Versatile Size-l Object Summaries for Relational Keyword Search

Georgios J. Fakas, Zhi Cai and Nikos Mamoulis

Abstract—The Object Summary (OS) is a recently proposed tree structure, which summarizes all data held in a relational database about a data subject. An OS can potentially be very large in size and therefore unfriendly for users who wish to view synoptic information about the data subject. In this paper, we investigate the effective and efficient retrieval of concise and informative OS snippets (denoted as size-l OSs). We propose and investigate the effectiveness of two types of size-l OSs, namely size-l OS(t)s and size-l OS(a)s that consist of l tuple nodes and l attribute nodes respectively. For computing size-l OSs, we propose an optimal dynamic programming algorithm, two greedy algorithms and preprocessing heuristics. By collecting feedback from real users (e.g. from DBLP authors), we assess the relative usability of the two different types of snippets, the choice of the size-l parameter, as well as the effectiveness of the snippets with respect to the user expectations. In addition, via thorough evaluation on real databases, we test the speed and effectiveness of our techniques.

Index Terms—Relational Databases, Keyword Search, Ranking, Summaries

1 Introduction

The success of the web keyword search (W-KwS) paradigm has encouraged the application of keyword search in relational databases (R-KwS) [1]–[3]. The R-KwS paradigm is used to find tuples that contain the keywords and their relationships through foreign-key links. For example, query "Faloutsos"+"Agrawal" over the DBLP database returns tuples Faloutsos and Agrawal from the Author table and their associations through co-authored papers. However, the R-KwS paradigm may not be very effective in extracting information about a particular *data subject* (DS) (e.g., "Faloutsos"), because it only retrieves tuples which contain the keywords. On the other hand, users may wish to view the *context* of most important tuples around a central tuple (i.e., a data subject). In view of this, in our previous work [4], we introduced the concept of *object summary* (OS); an OS summarizes all data held in a database about a particular DS, searched by some query keyword(s). More precisely, an OS is a *tree* with the tuple t^{DS} containing the keywords (e.g., Author tuple Christos Faloutsos) as the root node and its neighboring tuples, containing additional information (e.g. his papers, co-authors etc.), as child or descendant nodes. The precise definition of an OS is discussed in Section 2; in a nutshell, a tuple is included in the OS if it is of high affinity (based on link properties in the tuple network graph of the database) and it is connected to t^{DS} via a short path.

- *G.J. Fakas is with the Manchester Metropolitan University E-mail: g.fakas@mmu.ac.uk*
- *Z. Cai is with the Beijing University of Technology E-mail: caiz@bjut.edu.cn*
- *N. Mamoulis is with the University of Hong Kong E-mail: nikos@cs.hku.hk*

Example 1 illustrates the Christos Faloutsos OS in DBLP. Note that the complete OS consists of 1,309 tuples and it may not only be unfriendly to present to users that prefer a quick glance first before deciding whether they are interested in the OS, but also expensive to produce. On the other hand, an OS snippet of size l (denoted as size-l OS), composed of only l representative and important tuples, may be more appropriate. It is also in analogy to web search engines which show small snippets of the query results (web pages), in order for the users to select (click on) their most preferable ones. Example 2 illustrates the size-15 OS for Christos Faloutsos, which includes only a tree of 15 tuples showing the most important (representative) context of the corresponding t^{DS}.

Example 1: The OS for Christos Faloutsos

Author: Christos **Faloutsos**
Paper: On Power-law Relationalships of the Internet Topology.
Conference:SIGCOMM. **Year**:1999.
Co-Author(s):Michalis Faloutsos, Petros Faloutsos.
Cites: Building Shared...
Cited by: The Structure...
Paper:The R+-tree: A Dynamic Index for Multi-Dimensional Objects
Conference:VLDB. **Year**:1987.
Co-Author(s):N. Roussopoulos, T. Sellis.
Cites: Organization and... **Cited by:**Point-set topological...
Paper: Declustering Using Error Correcting Codes.
Conference:PODS. **Year**:1989. **Co-Author(s)**:Dimitris N. Metaxas.
Cites: Optimal... **Cited by:**Declustering spatial...
...
(Total 1,309 tuples)

In this paper, we investigate in detail the definition and generation of size-l OSs. We propose that the size-l OSs should be a stand-alone size-l sub-graph of the complete OS, composed of l important (tuple or attribute-value) nodes only, so that the user can comprehend it without any additional information. A stand-alone size-l OS should preserve meaningful and self-descriptive semantics about the DS. As we explain

in Section 3, for this reason, the l nodes should form a connected graph that includes the root of the OS (i.e. t^{DS}). To distinguish the importance of an individual tuple node t_i to be included in a size-l OS, a *local importance* score is defined for t_i by combining the tuple's *global importance* score in the database with the tuple's *affinity* [4] in the OS respectively. Based on the local importance scores of the nodes of an OS, we can find the partial OS of size l with the maximum aggregate importance, which includes nodes that are connected with the OS root.

Example 2: The size-15 OSs for Christos Faloutsos

Author: Christos **Faloutsos**
Paper:On Power-law Relationalships of the Internet Topology.
Conference:SIGCOMM. **Year**:1999.
Co-Author:M. Faloutsos, **Co-Author**: P. Faloutsos.
Paper:The QBIC Project: Querying Images by Content, Using, Color, Texture and Shape.
Conference:SPIE. **Year**:1993.
Co-Author:C. W. Niblack, **Co-Author**: D. Petkovic...
Paper:The R+-tree: A Dynamic Index for Multi-Dimensional Objects
Conference:VLDB
Co-Author:N. Roussopoulos, **Co-Author**: T. Sellis.

Although a size-l OS is more concise than the corresponding complete OS, the fact that it is a selection at the tuple level enforces the co-existence in the snippet of all attribute-value pairs in the same (selected) tuples, even though the affinity or importance of different attributes may vary. For example, in Figure 1, if a paper is chosen to be included in the size-l OS, essentially all values of the corresponding tuple would be included (e.g., title of paper, number of pages, keywords, abstract, etc.), although only the title may be of sufficient affinity or importance. While in the preliminary version of this paper [5], we studied size-l OS only at the tuple level, based on this observation, in this paper, we also define and study a version of size-l OS which includes the l representative and important *attribute-value pairs* (as opposed to tuples) in the OS. We denote the snippets at the tuple and attribute-value level as size-l OS(t) and size-l OS(a), respectively. Note that a size-l OS(a) is more flexible in selecting tuples and information from them (i.e., partial information from the selected tuples may be included in the snippet). For example, a size-l OS(a) for Christos Faloutsos would prefer selecting more paper titles over less but complete paper tuples.

The efficient computation of size-l OSs is a challenging problem; a brute force approach that considers all candidate size-l OSs before finding the one with the maximum importance can be very expensive. Thus, we first propose an optimal algorithm based on dynamic programming, which is efficient for small problems; however, it does not scale well with the OS size and l. Then, in view of this, we design two practical greedy algorithms and two preprocessing algorithms. We provide an extensive experimental study on the DBLP and TPC-H databases, which includes evaluation of the effectiveness of the size-l OSs concepts and comparative evaluation of the efficiency and approximation quality of the proposed algorithms. In order to verify the effectiveness of our framework, we collected user feedback, e.g. by asking DBLP authors (i.e. the DSs themselves) to assess the computed size-l OSs of themselves. The users suggested that the results produced by our method are very close to their expectations. We investigated in detail and verified the efficiency and approximation quality of our algorithms.

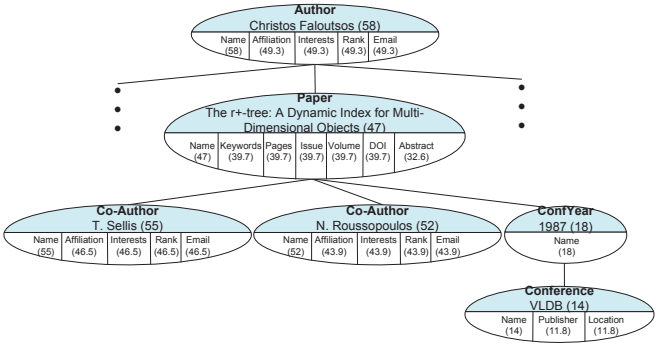


Fig. 1. A Faloutsos OS with all Attributes (DBLP)

Besides modeling answers to queries for data subjects in relational databases, OSs and size-l OSs have many other applications. For example, OSs can automate responses to data protection act (DPA) subject access requests (e.g. the US Privacy Act of 1974, UK DPA of 1984 and 1998 [6] etc.). According to DPA access requests, DSs have the right to request access from any organization to personal information about them. Thus, data controllers of organizations must extract data for a given DS from their databases and present it in an structured form (data controllers should ensure that the disclosed data excludes or anonymizes information about other third parties [7]). Size-l OSs can also be very useful in such applications, as they enhance the usability of OSs. In general, a size-l OS is a concise summary of the *context* around any pivot database tuple, finding application in (interactive) data exploration, schema extraction, etc.

A preliminary version of this paper covering the semantics and algorithms for size-l OS(t)s' appears in [5]. Here, we also propose and investigate size-l OS(a)s and their semantics and algorithms. As we explain and demonstrate in Sections 3 and 8 respectively, size-l OS(a)s are significantly better in terms of usability than size-l OS(t)s in databases with relations having many attributes. In addition, this paper introduces an optimized version of the DP algorithm [5] that reduces time complexity from $O(n^l)$ to $O(nl^2)$. Moreover, this paper introduces a methodology for the automatic selection of l (Section 7). Finally, we provide a more comprehensive evaluation for the usability of size-l OS(t)s and size-l OS(a)s and efficiency of the algorithms.

2 Background and Related Work

In this section, we first describe the concept of *object summaries* (OSs), which we build upon in this paper. We then present and compare with related work. To the best of our knowledge no previous work has focused on the computation of size-l OSs.

2.1 Object Summaries

According to the keyword search paradigm of [4], [8] an object summary (OS) is generated for each tuple (t^{DS}) found in the database that contains the query keyword(s) as part of an attribute's value (e.g. tuple "Christos Faloutsos" of relation Author in the DBLP database). An OS is a tree structure composed of tuples, having t^{DS} as root and t^{DS}'s neighboring tuples (i.e. those associated through foreign keys) as its children/descendants. To construct the OSs, the relations which hold information about the queried Data Subjects (DSs), denoted as R^{DS} (e.g. the Author relation), and the relations linked around R^{DS}s are used. For each R^{DS}, a Data Subject Schema Graph (G^{DS}) is generated. Figure 3 illustrates the G^{DS} for the Author relation of the DBLP[1] database, whose schema is shown in Figure 2 (for a G^{DS} on TPC-H[2], see [5]). A G^{DS} is a directed labeled tree with a fixed maximum depth that has an R^{DS} as a root node and captures the subset of the schema surrounding R^{DS}; any surrounding relations participating in loop or many to many relationships are replicated accordingly.

In order to generate an OS, the relations from G^{DS} which have high affinity with R^{DS} are accessed and joined. The affinity of a relation R_i to R^{DS} can be calculated using the following formula:

$$Af(R_i) = \sum_j w_j m_j \cdot Af(R_{Parent}), \qquad (1)$$

where j ranges over a set of metrics $(m_1, m_2, \ldots, m_n)$ and their corresponding weights $(w_1, w_2, \ldots, w_n)$, and $Af(R_{Parent})$ is the affinity of R_i's *parent* to R^{DS}. The metrics scores range in [0, 1] and the corresponding weights sum to 1; thus, the affinity score of a node is monotonically non-increasing with respect to the node's parent. More precisely we use four metrics: m_1 considers the distance of R_i to R^{DS}, i.e. the shorter the distance the bigger the affinity between the two relations. The remaining metrics consider the connectivity of R_i on both the database schema and data-graph. m_2 measures the relative cardinality, i.e. the average number of tuples of R_i that are connected with each tuple in R_{Parent} whereas m_3 measures their reverse relative cardinality, i.e. the average number of tuples of R_{Parent} that are connected with a tuple in R_i. m_4 considers the schema connectivity of R_i (i.e. the number of relations it is connected to in the relation graph). A small connectivity of R_i suggests bigger affinity with its R_{Parent} and R^{DS}. All relations with affinity higher than a threshold θ are used in the OS generation. For instance, OS of Example 1 is generated using DS "Christos Faloutsos", the Author G^{DS} of Figure 3, and θ=0.7.

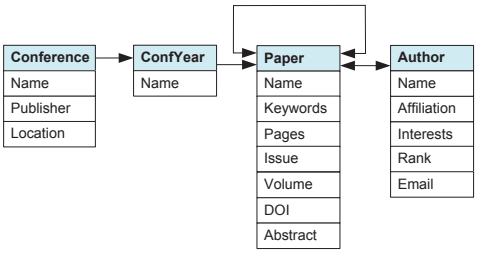


Fig. 2. The DBLP Database Schema

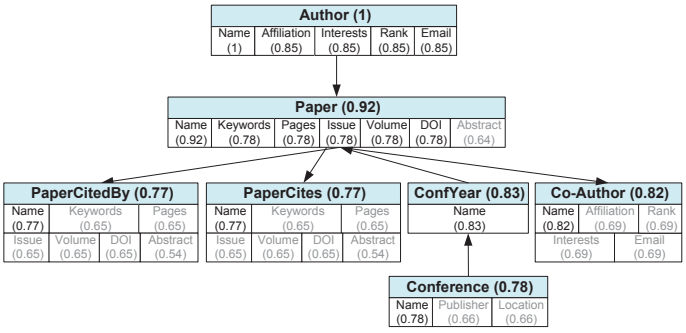


Fig. 3. The DBLP Author $G^{\text{DS}}(a)$ annotated with Affinity and Attributes Affinity (attributes with low transparency have affinity below θ'=0.7)

Instead of including complete tuples in the OS, we can select only a subset of attributes per considered tuple. For this purpose, the G^{DS} can be refined to a $G^{\text{DS}}(a)$, where attributes from each relation are selected based on an *attribute affinity* (i.e. $AAf(R_i.A_j)$) and a threshold (i.e. θ'). Figure 3 illustrates the DBLP Author $G^{\text{DS}}(a)(0.7)$, while Figure 4 illustrates the corresponding fraction of the Faloutsos OS with attributes pruning (i.e. θ=θ'=0.7). In order to automate the process of assigning a score for each attribute (and thus minimize input from DBAs or users), attributes are clustered into disjoint sets (see [4] for more details). More precisely, we define the following set of clusters for attributes: (1) the A_N cluster comprising of naming attributes, such as name, surname, first name, company name etc.; (2) the A_G cluster comprising of general attributes, such as address, date of birth etc.; and finally (3) the A_C cluster comprising of attributes including comments, descriptions, multimedia data etc. For example, considering the DBLP Paper relation, the A_N cluster includes the Name attribute, the A_G includes the Keywords, Pages, Issue, Volume, DOI attributes and A_C the Abstract attribute. (As future work, we plan to investigate the use of ontologies

1. the DBLP (www.informatik.uni-trier.de/∼ley/db/) version of the Microsoft Academic Search database obtained at http://academic.research.microsoft.com

2. www.tpc.org/tpch/

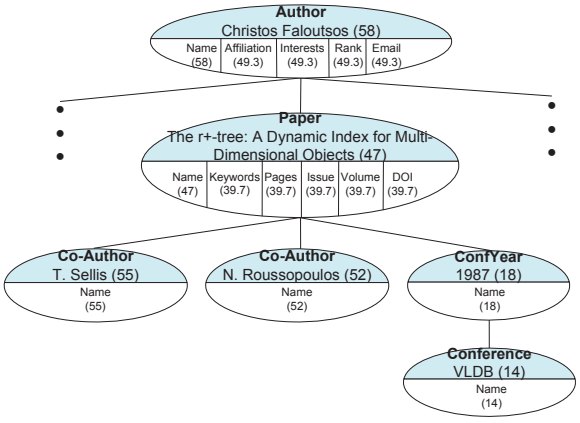


Fig. 4. A Fraction of the Faloutsos OS with filtering of Attributes using $G^{DS}(a)(0.70, 70)$

as to enhance clustering quality and flexibility, e.g. more and finer clusters, and thus facilitate tailored clustering solutions to all kinds of databases.) Then, the affinity of each cluster of attributes $R_i.A_j$ to R^{DS} can be calculated using the following formula:

$$AAf(R_i.A_j) = f(d_{A_j}) \cdot Af(R_i), \quad (2)$$

where $Af(R_i)$ is the affinity of the relation R_i to R^{DS} and $f(d_{A_j})$ is a discounting factor for the importance of the different clusters of attributes to the R^{DS}. For example, after clustering the attributes using this method, clusters A_N, A_G and A_C have discounting factors $f(d_{A_j})$ 1, 0.85 and 0.70 respectively.

2.2 R-KwS and Ranking

R-KwS techniques facilitate the discovery of joining tuples (i.e. Minimal Total Join Networks of Tuples (MTJNTs) [3]) that collectively contain all query keywords and are associated through their keys; for this purpose the concept of *candidate networks* is introduced; see, for example, DISCOVER [3], BANKS [1], [2], [9] etc. The OSs paradigm differs from other R-KwS techniques semantically, since it does not focus on finding and ranking candidate networks that connect the given keywords, but searches for OSs, which are trees centered around the data subject described by the keywords.

Précis Queries [10]–[12] resemble size-l OSs as they append additional information to the result of a R-KwS (i.e., the nodes containing the keywords), by considering neighboring relations that are implicitly related to the keywords. More precisely, a précis query result is a logical subset of the original database (i.e. a subset of relations and a subset of tuples). For instance, the précis of query "Faloutsos", illustrated by Example 3, is a subset of the database that includes the tuples of the three Faloutsos brothers and a subset of their (common) Papers, Co-Authors, Conferences, etc. Précis queries also support the narrative presentation of the results, e.g. "Christos Faloutsos is an Author ...". In contrast, our result is a set of three separate size-l OSs (one OS for each Faloutsos author). In this spirit, some tuples are replicated in OSs when necessary; e.g. the common paper "On Power-low..." appears in all OSs. On the contrary, précis queries do not replicate such information and, compared to OSs, they do not provide a view of the database centered around *each* of the data subjects, which are related to the keywords (see our usability comparison in Section 8.1.3). Précis, similarly to size-l OSs, may have a size constraint. However, the selection of tuples in a précis is random. In contrast, our approach uses algorithms that consider affinity and importance scores when constructing a size-l OS. Summing up, précis are different semantically and technically from OSs.

Example 3: Précis of query "Faloutsos"

Author: Christos **Faloutsos**
Michalis **Faloutsos**
Petros **Faloutsos**

Co-Author(s): N. Roussopoulos, T. Sellis, Carlton W. Niblack, Dragutin Petkovic, Peter Yanker, Demetri Terzopoulos.

...
Paper: On Power-law Relationalships of the Internet Topology.
The QBIC Project: Querying Images by Content, Using...
The r+-tree: A Dynamic Index for Multi-Dimensional Objects.
QoSMIC: Quality of Service Sensitive Multicast Internet Protocol.
The Virtual Stuntman: Dynamic Characters with a Repertoire of...
...
Conference: SIGCOMM. SPIE. VLDB. J. Intell. SIGGRAPH...
...
Year: 1987. 1993. 1994. 1999. 2001.
...

R-KwS techniques also investigate the ranking of their results. Such ranking paradigms consider:

1) IR-Style techniques, which weight the amount of times keywords appear in MTJNs [13], [14]. However, such techniques have limitations when applied to databases in general and also on OSs in particular. As they miss tuples that are related to the keywords, but they do not contain them [15]; e.g. when searching for "Faloutsos", tuples in relation Papers also have importance although they do not include the Faloutsos keyword.

2) Tuples' Importance, which weights the authority flow through relationships, e.g. ObjectRank [15], [16], ValueRank [17], PageRank [18], BANKS (PageRank inspired) [1], [2] and XRANK [19] etc. In this paper we use tuples' importance to model global importance scores and more precisely global ObjectRank (for DBLP) and ValueRank (for TPC-H) (note that our algorithms are orthogonal to how tuple importance is defined and alternative methods are applicable).

2.3 Other Related Work

Document summarization techniques have attracted significant research interest [20], [21]. Web snippets [21] are examples of document summaries that accompany search results of W-KwSs in order to facilitate their quick preview. They can be either static (e.g. composed of the first words of the document or

description metadata) or query-biased (e.g. composed of sentences containing many times the keywords) [20]. Still, the direct application of such techniques on databases in general and OS in particular is ineffective; e.g. they disregard the relational associations and semantics of the displayed tuples. For example, papers authored by Faloutsos (although they do not include the Faloutsos keyword) have importance analogous to their citations and authors; this is ignored by document summarization techniques. Another concept of summarization similar to ours is entity summarization in semantic knowledge graphs. More precisely in [22], given a semantic knowledge graph and an entity represented by a graph node q, a summary of q is a subset of the graph of size l with nodes surrounding node q. Other relevant work is also RELIN [23] that uses a random walk on a graph of features characterizing the entity.

XML keyword search techniques (e.g. [24]), similarly to R-KwSs, facilitate the discovery of XML subtrees that contain all query keywords (e.g. "Faloutsos"+"Agrawal"). Analogously, XML snippets [25] are sub-trees of the complete XML result, with a given size, that contain all keywords. An apparent difference between size-l OSs and XML snippets is their semantics which is analogous to the semantic difference between complete OSs and XML results. Therefore, their generation is a completely different problem. A common aspect of size-l OS and XML snippets is that they are sub-trees of the corresponding complete trees, stemming from the need to preserve self-descriptiveness.

The *information unit* [26] approach proposes analogous search semantics with OSs for the web. Namely, the result of a web keyword search is no longer a single physical document, but a *virtual document* that consists of a set of linked web pages that collectively contain all keywords. SphereSearch [27] investigate keyword search on heterogeneous data from unstructured (Web), semi-structured (XML), and structured data (databases). These techniques adopt and extend the semantics of R-KwS by searching of associations of nodes including the keywords.

3 Size-l OS

A size-l OS keyword query consists of (i) a set of keywords, (ii) a value for l and (iii) the type of the l nodes (either tuples or attributes). The result comprises a set of size-l OSs, one for each data subject which qualifies the keyword query. Next, we provide a uniform approach of addressing the two types of results by proposing OS(t) and OS(a) data structures and then detailed definitions for size-l OS(t)s and size-l OS(a)s and measures for assessing their quality.

3.1 OS(t) and OS(a) Data structures

In order to address in a uniform way size-l OS(t)s and size-l OS(a)s we propose the following structures.

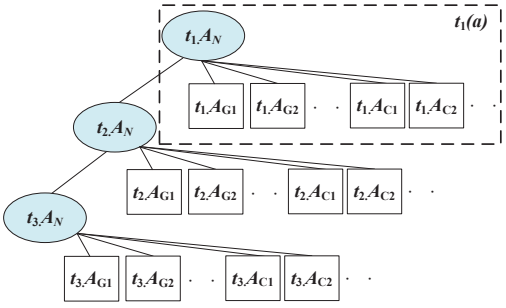


Fig. 5. OS(a) based on A_N, A_G and A_C Clusters

Definition 1 (OS(t)): An OS is a tree rooted at t^{DS} where each node carries a weight and corresponds to an OS tuple.

Definition 2 (OS(a)): Given an OS(t) tree, the corresponding OS(a) is a tree where each node corresponds to an OS attribute, carries the attribute's weight, and is constructed as follows. Each tuple node t_i of the original OS(t) is substituted by a tree $t_i(a)$ of its attributes, rooted at the attribute with the largest local importance (which is usually a naming attribute as it has the largest AAf) and has as children nodes (i) the rest of the attributes of t_i and (ii) the $t_i(a)$s of the corresponding tuple child nodes of t_i.

For this purpose, we apply the approach discussed in Section 2, where attributes are clustered into disjoint sets of attributes (i.e., A_N, A_G and A_C) and are assigned an attribute affinity score (denoted as $AAf(R_i.A_j)$. For instance, consider the example in Figure 5, where the original tuple t_1 is substituted by the tree $t_1(a)$ which consists of the root node $t_1.A_N$ and child nodes $t_1.A_{G1}$, $t_1.A_{G2}$,...,$t_1.A_{C1}$, $t_1.A_{C2}$,... Similarly, the child tuple t_2 is substituted by $t_2(a)$ and the corresponding $t_2.A_N$ becomes a child of $t_1.A_N$. Note, that $t_2.A_N$ has as parent the $t_1.A_N$ and also as children the A_Ns of all its tuples children.

Note that in both cases we have a weighted tree. Hereby, we denote as OS(.) or size-l OS(.) the general case of addressing either type.

3.2 Size-l OS(.)

Definition 3 (Candidate size-l OS(.)): Given an OS(.) and an integer size l, a candidate size-l OS(.) is any subset of the OS composed of l nodes (i.e. tuples or attribute-value pairs), which form a tree rooted at t^{DS} (i.e. the root of the OS tree).

Definition 3 guarantees that the size-l OS remains stand-alone, (so users can understand it as it is without any additional tuples). Consider the example of Figure 4 which is a fraction of the Faloutsos OS; even, if the "R+-tree..." paper has less local importance (e.g. 47) than the co-author(s) Sellis (e.g. 55) and Roussopoulos (e.g. 52), we cannot exclude the paper and include only the co-authors. The rationale is that by excluding the paper tuple we also exclude the semantic association between the author and co-author(s),

which in this case is their common paper. Also note that a size-l OS will not necessarily include the l tuples (or attributes) with the largest importance scores. For example, the co-author Roussopoulos, although with larger importance than the particular paper, may have to be excluded from a size-l OS(t) (e.g. from a size-3 OS(t) which will consist of (i) author "Faloutsos", (ii) paper "The R+-tree..." and (iii) co-author "Sellis"). Similarly, size-l OS(a) should remain stand-alone by including connecting nodes attributes to the root.

Given an OS, we can extract exponentially many size-l OS(.)s that satisfy Definition 3. In the next section we define a measure for the *importance* (i.e., quality) of a candidate size-l OS(.). Our goal then would be to retrieve a size-l OS(.) of the maximum possible quality.

3.3 Importance of a Size-l OS(.)

The (global) importance of any candidate size-l OS(.) S, denoted as $Im(S)$, is defined as:

$$Im(S) = \sum_{n_i \in S} Im(OS, n_i), \quad (3)$$

where $Im(OS, n_i)$ is the *local importance* of node n_i (to be defined below). We say that a candidate size-l OS is an *optimal* size-l OS, if it has the maximum $Im(S)$ (denoted as $max(Im(S))$) over all candidate size-l OSs for the given OS.

Local Importance of Tuples. The local importance of $Im(OS, t_i)$ of each tuple t_i in an OS can be calculated by:

$$Im(OS, t_i) = Im(t_i) \cdot Af(t_i), \quad (4)$$

where $Im(t_i)$ is the global importance of t_i in the database. We use global ObjectRank and ValueRank to calculate global importance, as discussed in Section 2.2. $Af(t_i)$ is the affinity of t_i to the t^{DS}; namely the affinity $Af(R_i)$ of the corresponding relation R_i where t_i belongs, to R^{DS}. This can be calculated from G^{DS} using Equation 1, as discussed in Section 2.1 (alternatively, a domain expert can set $Af(R_i)$s manually). For example, if tuple t_i is paper "The R+-tree..." with $Im(t_i)$ =51 and $Af(t_i) = Af(R_{Paper})$ =0.92 (see the affinity on Author G^{DS} in Figure 3), then $Im(OS, t_i)$ = 51*0.92=47. Multiplying global importance $Im(t_i)$ with affinity $Af(t_i)$ reduces the importance of tuples that are not closely related to the DS. The use of importance and affinity metrics is inspired by other earlier work; e.g. XRANK and précis employ variations of importance and affinity [12], [19].

Local Importance of Attributes. Similarly, the local importance $Im(OS, t_i.a_j)$ of an attribute a_j in a tuple t_i in an OS can be calculated by:

$$Im(OS, t_i.a_j) = Im(t_i) \cdot AAf(t_i.a_j), \quad (5)$$

where $Im(t_i)$ is the global importance of t_i in the database (and can be calculated as for $Im(OS, t_i)$) and $AAf(t_i.a_j)$ is the attribute affinity of $t_i.a_j$ to the t^{DS}, i.e. $AAf(R_i.A_j)$ and can be calculated by Equation 2. As shown in Equation 2, $AAf(t_i.a_j)$ is the result of multiplying the affinity $Af(R_i)$ to R^{DS} of the relation R_i where t_i belongs, with a function of the Attributes Cluster's Affinity Ranking $f(d_{A_j})$. (We used the settings in [4] to calculate $f(d_{A_j})$; alternatively, a domain expert can set $AAf(t_i.a_j)$s manually.)

3.4 Problem Definition

The generation of a size-l OS is a challenging task because we need to select l nodes that are connected to the root of the tree and at the same time result to the $max(Im(S))$. Hence, the problem we study in this paper can be defined as follows:

Problem 1 (Find an optimal size-l OS(.)): Given a t^{DS}, the corresponding G^{DS} and l, find the candidate size-l OS(.) S of maximum $Im(S)$.

A simple approach for solving this problem is to first generate the complete OS, also denoted as OS(.) and then determine the optimal size-l OS(.) from it. A more economical approach is to produce initially a preliminary partial OS, denoted as prelim-l OS(.), instead of a complete OS(.). The rationale of a prelim-l OS(.) is to avoid the extraction and further processing of fruitless tuples and their attributes that are not promising to make it in the size-l OS(.). Choosing to use either approach (complete or preliminary OS(.)), the problem can thus be solved in two phases; namely, (phase 1) generation of a complete or prelim-l OS(.) from the corresponding G^{DS} and (phase 2) determination of a size-l OS(.) from a given initial OS(.) (either complete or prelim-l OS(.)).

We first focus on the second sub-problem (i.e., computing a size-l OS(.) from a given initial OS(.)) and propose a dynamic programming algorithm (Section 4), which finds the optimal solution and two greedy heuristics (Section 5), which find sub-optimal solutions. Then, we brief a pre-processing algorithm that produces a prelim-l OS(.). Recall that the introduction of the OS(a) and OS(t) data structures facilitates the uniform reuse of all these algorithms for either type.

4 Optimal Size-l Determination Algorithm (DP Algorithm)

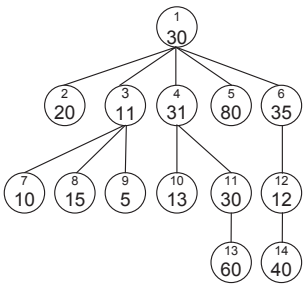
This section describes a dynamic programming (DP) algorithm, which, given an initial OS tree, finds the optimal size-l OS, i.e. an l-sized subtree $S_{root,l}$ of the OS rooted at t^{DS}, such that $Im(S_{root,l})$ is the maximum among all such subtrees. In other words, our objective is to find a subtree of the OS tree, which includes the OS root, it has l nodes, and the sum of its node weights is maximized. For example, consider the OS tree shown in Figure 6, where each node contains an identifier and a weight (i.e., local importance). The optimal size-5 OS is node-set $S_{opt} = \{1, 4, 5, 11, 13\}$ with sum of weights = 231.

Assume that the root of the size-l OS $S_{root,l}$ has a child v and the subtree $S_{v,i}$ rooted at v has i nodes. Then, $S_{v,i}$ should be the optimal size-i subtree rooted at v. DP operates based on exactly this assertion; for each candidate node v to be included in the optimal synopsis and for each number of nodes i in the subtree of v that can be included, we compute the corresponding optimal size-i synopsis and the corresponding sum of weights. The optimal size-i synopsis rooted at v is computed recursively from precomputed size-j synopses ($j < i$) rooted at v's children; to find it, we should consider all synopses formed by v and all size-$(i-1)$ combinations of its children and subtrees rooted at them. For example, to compute the optimal size-5 OS of the OS tree shown in Figure 6, we need to consider all combinations of the root's children and optimal subtrees rooted at them, such that the total number of selected nodes is 4 (e.g., one such combination is nodes $\{2, 4, 10, 6\}$).

Our DP algorithm systematically solves this problem and minimizes the number of required computations. Toward this goal, the algorithm computes two quantities for each node v of the OS:

- $\boldsymbol{S_{v,i}}$. Let $d(v)$ be the depth of node v in OS (the root node has depth 0). The subtree rooted at $d(v)$ can contribute at most $l - d(v)$ nodes to the optimal solution, because in every solution that includes v, the complete path from the root to v must be included (due to the fact that the root node should be included and the solution must be connected). The DP algorithm computes for all $i \in [1, l - d(v)]$, $S_{v,i}$: the optimal size-i subtree rooted at v. For example, let $l = 5$, and consider node $v = 4$ in the tree of Figure 6. Since $d(4) = 1$, DP must compute $S_{4,1}$, $S_{4,2}$, $S_{4,3}$, and $S_{4,4}$. We denote by $Im(S_{v,i})$ the respective weight.
- $\boldsymbol{P_{v,i}}$. In addition, for each node v and for all $i \in [1, l-d(v)]$, DP computes a set $P_{v,i}$ of subtrees which (i) collectively have i nodes, (ii) are all rooted at v or its *left* siblings, and (iii) have the maximum sum of weights among all possible collections of subtrees that satisfy (i) and (ii). For example, in Figure 6, $P(4, 4) = \{2, 4, 11, 13\}$, because among all possible sets of subtrees rooted at node 4 or any of its siblings on the left, the subtrees with the maximum sum of weights are node 2 and the path from node 4 to node 13. We denote as $Im(P_{v,i})$ the respective weight.

The ultimate objective of DP is to compute $S_{root,l}$; this result is computed by a bottom-up, postorder traversal of the nodes in the OS, up to depth $l - 1$. Nodes at depth $l-1$ can only contribute themselves in the optimal solution (nodes at depth at least l cannot participate in a size-l OS). For each encountered node v during the traversal, sets $S_{v,i}$ and $P_{v,i}$ are computed



$\boldsymbol{S_{v,i}}$	$\boldsymbol{P_{v,i}}$
$S_{2,1}$={2}	$P_{2,1}$={2}
$S_{7,1}$={7}	$P_{7,1}$={7}
$S_{8,1}$={8}	$P_{8,1}$={8}, $P_{8,2}$={7,8}
$S_{9,1}$={9}	$P_{9,1}$={8}, $P_{9,2}$={7,8}
$S_{3,1}$={3}, $S_{3,2}$={3,8}, $S_{3,3}$={3,7,8}	$P_{3,1}$={2}, $P_{3,2}$={2,3}, $P_{3,3}$={2,3,8}
$S_{10,1}$={10}	$P_{10,1}$={10}
$S_{13,1}$={13}	$P_{13,1}$={13}
$S_{11,1}$={11}, $S_{11,2}$={11,13}	$P_{11,1}$={11}, $P_{11,2}$={11,13}
$S_{4,1}$={4}, $S_{4,2}$={4,11}, $S_{4,3}$={4,11,13}	$P_{4,1}$={4}, $P_{4,2}$={4,11}, $P_{4,3}$={4,11,13}
$S_{5,1}$={5}	$P_{5,1}$={5}, $P_{5,2}$={4,5}, $P_{5,3}$={4,5,11}
$S_{14,1}$={14}	$P_{14,1}$={14}
$S_{12,1}$={12}, $S_{12,2}$ ={12,14}	$P_{12,1}$={12}, $P_{12,2}$ ={12,14}
$S_{6,1}$={6}, $S_{6,2}$={6,12}, $S_{6,3}$={6,12,40}	$P_{6,1}$={5}, $P_{6,2}$={5,6}, $P_{6,3}$={4,5,6}
$\boldsymbol{S_{1,4}}$**={1,4,5,6}**	–

Fig. 6. The DP Algorithm: Size-4 OS and the Corresponding S and P Sets (in bottom-up postorder)

for all $i \in [1, l - d(v)]$ as follows:

$$S_{v,i} = \begin{cases} v & \text{if } i = 1 \\ v \cup P_{rc,i-1} & \text{otherwise} \end{cases}, \quad (6)$$

where rc is the *rightmost* child of v;

$$P_{v,i} = \begin{cases} S_{v,i} & \text{if } v \text{ has no left sibling} \\ P_{ls,j} \cup S_{v,i-j} & \text{otherwise} \end{cases}, \quad (7)$$

where ls is the *left* sibling of v and $j = \arg\max_{0 \le j \le i} Im(P_{ls,j}) + Im(S_{v,i-j})$.

Note that due to the order of nodes in the traversal (bottom-up, left-to-right), when a node v is accessed all necessary information about its rightmost child and its left sibling have already been computed and can directly be used to compute all $S_{v,i}$'s and $P_{v,i}$'s. Note also that the computation of each $S_{v,i}$ is done at constant time given the precomputed $P_{rc,i-1}$; i.e., the optimal subtree of i nodes rooted at node v includes v and the set of optimal subtrees with at most $i - 1$ nodes rooted at v's children. Finally, $P_{v,i}$ for a node v can be derived from only the $P_{ls,j}$'s stored at the left sibling ls of v, by considering all cases of having j nodes in any of v's left siblings and $i - j$ nodes in a subtree of v, for $j = 0$ to i.

The table in Figure 6 illustrates the $S_{v,i}$ and $P_{v,i}$ sets computed for each node in the OS tree in postorder, for $l = 4$. Note that due to the small size of the complete OS tree some of the $S_{v,i}$'s do not exist

Algorithm 1 The Optimal Size-l OS (DP) Algorithm

DP(l, t^{DS}, G^{DS})
1: **OS Generation**(t^{DS}, G^{DS}) ▷ generates the complete OS, annotates with local importance each node
2: **for each** node v in a post-order traversal of the OS **do**
3: **for** i=1 to $l - d(v)$ **do**
4: use Equation 6 to compute $S_{v,i}$
5: **for** i=1 to $l - d(v)$ **do**
6: use Equation 7 to compute $P_{v,i}$
7: **return** $S_{root,l}$

(e.g., there is no $S_{5,2}$ since node 5 has no children). To illustrate the functionality of DP, assume that the current node is $v = 4$ and the objective is to compute $S_{4,3}$ and $P_{4,3}$. For $S_{4,3}$, we only need $P_{11,2}$, since 11 is the rightmost child of node 4. $P_{11,2}$ has already been computed from our visit to node 11 and $P_{11,2} = \{11, 13\}$. Therefore $S_{4,3} = \{4, 11, 13\}$. To compute $P_{4,3}$, we should consider the best combination of $P_{3,3}$, $P_{3,2} \cup S_{4,1}$, $P_{3,1} \cup S_{4,2}$, and $S_{4,3}$, which is $S_{4,3}$, thus $P_{4,3} = \{4, 11, 13\}$. Note that all $P_{4,i}$'s will be considered in the computation of all $P_{5,i}$'s; all $P_{5,i}$'s will be considered in the computation of all $P_{6,i}$'s, and eventually $P_{6,3}$ will be considered in the computation of the ultimate target: $S_{1,4}$.

Algorithm 1 is a high-level pseudocode of the DP algorithm. In terms of complexity, we need to compute for each node v in the OS up to depth $l - 1$ up to $l - d(v)$ $S_{v,i}$ and $P_{v,i}$ sets; i.e., $O(l)$ sets in total. Each $S_{v,i}$ takes constant time to compute, but each $P_{v,i}$ takes $O(i)$ time. Therefore, the cost of processing each node is $O(l^2)$ and the overall cost of DP is $O(nl^2)$. For large values of l, DP becomes impractical and we resort to the greedy heuristics described in the next section. The space complexity of the algorithm is just $O(l^2)$; for each node v, the sets $S_{v,i}$ are only used to compute the sets $P_{v,i}$ and afterwards they can be dumped; in turn, the sets $P_{v,i}$ are only used by the next node v' in the order (right sibling or parent of v) and afterwards they can also be cleared (recall that for each node only information from the left sibling and rightmost child is required). The following lemma proves the optimality of DP (for a proof, see [5]).

Lemma 1: Algorithm 1 finds the optimal size-l OS.

5 Greedy Size-l Algorithms

The cost of the DP algorithm can be high for large values of l, so in [5], we investigate two greedy heuristics that aim at producing a high-quality size-l OS, not necessarily being the optimal. Similarly to DP, both heuristics can take as input either an OS(t) or a OS(a) tree (which can be either a complete or a prelim-l OSs) and produce a corresponding size-l OS(t) or size-l OS(a), respectively. For the sake of completeness and due to space constraints, we only include short descriptions of these two methods. For detailed descriptions and analytic examples the reader is referred to [5].

5.1 Bottom-Up Pruning Size-l Algorithm

This algorithm, given an initial OS iteratively prunes from the bottom of the tree the $n-l$ leaf nodes with the smallest local importance, where n is the number of nodes in the complete OS. The rationale is that since nodes need to be connected with the root and lower nodes on the tree are expected to have lower importance, we can start pruning from the bottom. Thus, initially, a priority queue (PQ) organizes the current leaf nodes according to their local importance. Then, the algorithm iteratively prunes the leaves with the smallest local importance. Whenever a node becomes a new leaf (due to the pruning of its children), it is added to PQ. The algorithm terminates when only l nodes remain in the tree. The tree is then returned as the size-l OS. In terms of time complexity, the algorithm performs $O(n)$ delete operations in constant time, each potentially followed by an update to the PQ. Since there are $O(n)$ elements in PQ, the cost of each update operation is $O(logn)$. Thus, the overall cost of the algorithm is $O(nlogn)$. This is much lower than the complexity of the DP algorithm, which gives the optimal solution. This method will not always return the optimal solution. In practice, however, it is very accurate (see our experimental results in Section 8.2), due to the property of $Im(OS, t_i)$ and $Im(OS, t_i.a_j)$, which gives higher probability to nodes closer to the root to have a high local importance. Lemma 2 (proved in [5]) gives an optimality condition for this algorithm when OSs are monotonic.

Lemma 2: If the nodes of an OS have monotonically decreasing local importance scores to their distance from the root (i.e. the score of each parent is not smaller than that of its children), then the Bottom-Up Pruning Size-l Algorithm finds the optimal size-l OS.

5.2 Update Top-Path-l Algorithm

The second greedy heuristic initially computes for each node t_i of the OS, the *average importance per node* (denoted as $AI(p_i)$) in the path p_i from the OS root to t_i. This can be done at the same time while generating the OS (or prelim-l OS), which is given as input to the algorithm. The algorithm then selects the path p_i of nodes with the largest *average importance per node* (denoted as $AI(p_i)$), adds p_i to the size-l OS and removes the nodes of p_i from the OS; the remaining nodes in the OS now form a *forest*; each child of a node in p_i is the root of a tree. Due to the removal of p_i, the values of $AI(p_j)$ for each node t_j which is a descendant of any node in p_i are updated. The process of selecting the path with the highest $AI(p_i)$, adding it to the size-l OS, removing it from the OS, and revising any affected $AI(p_j)$'s is repeated on the resulting forest as long as less than l nodes have been

selected so far. If less than $|p_i|$ nodes are needed to complete the size-l OS then only the top nodes of the path are added to the size-l OS (because only these nodes are connected to the current size-l OS). Note that each time a path is selected, it is guaranteed to be connected with the previously selected paths (as the root of the selected path should be a child of a previously selected path), therefore the selected paths will form a valid size-l OS. The complexity of the algorithm can be as high as $O(nl)$, where n is the size of the complete OS, as at each step the algorithm may choose only one node which causes the update of $O(n)$ paths. In terms of approximation quality, this algorithm not always returns the optimal solution; however, empirically, this method gives better results than Bottom-Up Pruning.

6 Prelim-l Preprocessing Algorithms

Instead of operating on the complete OS, which may be expensive to generate and search, any of our algorithms can work on a smaller OS, which hopefully includes a good size-l OS. We denote such a preliminary partial OS as prelim-l OS (with size j where $l \leq j \leq |OS|$). In this case, DP is not expected to return the optimal result, unless the prelim-l OS is guaranteed to include it. Determining a prelim-l OS that includes the optimal size-l OS can be very expensive, therefore we propose a heuristic, which produces a prelim-l OS that includes at least the l nodes of the complete OS with the largest local importance (denoted as top-l set). Using *avoidance conditions*, *statistics* that summarize the range of local importance of every tuple in each relation and indexes we can infer upper bounds for the local importance of tuples and thus safely predict whether a candidate path can potentially produce useful tuples.

The algorithm for generating prelim-l OS(t)s, is described in detail in [5]. Prelim-l OS(a)s can be generated by adapting this algorithm. The difference is that when adding tuples to the prelim-l OS(a) (using the prelim-l OS(t) algorithms), the tuples are decomposed to attribute-value pairs and the local importance of each attribute (rather the local importance of the tuple node) are added to the OS. In addition, we introduce an additional *avoidance condition* that avoids adding leaf attributes (e.g. non Naming attributes) on the prelim-l OS(a) if their local importance is less than the largest l importances found so far.

The Prelim-l OS generation algorithms need up to n I/O accesses in the worst case, where n is the amount of tuples in the complete OS. In terms of approximation, if monotonicity holds on the tuples of the corresponding OS, then the corresponding prelim-l OS will include the optimal size-l OS. Note that $t_i(a)$ trees are always monotonic, since the local importance of the root Naming attribute (i.e. A_N) is always greater than the local importance of the leaf attribute nodes (i.e. A_G and A_C).

TABLE 1
Calculation of the Sc_l's (and **best** l's)

l	$li(t_i.a_j)$	$Im(S)$	$Sc_l(average)$
1	58.0	58.0	58.0
2	49.3	107.3	53.7
3	49.3	156.6	52.2
4	49.3	205.9	51.5
5	49.3	255.2	51.0
6	47.0	302.2	50.4
7	55.0	357.2	51.0
8	52.0	409.2	**51.2**
9	39.9	449.2	49.9
10	39.9	489.1	48.9
11	39.9	529.0	48.1
12	39.9	569.0	47.4
13	39.9	609.0	46.8
14	18.0	627.0	44.8
15	14.0	641.0	42.7

7 Selecting an Effective Value of l

So far, we have assumed that parameter l, which limits the size of an OS summary is given by the user. However, in practice, it could be too hard for a user to provide an appropriate value for l. In this section, we analyze the effect of l on the quality of a size-l OS and propose a mechanism for its automated selection.

Recall that the properties of a good size-l OSs are contradictory; the summary should be small in size and at the same time contain many important nodes. A larger size-l OS is preferable only if it includes significantly many more important nodes compared to a smaller summary of the same OS. We propose a technique that selects an effective l-value for a particular OS, which we denote as *best* l, from a given range L of l-values. The range L for a particular G^{DS} can be obtained with the help of user evaluators (or can be set by a database expert). In this work, we asked human evaluators to suggest appropriate ranges L for all G^{DS}s and set L to be the average of the proposed ranges (see Section 8.1.2 for more details). We denote as $L(t)$ and $L(a)$ the ranges of l for OS(t)s and OS(a)s respectively. For instance, the proposed $L(t)$ and $L(a)$ for DBLP Author are [3..7] and [5..12] and for TPC-H Customer they are [3..7] and [11..17], respectively (see also Table 2).

We propose to calculate a score for each size-l OS, denoted as Sc_l, for all values of l in L. The value of l that results in the maximum Sc_l is considered to be the best l. The best l should consider the local importance of the included nodes and additional nodes should only be included if they will significantly contribute with their large importance. A measure that satisfies these criteria is the average of the local importance of the nodes of a size-l OS. Hence, Sc_l can be calculated by:

$$Sc_l = \frac{Im(S)}{l}, \tag{8}$$

For example, considering the DBLP database,

$L(a)$ =[5..12] and the fraction of the Faloutsos size-l OS(a) (of Figure 4), $l = 8$ has $Sc_8 = 51.2$, which is the maximum among all Sc_ls. Therefore 8 is the recommended value of l for this summary. Table 1 illustrates the scores of all values of l in $L(a)$.

A simple approach of calculating the best l is to determine all size-l OSs for the range L in order to calculate their Sc_l score. For this purpose, we can easily adapt size-l OS determination algorithms and compute incrementally (rather than independently) the summaries for the whole range of l-values in L. Let l_{min} and l_{max} be the minimum and maximum values in L. The DP algorithm can directly be applied using l_{max}; upon reaching the root of the tree, the best combinations for all values of l in L are compared in order to select the one with the maximum score (and the corresponding value of l). For the Bottom-Up Pruning algorithm, we prune nodes from the OS until l_{min} nodes remain in the size-l OS, whereas for the Update Top-Path-l algorithm we append nodes on the size-l OS until we get l_{max} nodes. The class of the time bounds for finding the best l remains the same as determining the size-l OS. The cost of DP is $O(nl^2_{max})$, while the two heuristics require time $O((n-l_{min})logn)$ instead of $O((n-l)logn)$ and $O(l_{max}n)$ instead of $O(ln)$ respectively.

8 Experimental Evaluation

In this section, we experimentally evaluate the proposed size-l OS concept and algorithms. We evaluate both complete and prelim-l OSs on both size-l OS(t)s and size-l OS(a)s. Firstly, we thoroughly investigate the effectiveness and usability of the proposed size-l OSs and the effectiveness of the best l with the help of human evaluators. Then, we investigate the approximation quality of the size-l OSs produced by the greedy heuristics against to that of the corresponding optimal OSs. Finally, we comparatively investigate the efficiency of the proposed algorithms.

We used two databases: DBLP and TPC-H (with scale factor 1). The two databases have 2,959,511 and 8,661,245 tuples, occupying 513MB and 1.1GB on the disk, respectively. We used ObjectRank (global) [15] and ValueRank [17] to calculate the global importance for the tuples of the DBLP and TPC-H databases respectively. We investigate various settings of their parameters that have been studied in [15] and [17]; namely, two G^{A}s: (1) the G^{A1}s (default) (see Appendix of [5]) whereas (2) the G^{A2} for the DBLP has common transfer rates (0.3) for all edges and for the TPC-H neglects values (i.e. becomes an ObjectRank G^{A}) and three values of d: d_1=0.85 (default), d_2=0.10 and d_3=0.99. We use Equations 1 and 2 to calculate affinity. For the experiments, we used Java, MySQL, cold cache and a PC with an AMD Phenom 9650 2.3GHz (Quad-Core) processor and 4GB of memory.

8.1 Effectiveness

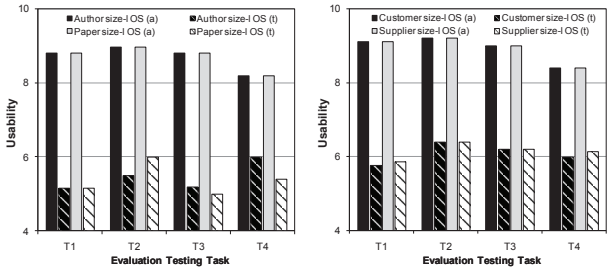
8.1.1 Usability of Size-OS(t)s and Size-l OS(a)s

We used human evaluators to measure effectiveness. First, we familiarized them with the concepts of OSs in general and size-l OSs in particular. Specifically, we explained that a good size-l OS should be a stand-alone and meaningful synopsis of the most important information about the particular DS. Then, we provided them with OSs and asked them to size-l them for l = 5, 10, 15, 20, 25, 30. None of our evaluators were involved in this paper. We measured the effectiveness of our approach as the average percentage of the nodes that exist in both the evaluators' size-l OSs and the computed size-l OS by our methods. This measure corresponds to *recall* and *precision* at the same time, as both the OSs compared have a common size.

Since the DBLP database includes data about real people and their papers, we asked the DSs themselves (i.e. eleven authors listed in DBLP) to suggest their own Author and Paper size-l OS(t)s and size-l OS(a)s. The rationale of this evaluation is that the DSs themselves have the best knowledge of their work and can therefore provide accurate summaries. For TPC-H, we presented 16 random OSs to eight evaluators and asked them to size-l them. The evaluators were professors and researchers from our Universities. In addition, for each OS and tuple, a set of descriptive details and statistics was also provided. For instance for a customer, the total number, size and value of orders and the corresponding minimum, median and maximum values of all customers were provided. The provision of such details gave a better knowledge of the database to the evaluators. The detailed results of this evaluation are omitted due to space constraints. [5] includes all results for size-OS(t)s. In general, the trends are similar between size-OS(t)s and size-l OS(a)s, with accuracy ranging between 60% and 90% and the size-l OS(a)s sometimes being significantly more accurate than size-OS(t)s.

In this paper, we also conducted a comparative study of the usability of size-l OS(t)s and size-l OS(a)s that verifies users' preference for size-l OS(a)s. Usability is the ease of use and learnability of a human-made object; i.e. how efficient it is to use (e.g. whether it takes less time to accomplish a particular task), how easy it is to learn and whether it is more satisfying to use[3]. More precisely, for a given OS, we measured the ease of use of both size-l OS(t)s and size-l OS(a)s through a usability test. We used the same evaluators and OSs as before for the respective databases. We presented to them the two versions of size-l OSs in a randomized order to avoid any bias. We also presented size-l OSs in plain layout and style as to limit evaluators focus only on the content (rather than the presentation; as preliminary evaluation showed that

3. www.wikipedia.org/wiki/Usability



(a) DBLP (b) TPC-H

Fig. 7. Usability of Size-l OS(a)s and Size-l OS(t)s

TABLE 2
Effectiveness of Best l

Database	OSs	Evaluators' $l(L)$ (rounded)	Effectivness of best l
DBLP	Author OS(t)	5 (3-7)	67%
	Paper OS(t)	5 (3-7)	81%
	Author OS(a)	8 (5-12)	66%
	Paper OS(a)	8 (5-12)	75%
TPC-H	Customer OS(t)	14 (11-17)	70%
	Supplier OS(t)	15 (12-18)	88%
	Customer OS(a)	5 (3-7)	92%
	Supplier OS(a)	5 (3-7)	75%

presentation affected significantly evaluators' judgment). Then, we gave them four tasks to complete for each OS and asked them to score (on a scale of 1 to 10) the usability of the two approaches when completing these tasks. Namely, to score them considering which approach takes less time to accomplish each task, is easier to learn and more satisfying to use.

The first task (T1) was to score the general use of the size-l OSs against their objectives (i.e., they should be short, stand-alone and meaningful synopsis of the most important information about the particular DS). The rest of these tasks were to extract information about the DSs that appears in both size-l OSs. For instance, for the Author G^{DS} in DBLP database, task 2 (T2) was to determine a particular close collaborator of the DS that appears in both size-l OSs (e.g. whether Timos Sellis is a collaborator of Christos Faloutsos), task 3 (T3) to extract the affiliation (e.g. the university that Christos Faloutsos works for) task 4 (T4) the research interests of the DS. Similar tasks were used for the remaining G^{DS}s (i.e., Paper in DBLP and Customer, Supplier in TPC-H). We selected the best values of l, as indicated by the human evaluators.

Figure 7 averages evaluators' usability scores of the two methods per G^{DS} and per task. The results show that evaluators favored significantly size-l OS(a)s over size-l OS(t)s for all G^{DS}s and tasks. More precisely, the percent difference of their usability scores ranges from 36% to 70%. This verifies that size-l OS(a)s are more friendly to users because they avoid unnecessary inclusion of all attributes of each selected tuple.

8.1.2 Effectiveness of Best l

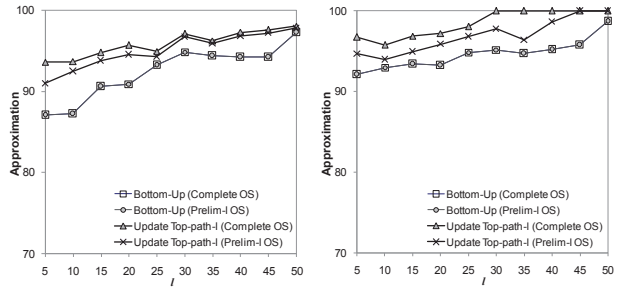
In the next experiment, we compare the best l value (denoted by l_{pred}) which we produced using the methodology of Section 7 against the l our evaluators proposed for each OS (denoted by l_{act}). For the purpose of this evaluation, we used the same evaluators and OSs as before. Firstly, we emphasized to our evaluators the contradicting properties of a good size-l OS, namely that it should be small in size and at the same time contain many important nodes. Then, we asked them to suggest an l_{act} value for the particular OSs. Again, we presented OSs in plain layout and style, as preliminary evaluation revealed that presentation affected significantly evaluators' judgment on proposing l_{act}. During this evaluation task, we also asked evaluators to propose for each OS a range L to select l from and we used that range for selecting the l_{pred} automatically (as discussed in Section 7).

Table 2 shows the rounded average of the l_{act}s (and respective Ls, i.e. Ls' l_{min} and l_{max} rounded averages, where l_{max} and l_{min} are the maximum and minimum l values in L respectively) suggested by our evaluators and the average of the effectiveness of our computed l_{pred} for the given OSs per G^{DS}. For each OS, we measure the effectiveness of l_{pred} as the percent difference between the l_{pred} and evaluators' l_{act} normalized in the range L, i.e. $100 * [1 - \frac{|l_{pred} - l_{act}|}{l_{max} - l_{min}}]$ (the denominator is always ≥ 1 since we assume that $l_{max} > l_{min}$ as otherwise this technique is meaningless). Our approach gives good results with overall average 74.5%.

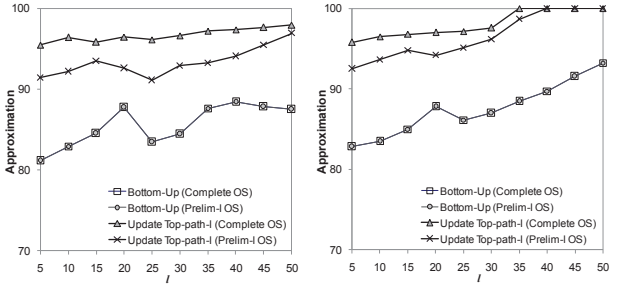
8.1.3 Comparative Evaluation

According to our knowledge there is not any other existing work directly relevant to size-l OSs, so we can compare our results with. Nevertheless, we have attempted a qualitative comparison with the most related previous work.

Précis. Précis queries share some similarity with size-l OSs as they also summarize information. The main difference is that we summarize the context around a single tuple (or attribute-value) which corresponds to a *data subject*; whereas a précis query generates a subset (summary) of the whole database based on the results of a keyword query (which may be a network of multiple tuples; Example 3). To verify this difference in practice, we conducted a comparative evaluation of the usability of précis and size-l OSs. More specifically, we asked our evaluators to compare the usability of size-l OSs and précis when trying to obtain a stand-alone and meaningful synopsis of the most important information about a particular DS (which is the objective of this work). For this purpose, we produced précis results containing the most important nodes using the same scoring values as in size-l OSs (to make a fair comparison). We used a size proportional to l and the amount of OSs, e.g.

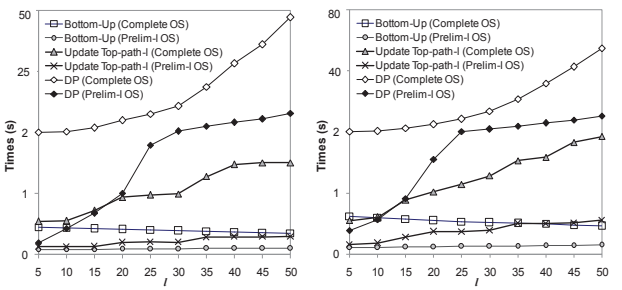


(a) Author OS(t) ($|\overline{OS}|$=1116) (b) Author OS(a) ($|\overline{OS}|$=1584)

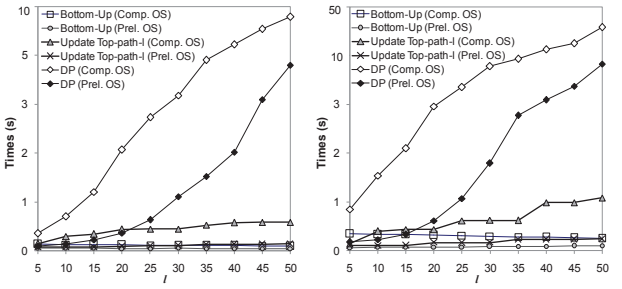


(c) Customer OS(t) ($|\overline{OS}|$=176) (d) Customer OS(a) ($|\overline{OS}|$=742)

Fig. 8. Approximation Quality on DBLP and TPC-H



(a) Author OS(t) ($|\overline{OS}|$=1116) (b) Author OS(a) ($|\overline{OS}|$=1584)



(c) Customer OS(t) ($|\overline{OS}|$=176) (d) Customer OS(a) ($|\overline{OS}|$=742)

Fig. 9. Efficiency on DBLP and TPC-H

l=10 and |OS| =3 (i.e. for instance for 3 Authors) then the size of précis will be 30 tuples. We experimented for various sizes of l and |OS|. As anticipated the evaluation verified users' significant preference for size-l OSs and thus we omit further details. The main justification for the preference of the evaluators was that it was very difficult to determine from précis results the association of additional information with the DSs. For instance, it was hard to determine the author of a particular paper (as they had to do it themselves manually through foreign keys). Even when |OS| =1, it was difficult to determine associations, e.g. the association of a conference with an author.

8.2 Approximation Quality

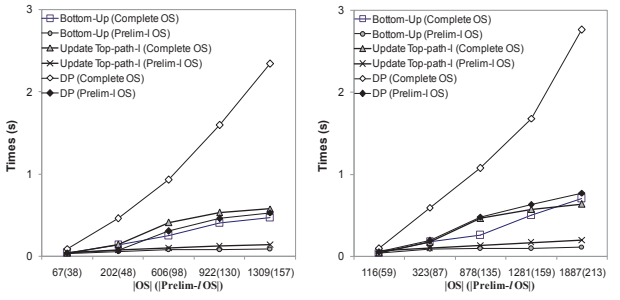
We now compare the importance of the size-l OSs produced by the greedy methods against the optimal ones. More precisely, the results of Figure 8 represent the approximation quality, namely the ratio of the importance of the computed size-l OS against the importance of the optimal size-l OS (computed by running DP on the complete OS). We present the average results for 10 random OSs per G^{DS}. The average size (i.e. the amount of nodes) of OSs is also indicated (denoted as ($|\overline{OS}|$)).

Figure 8 illustrates some characteristic results of our study. The Update Top-Path-l is always better than the Bottom-Up Pruning algorithm on both size-l OS(t)s and size-l OS(a)s. In general, the superiority of Update Top-Path-l over Bottom-Up Pruning is up to 14% on both types of size-l OSs. The evaluation also reveals that top-l prelim-l OSs have very low approximation quality loss. They have no impact on the Bottom-Up algorithm and only up to 4% on the Update Top-Path-l algorithm. The results also reveal that our algorithms achieve better approximation quality on size-l OS(a)s than on size-l OS(t)s and this is because monotonicity is observed more frequently on OS(a)s nodes. Recall that in OS(a)s, A_G and A_C nodes always have less local importance scores than the corresponding parent node A_N, since they always have less attribute affinity and common global importance. Another observation is that the contents of the G^{DS} and the values of the local importance scores also have a significant impact. For instance, for Paper OSs all methods achieved 100% quality. This is because the monotonicity property holds (Lemma 2); the Paper G^{DS} is Paper $\rightarrow$ (Author, PaperCitedBy, PaperCites, Year $\rightarrow$ (Conference)) and the local importance of Conferences is always smaller than those of the corresponding Years.

8.3 Efficiency

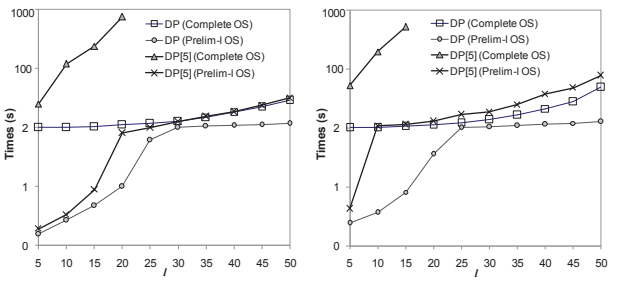
We compare the run-time performance of our algorithms. We used the same OSs as in Section 8.2 (i.e. the same 10 OSs per G^{DS}). Figures 9-12 show the costs of our algorithms for computing size-l OSs, excluding the time required to generate the OS where each algorithm operates on. Note also that the y-axes (time) in some graphs are split to two parts; one linear (bottom) and one exponential (top) in order to show how the expensive DP scales and at the same time keep the differences between the other methods visible.

Figure 9 shows the costs of our algorithms for computing size-l OS(t)s and size-l OS(a)s from OSs of two characteristic G^{DS}s with various sizes and using



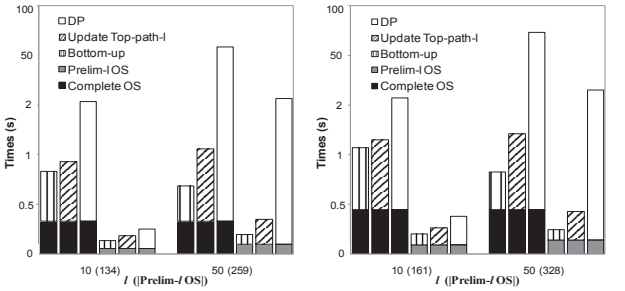
(a) Author (size-10 OS(t)) (b) Author (size-10 OS(a))

Fig. 10. Efficiency (varying OS size)



(a) Author OS(t) ($\overline{|OS|}$=1116) (b) Author OS(a) ($\overline{|OS|}$=1584)

Fig. 11. Efficiency of DP Algorithms



(a) Supplier OS(t) ($\overline{|OS|}$=1341) (b) Supplier OS(a) ($\overline{|OS|}$=1623)

Fig. 12. Efficiency (cost breakdown)

a range of l values. The average sizes of the OSs on which the algorithms operate are indicated in brackets for each G^{DS}. From the average sizes, we can also compare the difference between $\overline{|OS(t)|}$ and $\overline{|OS(a)|}$ (e.g. for the DBLP Author OSs, the average $\overline{|OS(t)|}$ and $\overline{|OS(a)|}$ correspond to 1,116 and 1,584 tuple and attribute nodes respectively). Figures 10(a) and (b) show scalability for Author OSs of different sizes, after fixing l=10 (analogous results were obtained from all G^{DS}s and thus we omit them). Each value on the x-axis represents an OS size (and the corresponding prelim-10 OS size). Comparing these numbers, we can get an indication of prelim-l OSs savings; e.g. the OS(t) with size 1,309 has a prelim-10 OS(t) size 157 (i.e. 11% of the size of the complete OS). Again, we can compare the difference between $|OS(t)|$ and $|OS(a)|$, e.g. the OS(t) with size 1,309 corresponds to an OS(a) with size 1,887.

As expected, both the OS size and l negatively affect the cost. We also observe that since size-l OS(t)s and size-l OS(a)s are computed from initial OSs of different sizes, where $|\text{OS}(t)| < |\text{OS}(a)|$, the cost of computing a size-l OS(a) is higher compared to computing the corresponding size-l OS(t) for the same DS. Bottom-Up Pruning is consistently faster than Update Top-Path-l, as it requires fewer operations. An interesting observation is that Bottom-Up Pruning on the complete OS becomes faster as l increases, because n-l drops and fewer de-heaping operations are needed. The cost of the DP algorithm becomes too high for moderate to large OSs and values of l. For instance, the DP takes in average 70 sec. for the Author size-50 OS(a)s determination (i.e. up to 150 times slower than counterpart heuristics). Evidently, this is still a very slow response for keyword search queries where inpatient users would expect fast responses. Figure 11 compares the new version of the DP algorithm against the version proposed in [5]. Note that the the DP [5] does not scale well and that we had to terminate the algorithm after 30 min. of running.

Figures 12(a) and (b) break down the cost to OS generation (bottom of the bar) and size-l computation (top of the bar) for each method on Supplier OSs (that have the largest sizes among all tested OSs). We investigated two approaches for generating the OSs; the first employs an in-memory data-graph and the second computes the OS directly from the database. The OSs are generated much faster using the data graph; thus, we present only the data-graph based results. For example, to generate a Supplier OS only 0.2 sec. are required using the data-graph, compared to 12.9 sec. directly from the database. The DBLP and TPC-H data-graphs take only 17 sec. and 128 sec. to generate and occupy 150MB and 500MB, respectively. More precisely, our data-graph nodes correspond to the database tuples and edges to tuples relationships (through their primary and foreign keys). Note that the data-graph is only an index and does not contain actual data as nodes capture only keys and global importance. The figures also show the average sizes of the complete OSs (1,341) and the prelim-l OSs (134 and 259 for $l = 10$ and $l = 50$, respectively). The prelim-l OS generation is always faster than that of the complete OS; thus, prelim-l OSs further reduce the time required by our algorithms. Bottom-Up Pruning becomes on average up to 5.7 times faster whereas the Update Top-Path-l is up to 4.1 times. Note that the size of the database does not impact the OS generation time, because hash-maps are used to look-up the required nodes of an OS; we omit experimental results, due to space constraints.

9 Conclusion and Future Work

In this paper, we proposed and studied the effectiveness of a new concept; the size-l OS targets the

synoptic and stand-alone presentation of a large OS. More precisely, we proposed two types of size-l OSs, i.e. size-l OS(t)s and size-l OS(a)s. Then, we investigated the effective and efficient generation of size-l OSs. We proposed a dynamic programming algorithm and two efficient greedy heuristics for producing size-l OSs. We also introduced a technique for the automated selection of an appropriate value for l. Finally, a systematic experimental evaluation conducted on the DBLP and TPC-H databases verifies the effectiveness, approximation quality and efficiency of our techniques. A direction of future work concerns the further exploration of algorithms using hashing and reachability indexing techniques [28]. Another challenging problem is the combined size-l and top-k ranking of OSs.

ACKNOWLEDGMENT

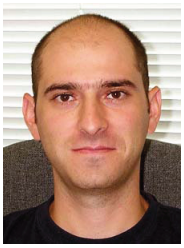
The work of Zhi Cai was supported by grants X0007-011201201 and X4011012201301 from the Beijing University of Technology. We would like to thank Yu Tang for his ideas towards optimizing the DP algorithm.

REFERENCES

[1] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, and P. S. Sudarshan, "BANKS: Browsing and keyword searching in relational databases," in *VLDB*, 2002, pp. 1083–1086.
[2] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword searching and browsing in databases using BANKS," in *ICDE*, 2002, pp. 431–440.
[3] V. Hristidis and Y. Papakonstantinou, "Discover: Keyword search in relational databases," in *VLDB*, 2002, pp. 670–681.
[4] G. J. Fakas, "A novel keyword search paradigm in relational databases: Object summaries," *Data Knowl. Eng.*, vol. 70, no. 2, pp. 208–229, 2011.
[5] G. J. Fakas, Z. Cai, and N. Mamoulis, "Size-l object summaries for relational keyword search," *PVLDB*, vol. 5, no. 3, pp. 229–240, 2011.
[6] 1988, http://en.wikipedia.org/wiki/Data_Protection_Act_1998.
[7] G. J. Fakas, B. Cawley, and Z. Cai, "Automated generation of personal data reports from relational databases," *JIKM*, vol. 10, no. 2, pp. 193–208, 2011.
[8] G. J. Fakas, "Automated generation of object summaries from relational databases: A novel keyword searching paradigm," in *ICDE Workshops*, 2008, pp. 564–567.
[9] A. Markowetz, Y. Yang, and D. Papadias, "Keyword search over relational tables and streams," *ACM Trans. Database Syst.*, vol. 34, no. 3, pp. 17:1–17:51, 2009.
[10] A. Simitsis, G. Koutrika, and Y. Ioannidis, "Generalized précis queries for logical database subset creation." in *ICDE*, 2007, pp. 1382–1386.
[11] ——, "Précis: From unstructured keywords as queries to structured databases as answers," *The VLDB Journal*, vol. 17, no. 1, pp. 117–149, 2008.
[12] G. Koutrika, A. Simitsis, and Y. Ioannidis, "Précis: The essence of a query answer," in *ICDE*, 2006, pp. 69–79.
[13] V. Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient ir-style keyword search over relational databases," in *VLDB*, 2003, pp. 850–861.
[14] Y. Luo, X. Lin, W. Wang, and X. Zhou, "SPARK: Top-k keyword query in relational databases," in *SIGMOD*, 2007, pp. 115–126.
[15] A. Balmin, V. Hristidis, and Y. Papakonstantinou, "Objectrank: Authority-based keyword search in databases," in *VLDB*, 2004, pp. 564–575.
[16] R. Varadarajan, V. Hristidis, and L. Raschid, "Explaining and reformulating authority flow queries," in *ICDE*, 2008, pp. 883–892.
[17] G. J. Fakas and Z. Cai, "Ranking of object summaries," in *DBRank'09, ICDE*, 2009, pp. 1580–1583.
[18] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," in *WWW Conference*, 1998, pp. 107–117.
[19] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram, "XRANK: Ranked keyword search over XML documents," in *SIGMOD*, 2003, pp. 16–27.
[20] A. Tombros and M. Sanderson, "Advantages of query biased summaries in information retrieval," in *SIGIR*, 1998, pp. 2–10.
[21] A. Turpin, Y. Tsegay, D. Hawking, and H. E. Williams, "Fast generation of result snippets in web search," in *SIGIR*, 2007, pp. 127–134.
[22] M. Sydow, M. Pikula, and R. Schenkel, "The notion of diversity in graphical entity summarisation on semantic knowledge graphs," *Journal of Intelligent Information Systems*, 2013.
[23] G. Cheng, T. Tran, and Y. Qu, "Relin: relatedness and informativeness-based centrality for entity summarization," in *The Semantic Web-ISWC 2011*, 2011, pp. 114–129.
[24] V. Hristidis, Y. Papakonstantinou, and A. Balmin, "Keyword proximity search on xml graphs," in *ICDE*, 2003, pp. 367–378.
[25] Y. Huang, Z. Liu, and Y. Chen, "Query biased snippet generation in XML search," in *SIGMOD*, 2008, pp. 315–326.
[26] W.-S. Li, K. S. Candan, Q. Vu, and D. Agrawal, "Retrieving and organizing web pages by "information unit"," in *WWW*, 2001, pp. 230–244.
[27] J. Graupmann, R.Schenkel, and G. Weikum, "The spheresearch engine for unified ranked retrieval of heterogeneous xml and web documents," in *VLDB*, 2005, pp. 529–540.
[28] A. Markowetz, Y. Yang, and D. Papadias, "Reachability indexes for relational keyword search," in *ICDE*, 2009, pp. 1163–1166.

Georgios J. Fakas is a Senior Lecturer in the School of Computing, Mathematics and Digital Technology of the Manchester Metropolitan University, UK. He also worked as a Research Associate at the Swiss Federal Institute of Technology - Lausanne (EPFL), Switzerland and at the University of Cyprus, Cyprus. He obtained his Ph.D. in Computation in 1998 from the Department of Computation, UMIST, Manchester, UK. His research interests include keyword search and ranking.

Zhi Cai is a Lecturer in the College of Computer Science of the Beijing University of Technology, China. He obtained his M.Sc. in 2007 from the Computer Science School of the University of Manchester and his Ph.D. in 2011 from the Department of Computing and Mathematics of the Manchester Metropolitan University, UK. His research interests include ranking in relational databases, keyword search and ontology engineering.



Nikos Mamoulis is a professor at the Department of Computer Science, University of Hong Kong, which he joined in 2001. His research focuses on the management and mining of complex data types, including spatial, spatio-temporal, object-relational, multimedia, text and semi-structured data. He has served on the program committees of over 90 international conferences and workshops on data management and data mining. He was the general chair of SSDBM 2008 and the PC chair of SSTD 2009. He is an associate editor for IEEE TKDE and the VLDB Journal.