7	0.3	3
	GetMap	2	0.5	3
C	GeoCode	0.5	0.2	2
	GetMap	1.5	0.9	2
D	GeoCode	0.3	0.7	2
	GetMap	1	0.4	2
E	GeoCode	0.6	0.7	3
	GetMap	0.8	0.5	3
TripPlaner Providers				
F	SearchTrip	2	0	2
	GetTrip	2	0.8	2
G	SearchTrip	1	0	3
	GetTrip	2	1	1
H	SearchTrip	2	1	2
	GetTrip	3	1	2
I	SearchTrip	3	1	3
	GetTrip	2	0	2

referred to as a Service Execution Plan (SEP). The QoWS of a SEP is computed by aggregating those of its member service operations using a set of predefined aggregation functions [35], [31].

Example 2: (Service Dominance). Consider three QoWS parameters, latency, fee, and reputation, which capture the response time, monetary cost, and users' rating of service. Assume that a rating takes values in [1, 5] and a smaller value reflects a better rating. Latency and fee of a SEP are computed as the sum of those from its member operations. The reputation is set as the average of those from the member operations. The QoWS of the SEPs for the five possible map providers are (1.5, 0.8, 2), (2.7, 0.8, 3), (2, 1.1, 2), (1.3, 1.1, 2), and (1.4, 1.2, 3). For two providers p1 and p2, if SEP1 is as good as SEP2 in all QoWS aspects and better than SEP2 in at least one QoWS aspect, then SEP1 dominates SEP2, or denoted as SEP1 $\prec$ SEP2. Therefore, SEPA dominates both SEPB and SEPC, and SEPD dominates SEPE.

Since SEPA and SEPD are not dominated by any other providers, it is said that they are in the map service skyline. On the other hand, since SEPB, SEPC, and SEPE are dominated by SEPA and SEPD, respectively, they are not in the skyline. More formally, a service skyline or S-Sky can be regarded as a set of SEPs that are not dominated by others in terms of all user interested QoWS aspects, such as response time, fee, and reputation. The service providers in the skyline represent the best tradeoffs among different user interested quality aspects [29]. As another example, SEPF and SEPG constitute the Trip- Planner service skyline as highlighted in Table 1. Example 3: (Composite Service Skyline). Let us consider the composition (referred to as

International Journal of Global Engineering (IJGE) E- ISSN: 2456-3099

VOL 3 ISSUE 2 (2018) PAGES 175 - 185

Received: 20/02/2018. Published: 22/03/2018

TravelAssistance) of two services: TripPlanner and Map. A composite service execution plan or CSEP that consists of four operations (TripSearch, GetTrip, Geocode, GetMap) can be generated to access the composite service. The QoWS of a CSEP is computed by aggregating the QoWS of the SEPs in a similar way described in Example 2. A composite service skyline or C-Sky is a set of CSEPs that are not dominated by any other CSEPs. A naive way to find the

C-Sky of the TravelAssistance composition is to generate all 20 possible CSEPs and then check service dominance among them.

1.2 Computing the C-Sky

For a service composition with m services and assuming that there are n1,..., and nm providers for each of them services, Qm i=1 ni number of compositions need to be considered in order to find the C-Sky. For a complex composition with relatively large m (e.g., 10), a moderate number of providers for each service (e.g., 100) will introduce prohibitive overhead (e.g., 10010 compositions need to be considered). An intuitive solution that can significantly reduce the computational overhead is to compute the C-Sky from individual service skylines. This is due to a key observation: a C-Sky can be completely determined by only considering the SEPs from S-Skies.

In this case, only N = Qm i=1 ki maximum number of compositions need to be evaluated, where ki is the size of the ith S-Sky. Since the size of an S-Sky is typically much smaller than the number of providers, the computational cost can be reduced with several orders of magnitude. An example is given in Table 2. Instead of considering all 20 CSEPs, the C-Sky can be determined by only four candidate CSEPs, which are formed by combining the map and TripPlanner service skylines. Based on the above observation, we developed two preliminary algorithms in our recent work for computing the composite service skylines [33]. The first algorithm, which is referred to as One Pass Algorithm or OPA, performs a single pass on the N compositions and outputs the skyline. The second algorithm, which is referred to as Dual Progressive Algorithm or DPA, progressively reports the skyline. Nevertheless, both algorithms fail to generate the C-Sky for moderate-sized compositions (e.g., m = 5) within a reasonable time. Hence, a more efficient and scalable algorithm needs to be developed in order to cope with relatively complex compositions. We propose a novel Bottom-Up computation framework that enables to compute a C-Sky in a much more efficient fashion. We conduct an in-depth analysis of the preliminary service skyline algorithms with a focus on DPA. Although directly using DPA to compute a C-Sky is computationally expensive, DPA offers some key properties that inspire the design of a Bottom-Up Algorithm (BUA). BUA integrates a linear composition strategy into the proposed bottom-up framework to significantly boost the scalability of the algorithm, which makes it suitable to compute skylines for very complex service compositions.

International Journal of Global Engineering (IJGE) E- ISSN: 2456-3099

VOL 3 ISSUE 2 (2018) PAGES 175 - 185

Received: 20/02/2018. Published: 22/03/2018

TABLE 2
C-Sky Example

CSEP	Latency	Fee	Reputation
CSEP(A,F)	5.5	1.6	2
CSEP(A,G)	4.5	1.8	2
CSEP(D,F)	5.3	1.9	2
CSEP(D,G)	4.3	2.1	2

The analytical study shows that BUA is able to compute the C-Sky with nearly optimal time complexity. We conduct an extensive set of experiments to evaluate the performance of the C-Sky algorithms.

2 PROBLEM DEFINITION

Given m services with d user interested QoWS attributes, where the ith service has ni different providers, the problem of composite skyline computation is to compute the composite service skyline C-Sky over all possible service compositions.

It is worth to note that we do not consider candidate compositions with the different number of services. In another word, the C-Sky is computed only from the composite services that compose the fixed m services.

The problem can also be formulated in a more general sense as computing an Aggregate Skyline (or AS) over m source tables T1, ..., Tm, where each source table Ti has a set of columns Ci. Any two source tables Ti and Tj share a common set of columns, i.e., (Ci ∩ Cj) = {c1, ...cd}. For example, a travel agency database contains three tables that are used to store flight, hotel, and rental car information. Among other columns, all tables have three common columns, fee, service class, and user rating. When creating travel packages that include flight, hotel, and rental car, an aggregate table A is formed by aggregating the three source tables, where A(i).cj = fj(T1(i1).cj, ..., Tm(im).cj), ∀j ∈ [1, d]. A(i) is the i-th row of table A and cj is an aggregate column computed by aggregation function fj as discussed early in this section. More specifically, A is formed by performing a Cartesian product over the m source tables and aggregate columns of A are obtained by combining the matching columns in the source tables. An aggregate skyline consists of the rows in A, which are not dominated by any other rows on aggregate columns ci, ..., cd, i.e., A(i) ∈ AS if ∄A(j),A(j) ≺{c1,...,cd} A(i). For the travel agency example, computing such an aggregate skyline is instrumental to locate the most attractive travel packages for its customers.

A straightforward way to compute the C-Sky is to first materialize all possible compositions (or all rows in the aggregate table) and then apply the existing skyline algorithm to find the skyline. However, as discussed in the Introduction section, computing the C-Sky in such a brute force manner is computationally intensive. The following observation helps significantly improve performance.

International Journal of Global Engineering (IJGE) E- ISSN: 2456-3099

VOL 3 ISSUE 2 (2018) PAGES 175 - 185

Received: 20/02/2018. Published: 22/03/2018

LEMMA 1: (Local Search Strategy) Given m services S1, ..., Sm and the set of S-Skies SK1,...,SKm, computed for each of them, the C-Sky over S1, ..., Sm can be completely decided by SK1,...,SKm. PROOF: For a CSEP, ∈ C-Sky, assume that it consists of m SEPs, SEP1,...,SEPm, one from each service. Assume that each SEP is from the corresponding S-Sky respectively, except for SEPj , such that SEPj /∈ SKj . Thus, there must be a SEP′ j ∈ SKj such that SEP′ j ≺ SEPj. Therefore, we can find a CSEP ′ by replacing SEPj in with SEP′ j such that ′ ≺ . This contradicts the fact that is a skyline CSEP.

Lemma 1 enables us to compute the S-Sky for each service (i.e., perform a local search) and then compute the C-Sky by only considering the SEPs in the S-Skies. This lemma directly leads to the development of the One Pass Algorithm (OPA), which performs a single pass on the CSEP space with a size of N = Qm i=1 ki to compute the C-Sky. The Dual Progressive Algorithm (DPA) leverages an expansion lattice and a heap to progressively compute the skyline. As shown in both the analytical and experimental studies, the high computational complexity of DPA makes it impractical to compute the C-Sky with a large number of services. Nevertheless, DPA offers a nice theoretical underpinning to a much faster Bottom-Up Algorithm (BUA). BUA is built up a powerful bottom-up computational framework that exploits a linear composition strategy to achieve significantly better scalability and a nearly optimal time complexity. 3 RELATED WORK In this section, we give an overview of the existing works that are most relevant to the proposed service skyline algorithms.

Skyline computation has been intensively investigated in the database community [7], [28], [17], [24], [12], [14]. Block Nested Loops (BNL) and divide-and-conquer are among the first attempts to tackle the skyline computation problem [7]. BNL was extended by the Sort Filter Skyline (SFS) algorithm [12], which adopts a presorting scheme to improve efficiency. SFS was further improved by the Linear Elimination Sort for Skyline (LESS) [14]. Fewer exploits a small set of best data objects, referred to as an Elimination Filtering window (or EF window), to prune other objects in the initial pass of the external sorting. A special function is adopted in [4] that sorts the data points based on their minimum coordinate value, which avoids the scanning of the entire dataset. The sorting based algorithms can be used in conjunction with the local search strategy presented in Lemma 1 to compute composite service skylines. More specifically, we compute the S-Sky for each individual service and then instantiate the CSEPs from the S-Skies.

The CSEPs are then sorted to generate the C-Sky. Our extensive experiments demonstrate that the proposed algorithms are computationally much more efficient than the sorting based algorithms on computing composite service skylines. Index structures, such as B-tree [7], have also been leveraged to improve the performance of skyline analysis. Two index structures were presented in [28] with the ability to progressively report the skyline. NN and BBS are another two representative algorithms that can progressively process the skyline based on an R-tree structure [17], [24]. Since the composite services are generated dynamically, it is infeasible to pre-compute any index structures. This hinders us in exploiting the index-based approaches to compute the composite service skylines. As dimensionality increases, the size of skyline may become very large and easily overload the end-users. There are several key extensions on skyline

International Journal of Global Engineering (IJGE) E- ISSN: 2456-3099
VOL 3 ISSUE 2 (2018) PAGES 175 - 185
Received: 20/02/2018. Published: 22/03/2018

analysis aiming to make it more flexible and adapt to user's preferences. A novel fuzzy skyline concept is proposed in [15], where five lines of extensions are presented to "fuzzify" a skyline query to increase its flexibility and discrimination power. k-dominant skyline relaxes the idea of dominance to k-dominance. A point p is said to k-dominate another point q if there are k (≤ d) dimensions in which p is better than or equal to q and is better in at least one of these k dimensions [9]. A novel concept, called - dominant skyline is proposed in [5] based on fuzzy dominance. An - dominant skyline gives preference to services with a good compromise between QoS attributes. It also gives users the flexibility to control the size of the skyline. The fuzzy dominance, which signifies the degree to which p dominates q, leads to the definition of fuzzy dominating score [6]. This enables to rank-order candidate points and returns the top-k candidates to a user.

Skyline computation has also been extended to a distributed environment, where data points are stored, accessed, and processed in a distributed fashion [3]. A progressive distributed skyline algorithm was proposed in [19] that can progressively report the skyline points in a distributed environment. Constrained skyline queries are investigated in a large-scale unstructured distributed environment [11]. A partitioning algorithm is exploited to divide distributed data sites into incomparable groups, which allows efficient parallel skyline query processing. A novel feedback-driven mechanism is developed in [37] to minimize the network bandwidth when computing a skyline on a dataset that is horizontally partitioned onto geographically distant servers. Efficient skyline analysis techniques have also been developed in a Peer-to-peer (P2P) computing environment [13]. Jin et al. investigated the skyline operator on multi-relational databases [16].

The focus is on integrating efficient join methods into skyline computation based on the Primary Key and Foreign Key (PK-FK) relationship. Sun et al. studied a similar problem in the distributed environment [27]. Similar to [16], it also relies on the join attributes to prune candidate join tuples. The proposed C-Sky algorithms assume that Cartesian product is performed over multiple source tables. Hence, no PKFK relationship or join attributes can be leveraged to prune the searching space. Cartesian product typically results in a much larger candidate space, which makes the problem more challenging. Furthermore, both [16] and [27] assume a standard join operation, which does not generate any aggregate columns. In contrast, our algorithms essentially compute a skyline over aggregate columns obtained by combining the corresponding columns in the source tables.

4 ONE PASS ALGORITHM

We present the OPA algorithm in this section. During the single pass of the CSEP space, OPA enumerates the candidate CSEPs one by one and only stores the potential skyline CSEPs. It outputs the skyline after all the candidate CSEPs have been examined. OPA requires that all the S-Skies are sorted according to the scores of the SEPs. OPA works as follows (shown in Algorithm 1). It starts by evaluating the first CSEP (referred to as CSEP1) that is formed by combining the top SEPs from each S-Sky. It is guaranteed that CSEP1 ∈ C-Sky because CSEP1

International Journal of Global Engineering (IJGE) E- ISSN: 2456-3099

VOL 3 ISSUE 2 (2018) PAGES 175 - 185

Received: 20/02/2018. Published: 22/03/2018

has the minimum score so that no other CSEPs can dominate it. With the minimum score, CSEP1 is expected to have a very good pruning capacity. Thus, OPA puts CSEP1 on the top of the C-Sky so that the non-skyline CSEPs which are dominated by CSEP1 can be pruned at the earliest time. After this, OPA continues to enumerate all other CSEPs one by one. CSEPi will be inserted into the C-Sky if it is not dominated by any CSEP in C-Sky. Otherwise, the algorithm prunes CSEPi and starts to check CSEPi+1. During the checking process, whenever CSEPj ∈ C-Sky is dominated by CSEPi, C-Sky is updated by removing CSEPj.

Algorithm 1 One Pass Algorithm Input: m sorted S-Skies SK1, ..., SKm

Output: The C-Sky

1: N = Qm i=1 |SKi|; // number of candidate CSEPs

2: CSEP1 = Aggregate(SEP11,...,SEPm1);

3: C-Sky.add(CSEP1);

4: for all i ∈ [2,N] do

5: CSEPi = EnumerateNext(SK1,...,SKm);

6: IsDominated = False;

7: for all j ∈ [1, |C-Sky|] do

8: CSEPj = C-Sky.get(j);

9: if CSEPi.score < CSEPj.score then

10: if CSEPi ≺ CSEPj then

11: C-Sky.remove(j);

12: end if

13: else

14: if CSEPj ≺ CSEPi then

15: IsDominated = True;

16: break;

International Journal of Global Engineering (IJGE) E- ISSN: 2456-3099

VOL 3 ISSUE 2 (2018) PAGES 175 - 185

Received: 20/02/2018. Published: 22/03/2018

17: end if

18: end if

19: end for

20: if IsDominated == False then

21: C-Sky.add(CSEPi);

22: end if

23: end for

Bottom-Up Algorithm

As shown in last section, the performance of DPA is decided by two major factors: (F1) heap operation and (F2) skyline comparison. More specifically, F2 is bounded by O(N), where N = Qm i=1 ki, and N grows exponentially with the number of services. On the other hand, F1 is determined by Eq. (7). Our complexity analysis shows that the upper bound of the heap size also grows exponentially with the number of services due to Theorem 1. In this section, we present the BUA algorithm that is built up a novel bottom-up computation framework and exploits a linear composition strategy to gain significantly better efficiency and scalability. Furthermore, BUA also inherits all the nice properties of DPA, including progressive and pipelineable.

RESULT AND DISCUSSION

An interesting question here is whether we can use OPA instead of DPA in BUA. We choose DPA over OPA due to two major reasons. First, by using DPA, BUA is able to progressively report the C-Sky. Second, OPA relies on the Enumerate Next function, which is most effective only when the skylines are sorted. However, since the skylines generated by OPA are no longer sorted, we may need to sort each intermediate skyline if using OPA with BUA, which will introduce significant overhead. Also, since a sorting needs to be performed before the intermediate skyline can be used for the next-phase computation, BUA is no longer pipelineable. On the other hand, if DPA is used, all the intermediate skylines are automatically sorted and can be directly used for the next-phase computation. In addition, it is worth to note that DPA has a high computational complexity only when the number of services (i.e., m) is large, which is mainly due to the increase of the heap size. In BUA, we only use DPA to combine two skylines in each step. Due to Lemma 5, the heap size is bounded by k, where k is the size of the smaller skyline. In this case, DPA has a very similar performance with OPA, which has been demonstrated in the experiments.

[1] M. Alrifai and T. Risse. Combining global optimization with local selection for efficient qos-aware service composition. In WWW, 2009.
[2] M. Alrifai, D. Skoutas, and T. Risse. Selecting skyline services for qos-based web service composition. In WWW, 2010.
[3] W.-T. Balke, U. Guntzer, and J. X. Zheng. Efficient distributed skylining for web information systems. In EDBT, 2004.
[4] I. Bartolini, P. Ciaccia, and M. Patella. Efficient sort-based skyline evaluation. ACM Trans. Database Syst., 33(4):1–49, 2008.
[5] K. Benouaret, D. Benslimane, and A. HadjAli. On the use of fuzzy dominance for computing service skyline based on qos. In ICWS, 2011.
[6] K. Benouaret, D. Benslimane, A. HadjAli, and M. Barhamgi. Topk web service compositions using fuzzy dominance relationship. In IEEE SCC, 2011.
[7] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In ICDE, 2001.
[8] D. Chakerian and D. Logothetti. Cube Slices, Pictorial Triangles, and Probability. Mathematics Magazine, 64(4):219–241, 1991.
[9] 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.
[10] S. Chaudhuri, N. Dalvi, and R. Kaushik. Robust cardinality and cost estimation for skyline operator. In ICDE, 2006.
[11] L. Chen, B. Cui, and H. Lu. Constrained skyline query processing against distributed data sites. IEEE Trans. on Knowl. and Data Eng., 23:204–217, February 2011.
[12] J. Chomicki, P. Godfrey, J.Gryz, and D. Liang. Skyline with presorting. In ICDE, 2003.
[13] B. Cui, L. Chen, L. Xu, H. Lu, G. Song, and Q. Xu. Efficient skyline computation in structured peer-to-peer systems. IEEE Trans. on Knowl. and Data Eng., 21:1059–1072, July 2009.
[14] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In VLDB, 2005.
[15] A. Hadjali, O. Pivert, and H. Prade. On different types of fuzzy skylines. In ISMIS, 2011.
[16] W. Jin, M. Ester, Z. Hu, and J. Han. The multi-relational skyline operator. In ICDE, 2007.
[17] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In VLDB, 2002.
[18] J. Lee and S.-w. Hwang. Bskytree: scalable skyline computation using a balanced pivot selection. In EDBT, 2010.
[19] E. Lo, K. Y. Yip, K.-I. Lin, and D.W. Cheung. Progressive skylining over web-accessible databases. Data Knowl. Eng., 57(2):122–147, 2006.
[20] Y. Luo, X. Lin, and W. Wang. Spark: Top-k keyword query in relational databases. In SIGMOD, 2007.
[21] B. Medjahed and A. Bouguettaya. A multilevel composability model for semantic web services. IEEE Transactions on Knowledge and Data Engineering, 17(7), 2005.
[22] M. Morse, J. M. Patel, and H. V. Jagadish. Efficient skyline computation over low-cardinality domains. In VLDB, 2007.
[23] M. Ouzzani and B. Bouguettaya. Efficient Access to Web Services. IEEE Internet Computing, 37(3), March 2004.

[24] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In SIGMOD, 2003.
[25] J. Riordan. An Introduction to Combinatorial Analysis. John Wiley and Sons, Inc, New York, 1958.
[26] D. Skoutas, D. Sacharidis, A. Simitsis, and T. Sellis. Ranking and clustering web services using multicriteria dominance relationships. IEEE Transactions on Services Computing, 3:163–177, 2010.
[27] D. Sun, S. Wu, J. Li, and A. K. H. Tung. Skyline-join in distributed databases. In ICDE Workshops, pages 176–181, 2008.
[28] K. Tan, P. Eng, and B. Ooi. Efficient progressive skyline computation. In VLDB, 2001.
[29] Q. Wan, R. C.-W. Wong, I. F. Ilyas, M. T. Ozsu, and Y. Peng. Creating competitive products. In VLDB, 2009.
[30] Z. Xu, P. Martin, W. Powley, and F. Zulkernine. Reputationenhanced qos-based web services discovery. In ICWS, pages 249– 256, 2007.
[31] Q. Yu and A. Bouguettaya. Framework for Web Service Query Algebra and Optimization. ACM Trans. Web, 2(1), 2008.
[32] Q. Yu and A. Bouguettaya. Computing service skyline from uncertain qows. IEEE Transactions on Services Computing, 3(1):16– 29, 2010.
[33] Q. Yu and A. Bouguettaya. Computing service skylines over sets of services. In ICWS, pages 481–488, 2010.
[34] T. Yu, Y. Zhang, and K.-J. Lin. Efficient algorithms for web services selection with end-to-end qos constraints. ACM Trans. Web, 1(1):6, 2007.
[35] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Sheng. Quality-driven Web Service Composition. In WWW, 2003.
[36] S. Zhang, N. Mamoulis, and D. W. Cheung. Scalable skyline computation using object-based space partitioning. In SIGMOD, 2009.
[37] L. Zhu, Y. Tao, and S. Zhou. Distributed skyline retrieval with low bandwidth consumption. IEEE Trans. on Knowl. and Data Eng., 21:384–400, March 2009.