Diverse and Proportional Size-l Object Summaries for Keyword Search

Georgios J. Fakas  
Department of Computer Science and Engineering  
HKUST  
Hong Kong  
gfakas@cse.ust.hk

Zhi Cai  
College of Computer Science  
Beijing University of Technology  
Beijing, China  
caiz@bjut.edu.cn

Nikos Mamoulis  
Department of Computer Science  
The University of Hong Kong  
Hong Kong  
nikos@cs.hku.hk

ABSTRACT

The abundance and ubiquity of graphs (e.g., Online Social Networks such as Facebook; bibliographic graphs such as DBLP) necessitates the effective and efficient search over them. Given a set of keywords that can identify a Data Subject (DS), a recently proposed relational keyword search paradigm produces, as a query result, a set of Object Summaries (OSs). An OS is a tree structure rooted at the DS node (i.e., a tuple containing the keywords) with surrounding nodes that summarize all data held on the graph about the DS. OS snippets, denoted as size-l OSs, have also been investigated. Size-l OSs are partial OSs containing l nodes such that the summation of their importance scores results in the maximum possible total score. However, the set of nodes that maximize the total importance score may result in an uninformative size-l OSs, as very important nodes may be repeated in it, dominating other representative information. In view of this limitation, in this paper we investigate the effective and efficient generation of two novel types of OS snippets, i.e. diverse and proportional size-l OSs, denoted as DSize-l and PSize-l OSs. Namely, apart from the importance of each node, we also consider its frequency in the OS and its repetitions in the snippets. We conduct an extensive evaluation on two real graphs (DBLP and Google OSs, denoted as l). We verify the efficiency of our algorithms and evaluate the quality of the OSs, as very important nodes may be repeated in it, dominating other representative information. In view of this limitation, in this paper we investigate the effective and efficient generation of two novel types of OS snippets, i.e. diverse and proportional size-l OSs, denoted as DSize-l and PSize-l OSs. Namely, apart from the importance of each node, we also consider its frequency in the OS and its repetitions in the snippets. We conduct an extensive evaluation on two real graphs (DBLP and Google OSs, denoted as l). We verify the efficiency of our algorithms and evaluate the quality of the OSs, as very important nodes may be repeated in it, dominating other representative information.

1. INTRODUCTION

Keyword search on the web (W-KwS) has dominated our lives, as it facilitates users to find easily and effectively information using only keywords. For instance, the result for query Q1 = "Faloutsos" consists of a set of links to web pages containing the keyword(s) together with their respective snippets. Snippets are short fragments of text extracted from the search results (e.g., web pages); they significantly enhance the usability of search results as they provide an intuition about which results are worth accessing and which can be ignored. Furthermore, snippets may provide the complete answer to the user’s actual information needs (if, for example, the user is only interested in whether Michalis Faloutsos is a Professor), thus preventing the need to retrieve the actual result [27].

The keyword search paradigm has also been introduced in relational databases (R-KwS) (e.g., [19]). According to the R-KwS paradigm, we search for networks of tuples connected via foreign key links that collectively contain the keywords. For example, the 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 when searching information about a particular data subject (DS) (e.g., Faloutsos and his papers, co-authors, etc.), because it only returns tuples containing the keywords (in this case only Faloutsos Author tuples). Hence, R-KwS fails to address search for the context of most important tuples around a central tuple (i.e., DS).

In view of this, in [12], the concept of object summary (OS) was introduced; an OS summarizes all data held in a database about a particular DS, searched by some keyword(s). More precisely, an OS is a tree with the tuple id containing the keywords (e.g., Author tuple M. 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 importance (based on link properties in the tuple network graph of the database) and it is connected to the DS via a short path. For instance, the result for Q2 is a set of OSs: one for each Faloutsos brother. Example 1 illustrates the OS for Michalis Faloutsos (the complete set of papers and the OSs of the other two brothers are omitted due to lack of space). Note that the OS paradigm is in more analogy to the W-KwS paradigm, compared to R-KwS paradigm. For instance, Example 1 resembles a web page (as both include comprehensive information about the DS). Therefore, for the non-technical users with experience only on W-KwS, the OS paradigm will be friendlier and also closer to their expectations.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGMOD’15, May 31–June 4, 2015, Melbourne, Victoria, Australia. 
Copyright 2015 ACM 978-1-4503-2758-9/15/05...

Copyright © 2015 ACM 978-1-4503-2758-9/15/05...

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGMOD’15, May 31–June 4, 2015, Melbourne, Victoria, Australia. 
Copyright 2015 ACM 978-1-4503-2758-9/15/05...

Copyright © 2015 ACM 978-1-4503-2758-9/15/05...
In general, an OS is a concise summary of the context around any pivot database tuple or graph node, finding application in (interactive) data exploration, schema extraction, etc. (e.g., [16]). Another application of this summarization concept is on semantic knowledge graphs [25, 6].

In [14, 15], OS snippets are proposed (denoted as size-\(l\) OSs). size-\(l\) OSs are composed of only the \(l\) most important nodes. Example 2 illustrates the size-\(l\) OS for M. Faloutsos with \(l = 15\) on the DBLP database. According to [14, 15], a size-\(l\) OS should be a standalone sub-graph of the complete OS, composed of the \(l\) most important nodes only, so that the user can comprehend it without any additional information. For this reason, the \(l\) nodes should form a connected graph that includes the root of the OS (i.e., \(n_{\text{DS}}\)). To distinguish the importance of an individual node \(n_{i}\) to be included in a size-\(l\) OS, a local importance score (denoted as \(l_i(n_i)\)) is defined. Based on the local importance scores of the nodes of an OS, we can find the partial OS of size \(l\) with the maximum importance score.

However, this selection criterion (i.e., maximizing importance score) can render such snippets ineffective. For instance, in Example 2, the co-authorship of Michalis with Christos Faloutsos, who is a very important author monopolizes the snippet with papers co-authored only with Christos. Thus, we argue that the diversity of constituent nodes will improve the snippet’s effectiveness. In addition, we argue that frequent appearances of nodes in an OS should also be proportionally represented in an effective snippet. Hence, in this paper we propose two novel snippets, namely diverse and proportional size-\(l\) OSs denoted as DSsize-\(l\) OS and PSsize-\(l\) OS respectively. More precisely, in a DSsize-\(l\) OS we favor diversity by penalizing repetitions of nodes, whereas, in a PSsize-\(l\) OS, we favor proportionality, i.e., a frequent node should be analogously represented, facilitating diversity at the same time. For instance, the DSsize-\(l\) OS of Example 3 includes C. Faloutsos only twice, allowing the appearance of other important co-authors as well. Similarly, the PSsize-\(l\) OS of Example 4 includes also frequent co-authors S. Krishnamurthy and L. Cui who do not appear at all in the DSsize-\(l\) OS. To compute them, we calculate a combined score per node, which integrates (1) importance, (2) affinity to the data subject node \(n_{\text{DS}}\) and (3) diversity or proportionality. The efficient generation of DSsize-\(l\) or PSsize-\(l\) OSs is a challenging problem since information about the repetitions and frequencies of nodes is required and incremental computation is not possible (as opposed to the original size-\(l\) OS computation problem [14]). We propose a brute force algorithm, BF-\(l\), that produces optimal solutions but scales badly. Then, we propose a greedy algorithm (LASP) and its optimization (2-LASP). Both algorithms are general and can address both DSsize-\(l\) and PSsize-\(l\) OS snippet types (with minor modifications). In addition, we propose two preprocessing techniques for the two snippet types (PPrelim-\(l\) and DPrelim-\(l\)) that prune the input OSs before processing them.

We conducted an extensive experimental study on the DBLP bibliographic and Google+ social network datasets. We verify effectiveness by collecting user feedback, e.g., by asking DBLP authors (i.e., the DSs themselves) to evaluate our size-\(l\) OSs. The users suggested that the results produced by our method are very close to their expectations and they are more usable than the respective size-\(l\) OSs, which disregard diversity. In addition, we investigated in detail and verified the efficiency and approximation quality of our algorithms.

The contributions of this paper can be summarized as follows: (1) the introduction of two novel OS snippets, DSsize-\(l\) and PSsize-\(l\) OS, which capture diversity and proportionality respectively; (2) the introduction of a brute force algorithm and efficient greedy algorithms for their generation; (3) an extensive experimental evaluation that verifies the proposed methodologies.

The rest of the paper is structured as follows. Section 2 describes background and related work. Section 3 describes the semantics of DSsize-\(l\) and PSsize-\(l\) OSs. Sections 4 and 5 introduce the optimal and greedy algorithms respectively, whereas Section 6 introduces preprocessing algorithms for DSsize-\(l\) and PSsize-\(l\) OS computation. Section 7 presents experimental results. Finally, Section 8 provides concluding remarks.

2. BACKGROUND AND RELATED WORK

In this section, we first describe the concepts object summary (OS) and size-\(l\) OS, which we build upon in this paper. We then present related work in R-KwS, ranking, and summarization. To the best of our knowledge, no previous work has focused on the computation of diverse and proportional size-\(l\) OSs.

2.1 Object Summaries

According to the keyword search paradigm of [11, 12], an object summary (OS) is generated for each node (tuple) \(n_{\text{DS}}\) found in a graph (database) that contains the query keyword(s) (e.g., ‘Michalis Faloutsos’ node of Author relation in the DBLP database\(^1\)). An OS is a tree structure having \(n_{\text{DS}}\) as a root and its neighboring nodes (i.e., those associated through foreign keys) as its children/ancestor nodes. To construct the OSs, the relations which may hold the queried Data Subjects (DSs), denoted as \(R_{\text{DS}}\) (e.g., the Author relation), and the relations linked around \(R_{\text{DS}}\) are used. For each \(R_{\text{DS}}\), a Data Subject Schema Graph \(G_{\text{DS}}\) is generated. Figure 2 illustrates the \(G_{\text{DS}}\) for the Author relation of the DBLP database, whose schema is shown in Figure 1. A \(G_{\text{DS}}\) is a directed labeled tree with

\(^1\)www.informatik.uni-trier.de/~ley/db/
a fixed maximum depth that has an $R^{DS}$ as a root node and captures the subset of the schema surrounding $R^{OS}$; any surrounding relations participating in loop or many to many relationships are replicated accordingly. Namely, a $G^{DS}$ is a "treeification" of the schema graph, where $R^{DS}$ becomes the root, $R^{DS}$’s neighboring relations become child nodes and so on. Any looped and many to many relationships are replicated together with statistical information that we describe later together with our methods. In order to generate an OS, the relations from $G^{DS}$, which have high affinity with $R^{OS}$ are accessed and joined. The affinity of a relation $R_i$ to $R^{DS}$ can be calculated by the formula:

$$af(R_i) = \sum_j w_j m_j \cdot af(R_{Parent}),$$

(1)

where $j$ ranges over a set of metrics ($m_1, m_2, \ldots, m_s$) and their corresponding weights ($w_1, w_2, \ldots, w_s$), 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). Given an affinity threshold $\theta$, a subset of $G^{DS}$ can be produced. Given a node (tuple) $n^{DS}$ in $R^{DS}$, by traversing the $G^{DS}$ (e.g., by joining the corresponding relations) we can generate the OSs (Algorithm 4). For instance, for $Q$="Faloutsos", and for $n^{DS}="Michalis Faloutsos"$ in the Author $R^{DS}$ of the DBLP database, the OS presented in Example 1 will be generated.

Every tuple $v_i$ in the database carries a global importance weight $gi(v_i)$, calculated using PageRank-inspired measures such as ObjectRank [3, 28] and ValueRank [13]. Due to the “treeification” of the schema graph by $G^{DS}$, multiple tuples in an OS may correspond to the same tuple in the database. For instance, the same co-author (e.g., Christos Faloutsos) may appear multiple times (e.g., 12) in the OS of Michalis. Formally, for a node $n$ of an OS, we use function $g(n)$ to denote the corresponding tuple $v$ in the database. Thus, for two OS nodes $n_i$ and $n_j$, we may have $g(n_j) = g(n_j) = v$. We also denote as $fr(v)$ (i.e., $fr(g(n_i))$, or simply $fr(n_i)$) the frequency of $v$ in the given OS.

2.2 Size-l OSs

According to [14], given an OS and an integer $l$, a candidate size-$l$ OS is any subset of the OS composed of $l$ nodes, such that all $l$ nodes are connected with $n^{DS}$. The authors in [14] argue that a good size-$l$ OS should be a standalone and meaningful synopsis of the most important and representative information about the particular DS (so that users can understand it as, without any additional nodes). In particular, any intermediate nodes that connect $n^{DS}$ (e.g., M. Faloutsos) with other important nodes (e.g., C. Faloutsos) in the size-l OS guarantee that the size-l remains standalone, since these connecting nodes (e.g., co-authored papers) include the semantics of the associations. For instance, in Example 2, if we exclude paper “On Power-law . . .” but only include the co-authors, we exclude the semantic association between $n^{DS}$ and co-author(s), which in this case is their common paper.

The holistic importance $Im(Sl)$ of any candidate size-$l$ OS $Sl$ is defined as $Im(Sl) = \sum_{n_i \in Sl} li(n_i)$, where $li(n_i) = af(n_i) \cdot gi(n_i)$. Thus, the affinity $af(n_i)$ of a node $n_i$ equals the affinity of the relation where $n_i$ belongs (Eq. 1); the global importance $gi(n_i)$ was defined in Section 2.1. A candidate size-l OS is an optimal size-l OS, if it has the maximum $Im(Sl)$ over all candidate size-l OSs.

The generation of a size-l OS is a challenging task because we need to select $l$ nodes that are connected to $n^{DS}$ and at the same time result in the maximum score. An optimal dynamic programming algorithm (requiring $O(n^l)$ time where $n$ is the amount of nodes in the OS) and greedy algorithms were proposed in [14].

2.3 Keyword Search and Ranking

R-KwS paradigms facilitate the discovery of joining tuples (i.e., Minimal Total Join Networks of Tuples (MTJNTs) [19]) that collectively contain all query keywords and are associated through their keys; hence the concept of candidate networks is introduced (e.g., DISCOVER [19, 18]). R-KwS paradigms differ from OSs semantically, since they search for connections of keywords. Précis Queries [22, 24] resemble size-l OSs as they append additional information to the nodes containing the keywords, by considering neighboring relations. However, a précis query result is a logical subset of the original database (see [12] for a detailed comparison to size-l OSs). Other works in this context also investigate indexing and ranking techniques to facilitate efficiency [23].

Related ranking paradigms consider Importance, which weights the authority flow through relationships (e.g., ObjectRank [3, 28], ValueRank [13], PageRank [4], etc.). In this work we use nodes’ importance to model gi and more precisely global ObjectRank (for DBLP). Our algorithms are orthogonal to how importance of nodes is defined (alternative methods could also be investigated).

Document Summarization. Web snippets [26, 27, 20] 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., the first words of the document or metadata) or query-biased (e.g., containing the query keywords). However, the direct application of such techniques on OSs and databases in general is ineffective as they disregard the relational associations (e.g., for Q1, papers authored by Faloutsos will be disregarded as they do not include the “Faloutsos’ keyword).

### 2.4 Diversity

Diversification of query results has attracted a lot of attention recently as a method for improving the quality of results by balancing similarity (relevance) to a query $q$ and dissimilarity among results. Typically, given a query $q$ and a desired number of results $k$, firstly we get a ranked list of results $S$ in descending order of their similarity $\text{sim}(s_i, q)$ to $q$. Then, the objective of diversification is to find a subset of $S$, $R \subseteq S$ of size $k$, such that the elements in $R$ are similar to $q$ (w.r.t $\text{sim}(q, s_i)$) and at the same time dissimilar to each other (w.r.t. $\text{dis}(s_i, s_j)$).

The definition of $\text{sim}$ and $\text{dis}$ scores is orthogonal to the diversification problem per se and a variety of IR-based or probabilistic approaches have been used to define such functions (e.g., PageRank-based similarity). In most cases, the same function is used to estimate both scores (i.e., $\text{dis}(s_i, s_j) = 1 - \text{sim}(s_i, s_j)$).

One of the earliest and most influential diversification functions is Maximal Marginal Relevance (MMR) [5], which trades off between the novelty (a measure of diversity) and relevance of search results; a parameter is used to control this trade off. A general framework for result diversification appears in [17] with eight axioms. In [17, 29], max-sum and max-min mono-objective functionals and algorithms are proposed. Our proposed diversity definition is inspired by this mono-objective approach, where for each document, a single score trades off the relevance to the query and the dissimilarity from other documents. In [1, 2], an intuitive definition of diversity, called $\text{DisC}$ diversity, is proposed where the computed diverse subset $R$ covers all elements of $S$ in the sense that for element in $S$ there should be a similar element in $R$ and at the same time the elements in $R$ are dissimilar to each other (i.e., diversity). In [21], LogRank is proposed, a principled authority-flow based algorithm that computes a representative summary of the user’s activities by selecting activities that are simultaneously important, diverse and time-dispersed.

**Proportionality.** [30, 8, 7] investigate proportional diversification. More precisely, in [8] an election-based method is proposed to address the problem of diversifying searched results proportionally (our work is inspired by this approach). However, this method disregards the similarity (or importance) of the computed set $R$ to the query $q$ and thus may result in including irrelevant objects into $R$. In [30], this limitation is addressed by considering relevance in the objective function.

**Differences.** Our problem has two significant differences from the existing related works that renders their straightforward application inappropriate. (1) Related work considers diversifying a set of mutually independent results (i.e., $S$ and $R$ sets) whereas we aim at finding an $l$-sized connected subtree of the OS, which is required to include the root $n^{\text{root}}$. (2) Previous work considers interval-scaled pairwise dissimilarity of elements (i.e., each element $s_i \in S$ has a degree of dissimilarity to any other element $s_j \in S$) whereas in our problem the dissimilarity between nodes is binary, i.e., OS nodes are either equal or not (i.e., either $g(n_i) = g(n_j)$ or $g(n_i) \neq g(n_j)$).

### 3. DSIZE-L AND PSIZE-L SNIPPETS

We propose two types of size-l OSs, namely diverse DSsize-l OSs and proportional PSsize-l OSs, which extend the size-l OS definition [14] to capture diversity and proportionality, respectively. Similarly to size-l OSs, both DSsize-l OSs and PSsize-l OSs should be standalone sub-trees of the OS, composed of $l$ important and representative nodes only, so that the user can understand them without any additional information. Thus, the $l$ nodes should form a connected graph that includes the root of the OS (i.e., $n^{\text{root}}$).

We argue that an an effective DSsize-l (PSsize-l) OS should gracefully combine diversity (proportionality) and the local importance scores of constituent nodes. Hence, we propose that for each OS node $n_i$, we estimate a respective score for diversity, proportionality, and local importance, denoted by $d_v(n_i)$, $p_p(n_i)$ and $l_i(n_i)$, respectively. $d_v(n_i)$ can be combined to a single score $d_v(n) = \{p_p(n_i)\}$ for a DSsize-l (PSsize-l) OS, simply denoted by $w_i(n_i)$ when the context (i.e., diversity or proportionality) is clear. Then, the objective is to select the $l$ nodes that (1) include $n^{\text{root}}$ and form a connected subtree of the OS and (2) their sum of their $w$ score is maximized. Local importance (i.e., $l_i(n_i)$) can be calculated as in the original size-l OS problem (i.e., by multiplying affinity with global importance [14]), thus hereby we discuss only diversity and proportionality.

Recall that each OS node corresponds a single graph node (i.e., tuple) and that two different OS nodes may either correspond to the same tuple (e.g., $g(n_i) = g(n_j)$) or to different ones (i.e., $g(n_i) \neq g(n_j)$) and this is the only criterion for their similarity. We do not consider the textual similarity among nodes, because this restricts the generality of our problem. For instance, although we can define similarity between papers (e.g., “On Power-law relationships of the Internet Topology” vs. “Power laws and the AS-Level Internet topology”) meaningfully, it does not make sense to define similarity between authors (e.g., “Chen” vs. “Cheng”). By considering categorical similarity only, we signify the importance of frequent nodes regardless their respective local importance.

### 3.1 Diversity (DSsize-l OSs)

We suggest that the $l$ nodes should be diversified by preventing the domination of very important nodes. For example, in the Michalis Faloutsos OS, the co-authorship with the very important author Christos Faloutsos dominates the snippet and this renders the snippet not representative. A natural criterion objective towards measuring diversity is to maximize the sum of dissimilarities between nodes. Thus, for a given graph node we suggest to estimate diversity as follows:

$$d_v(n_i) = 1 - \sum_{n_j \in S, n_j \neq n_i} \text{sim}(n_i, n_j) \frac{1}{l - 1} = 1 - \frac{z(g(n_i) - 1)}{l - 1},$$

where $\text{sim}(n_i, n_j) = 1$ if $g(n_i) = g(n_j)$ or 0 otherwise; therefore the summation of similar nodes for $n_i$, in a snippet is $z(g(n_i))$, where $z(g(n_i))$ is the amount of times $g(n_i)$ appears in the snippet. Dividing by $l - 1$, we normalize $d_v(n_i)$ in the range [0,1]. We also
denote as $dv[z]$ the diversity for any node considering it appears $z$ times in the snippet. For instance, for all nodes appearing only once, $dv[1] = 1$. As another example, consider $l = 10$ and that C. Faloutsos appears 2 times (i.e., $z = 2$); $dv[2] = 1 - \frac{2}{10} = 0.89$ (Table 1). Note that this score corresponds to the graph node $g(n_i)$; thus, both nodes will have the same $dv$, i.e. 0.89 (an alternative way would be to score the first occurrence as 1 and the second as 0.78, since $14 \times 0.78 = 0.89$). This equation is inspired by (1) max-sum diversification that maximizes the sum of the relevance and dissimilarity of a set and by (2) the use of a mono-objective formulation, which, similarly to our equation, combines relevance and dissimilarity to a single value for each document [17].

Note that, in general, diversification approaches trade off (1) the similarity of results with the given query and (2) the dissimilarity among such results using a similarity measure (e.g., IR techniques). For instance, given a query “Internet Topology”, papers “On Power-law relationships of the Internet Topology” and “Power laws and the AS-Level Internet topology” have some similarity to this query but they also have some similarity among them; both types can be estimated using a common IR metric. This is not the case here, since (1) we do not have a similarity of nodes to the query but a local importance score in relation to $n^{OS}$ and (2) the similarity among nodes is categorical. Thus, local importance and categorical similarity are not meaningfully comparable. Note also that their respective values may not be in the same range (e.g., local importance may range in [0,10] whereas $dv$ always ranges in [0,1]). Hence, unlike most diversification approaches, in the combining function $dw(.)$ (to be defined in Section 3.3) we do not sum up local importance and dissimilarity, but multiply them.

### 3.2 Proportionality (PSize-l OSs)

We observe that in an OS we often find graph nodes (i.e., database tuples) multiple times. For instance, in the Michalis Faloutsos OS (see Table 1), we have 37 instances of S. Krishnamurthy, 12 instances of C. Faloutsos, 18 papers in INFOCOM, etc. We denote the frequency of a graph node $v$ in an OS as $fr(v)$ (or simply by $fr$ when the context is clear). Graph nodes that appear in an OS multiple times could sometimes be comparatively weak in terms of importance, but still given their frequency in the OS, they should be represented analogously in an effective snippet. Thus, we suggest that in a PSize-l snippet, disregarding local importance (i.e., assuming that all nodes have the same $l$), we should include nodes in proportion of their frequency. For example, if a graph node $v$ appears 37 times in the total of 1,259 OS nodes, then $v$ should ideally appear $\lfloor 37/1\rfloor \times 259$ times in the respective PSize-l OS. (Note that this may not practically possible as in-between nodes may also be required, i.e., the co-authored papers in our example.)

Since $l$ is small in size, we need some incremental selection policy to cover proportionally the different frequent sets; namely, at each iteration we can append a different frequent node (facilitating to some extend a round-robin selection). For this purpose, we propose the use of the *proportional quotient* as follows:

$$pq(n_i) = \frac{fr(g(n_i))}{\alpha \cdot z(g(n_i)) + 1}$$

where $z(g(n_i))$ is the times that the node $n_i$ is about to be added in the snippet (i.e., it has already been added $z-1$ times), $fr(g(n_i))$ is the frequency that the node appears in the OS and $\alpha$ is a constant that can tune proportionality. This equation is inspired by the Sainte-Laguè Algebra. We found that it is very effective for our problem (other equations can also be considered, e.g., [7, 30]). For a given $z$ value, i.e., a given $z(g(n_i))$, we denote the respective quotient as $pq(n_i)[z]$. The rationale of this quotient is to favor a frequent node and each time a node is added to the snippet its $pq(n_i)[z]$ score is significantly decayed so that other frequent items will be selected, in turn. This way diversification is also facilitated. For instance, considering $fr = 12$ and $\alpha = 2$ for C. Faloutsos, by adding this node for the first time we get $pq[1] = 12/3 = 4$ and for the second time we get $pq[2] = 12/5 = 2.4$.

#### Nodes Addition Order Criterion.

Note that the order we consider adding a node determines the respective $pq(n_i)[z]$ and thus the aggregate score of a PSize-l OS $PSl$. Thus we choose to add them in the order that maximizes the holistic $Im(PSl)$ score (Equation 5; to be defined in Section 3.3). For instance, consider OS nodes $n_i$ and $n_j$ with $g(n_i) = g(n_j)$; if we add $n_i$ first, we get $pq(n_i)[1]$ that will result to the aggregate score $Im(PSl_1)$; if we add $n_j$ first, we get $pq(n_j)[1]$ and respective aggregate $Im(PSl_2)$ score. Therefore, we choose the order that will result to the largest aggregate score (i.e., $\arg \max (Im(PSl_1), Im(PSl_2))$).

Note that we need this constraint as otherwise we may not get deterministic aggregate score (i.e., the same PSize-l OS may produce different scores depending the order we consider to append nodes).

### 3.3 DSize-l and PSize-l Definitions

Based on the above discussion, for DSize-l OSs, we propose the following combining score per node:

$$dw(n_i) = li(n_i) \cdot dv(n_i)$$

where $li(n_i) = af(n_i) \cdot g(n_i)$ is the local importance of $n_i$ and $dv$ is the diversity factor (Equation 2). Table 1 depicts examples of how these scores can be obtained by constituent scores for $l = 10$. For instance, consider the simplified example where we need to select 5 authors (and thus an intermediary paper), then we will select twice C. Faloutsos (i.e., $0.89 \times 1.8 = 0.89 \times 1.8$) and once S. Madden (1.1 \times 1.6), M. Mitzenmacher (1.1 \times 1.4) and G. Varghese (1 \times 1.4). Note that a third addition of C. Faloutsos cannot complete the total 1.4 score, as the additional score is only 1.12 (i.e., $(3 \times 0.78 \times 0.78 \times 0.78 \times 0.78) + 1.1 = 0.56 \times 0.8 + 1.08)$.

**Definition 1 (DSize-l OS).** Given an OS and $l$, a DSize-l OS is a subset of OS that satisfies the following:

1. The size in nodes of DSize-l OS is $l$ (where $l \leq |OS|$)
2. The $l$ nodes form a connected tree rooted at $n^{OS}$
3. Each node $n_i$ carries a weight $dw(n_i)$ (obtained by Eq. 4)

(4) The aggregated score of a DSize-l OS $DSl$ can be calculated by:

$$Im(DSl) = \sum_{n_i \in DSl} dw(n_i)$$

Let a candidate DSize-l OS be any OS subset satisfying conditions 1-3; then, the optimal DSize-l OS is the candidate snippet that has the maximum $Im(DSl)$ among all such candidates.

**Problem 1 (Find an optimal DSize-l OS).** Given an OS and $l$, find the candidate DSize-l OS of maximum score (according to Definition 1).

Analogously, we define the proportionality score per node (i.e., $pw = li \cdot pq$ instead of Eq. 4), PSize-l OS and the optimal PSize-l OS problem (a formal definition is omitted due to the interest of space). For instance, we observe that our selection policy will favor first the addition of S. Krishnamurthy author and the respective co-authored paper; then, the addition of author C. Faloutsos with a co-authored paper; then, another round with these two authors, etc.

**Notation.** For simplicity, we unify $dw$ and $pw$ into a single notation $w$ and refer to the corresponding diversity or
...the diversity/proportionality score of a node simply by drop n respectively. In the rest of the paper, whenever the context is clear, we denote it below in more detail). Once we have appended OS in a breadth first fashion, recursively (alternative traversals such as BF-OS would consider all candidate size-l OSs, (i.e., they affect each others’ score due to the diversity components of the scoring formulae). In other words, given an OS tree T that can be decomposed to subtrees T1, T2, etc., the optimal diversified size-l OS is not necessarily composed by some optimal diversified size-l’ OSs (l’ < l) of the subtrees T1, T2, etc. (which is the case for size-l OSs [14]). The reason is that the computation of an optimal diversified size-l’ OS in a subtree Ti disregards the diversity of its constituent nodes w.r.t. the snippets chosen from the other subtrees Tj ≠ Ti. If we were to consider all possibilities of graph node frequencies in the snippets of other subtrees, this would require exponential space.

5. LARGEST AVERAGED SCORE PATH (LASP)

The BF-l algorithm can be very expensive for large values of l or |OS|; thus we propose LASP, a greedy algorithm, that can produce a size-l OS of high quality at a much lower cost. In a nutshell, LASP firstly generates the OS. It also calculates for each node n, the w(n) score (i.e., using Equation 4 for z=1) and its corresponding average w(n) score per node (denoted as AP(n)) of the path from n to the root.

Then, the algorithm iteratively selects and adds to the size-l OS the path p of nodes with the largest AP. The rationale behind selecting paths instead of single nodes with the largest score is that, we can include nodes of very large importance while their ancestors have less importance as their score is averaged. LASP can compute both types of size-l OSs, i.e., PSize-l and DSize-l OSs. The difference between the two types is that the proportionality equation also considers the frequency of each node, which is calculated during the OS generation process. Thus, given an OS annotated with w and AP scores, the problem of determining either PSize-l or DSize-l is the same. Algorithm 2 is a pseudocode of the heuristic and Figure 3 illustrates an example.

More specifically, LASP firstly generates the OS; during the OS generation, we also maintain a hash table of graph nodes and their corresponding AP values. The rationale behind this is that the AP value of a node can be computed efficiently by traversing the tree in breadth-first order.

Algorithm 2 The Largest Averaged Path Algorithm

LASP(l, n

Algorithm 1 The BF-l Algorithm

BF-l (OS, l)
1: maxScore = 0
2: OSSize-l = {}, curQ = {}, CSize-l = {}
3: Optimal-l (OS, l, OS.n
4: return OSSize-l

Optimal-l (OS, curLen, sNode, CSize-l)
1: if (curLen == 0) then
2: Score = Calculate(CSize-l) ➔ Eq. 4
3: if (Score > MaxScore) then
4: OSSize-l = CSize-l
5: MaxScore = Score
6: return
7: curNode = sNode
8: while (curNode ≠ null) do
9: AddS-(CSize-l, curNode)
10: enQueue(CurQ, ChildrenOf(curNode))
11: NextBF = deQueue(curQ)
12: Optimal-l (OS, curLen - 1, NextBF, CSize-l)
13: curNode = deQueue(curQ)

4. BRUTE FORCE (BF-l) ALGORITHM

A brute force (BF-l) algorithm for computing the optimal DSize-l (or PSize-l) OS would consider all candidate size-l trees, compute the respective scores, and eventually find the optimal solution. BF-l generates all possible candidate trees by traversing the complete OS tree and Figure 3 illustrates an example.
moving the nodes of $p_1$ from the OS, the tree now becomes a forest; each child of a node in $p_i$ is the root of a tree. Accordingly, the $AP$ of affected nodes is updated (1) to disregard the removed nodes in the path selected at the previous step (lines 9 and 10) (2) and also to consider the revised $z$’s (i.e., $z_{+4}$, lines 12-17). This process (i.e., the selection of the path with the largest $AP$, the addition to the size-$l$ OS, and the required updates) continues iteratively as long as the selected nodes are less than $l$. If less than $|p_i|$ nodes are needed to complete 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.

Take for instance the example in Figure 3. Node $n_{11}$ (i.e., graph node $v_7$) has $AP=48.6$, because its path includes nodes $n_1$, $n_5$, and $n_{11}$ with average $w(n_{11})$ being $30+36+80)/3=48.6$. Assuming that $l=5$, at the first iteration, the algorithm selects and appends to size-$l$ OS the path comprising nodes $n_1$, $n_5$, and $n_{11}$ with the largest $AP$, i.e., 48.6. For the remaining nodes, $AP$ is updated to disregard the removed nodes (Figure 3(b)) and also $w$ and thus the respective $AP$s are updated according to new occurrences of graph nodes. Namely, the new $AP$ for $n_{10}$ is 50.5, because its path now includes only $n_4$ and $n_{10}$ with average $w(n_{10})$ being 50.5. Also, nodes $v_7$ and $v_9$ which correspond to the same graph node as $n_{11}$ which has just been added to the size-$l$ OS (i.e., $g(v_7)=g(n_1)= g(n_{11})=v_7$) also need to be updated with new $w$ and $AP$ scores as to consider the revised $z_{v_7}=2$. In general, if such nodes have descendants, then their descendants should also be updated because both their $AP$s and $w$ are affected. The next path to be selected is that ending at $n_{10}$, which adds two more nodes to the size-$l$ OS (Figure 3(c)). Note that $AP(n_{11})$ for each node $n_i$ corresponds to the path starting from $n_1$ to the root of the corresponding unsellected tree (from the unselected forest). For instance, during the second update, $p_{10}$ comprises $n_4$ and $n_{10}$. Note also that the path’s root (e.g., $n_4$) is always the child of a node (in the OS) which already exists in the current size-$l$ OS, e.g., $n_1$ in this case. Thus, each time we select a path to append to the size-$l$ OS, we always get a valid OS.

The time complexity of the algorithm is $O(nl \log(n))$, 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 twice (firstly for the path size update and secondly for the $z$ updates). Each update costs $\log(n)$ time using the priority queue $W$. In terms of approximation quality, this algorithm empirically produces very good results. Hereby, we show cases where this algorithm will return the optimal DSize-$l$ and PSize-$l$ OS.

**Theorem 1.** If the nodes of an OS have monotonically decreasing $w$ scores with respect to their distance from the root (i.e., when the score of each parent is not smaller than that of its children), then LASP returns the optimal PSize-$l$ OS or DSize-$l$ OS. We denote such an OS as monotonic OS($w$).

**Proof.** The optimal PSize-$l$ or DSize-$l$ should include the $l$ nodes with the largest possible $w$ score. Thus, we need to prove that our algorithm can achieve this goal when monotonicity on the $w$ scores hold. Firstly, we show that since monotonicity holds on $w$, the respective $AP$ scores should also be monotonic. For instance, consider the path $n_1, n_2, \ldots, n_k$ with $w(n_1) > w(n_2) > \ldots > w(n_k)$. Then, $AP(n_1) > AP(n_2) > \ldots > AP(n_k)$ since for $i < j$ we have $AP(n_i) = \frac{w(n_i)+\ldots+w(n_j)}{\log(n_j)}$ and $AP(n_j) = \frac{w(n_j)}{\log(n_j)}$. As a result, $j \cdot (w(n_1) + \ldots + w(n_j)) > \sum_{i=1}^{j} (w(n_1) + \ldots + w(n_j)) = \sum_{i=1}^{j} w(n_i)$ and finally that $w(n_1) + \ldots + w(n_i) > i \cdot w(n_j)$. Thus, we can also easily see that the $w$ score of a node is greater than the $AP$ scores of all its descendants. Also note that Equation 4 is monotonic to the repetitions of the same node (i.e., anti-monotonic to $z$), e.g., $w(n_1)[1] > w(n_1)[2]$. Also, we still maintain the monotonicity of descendant nodes after repetitions, i.e., $w(n_1)[z] > w(n_2)[z]$ where $n_3$ is the ancestor of $n_2$ (since $dv$ or $pq$ score is common and $li$ is smaller; also note that a child node cannot have a $z$ greater than its ancestor).

Thus, this algorithm will always select the unsellected node with the maximum $w$ score which is always a child of an already selected node (i.e., a root of a tree of the unselected forest). Note that because of the monotonicity properties mentioned above, only child nodes of already selected nodes can have the largest score. Thus, progressing iteratively, the $l$ nodes will comprise the optimal PSize-$l$ or DSize-$l$ OSes since this set of $l$ nodes will give the maximum score. 

**Theorem 2.** When the nodes of an OS have monotonically decreasing local importance (i) to their distance from the root (we denote such an OS as monotonic OS(i)), then LASP returns the optimal DSize-$l$ OS.

**Proof.** The $dv$ equation (in contrast to $pq$) is monotonic to $li$. That is $dv(n_1)[z] \leq li(n_1)$ for any $z$. Thus, if an OS is monotonic w.r.t. $li$, it will also be monotonic w.r.t. $w$. Based on Theorem 1, the algorithm will give the optimal result.

**Theorem 3.** When we have a monotonic OS(i) and all nodes have $dv \leq \alpha + 1$, then LASP returns the optimal PSize-$l$ OS.
2-LASP

The runtime cost of LASP is dominated by the numerous updates it applies; each time we add a node (or path) to the snippet, we have to update up to twice each of the remaining nodes. Hereby, we investigate the potential of reducing these updates by exploiting useful monotonicity properties. We observe that because of the proposed equations we expect recurrent monotonicity in the OSs; i.e., recurrent cases where the \(w\) of the parent is bigger than that of its child. More precisely, \(li = af \cdot gi\) where \(af\) is monotonic by definition and additionally both \(dv\) and \(pg\) equations are monotonic to \(z\), where \(z\) is expected to be larger at the bottom levels of the OS tree. Thus, we propose to relax LASP by averaging \(w\) pairs of nodes (hence the name 2-LASP). Namely, we take the average between the current node and the parent instead of the whole path from the current node to the root. As a consequence of this relaxation, updates will be required only on the affected pairs rather on the whole path to the root. Recall that the rationale of considering the average from each node to the root in LASP was to exploit nodes lower on the tree with larger scores than their ancestors.

Specifically, in 2-LASP, at each addition, we update only pairs of scores (denoted as \(2−AP\)), i.e., \(AP\) between the affected node and its parent (instead of all remaining paths towards the root). For example, consider Figure 4; after adding \(n_{11}\), in 2-LASP we only need to update (1) nodes at level 1, except the included \(n_5\) node (since node \(n_1\) is removed), plus \(n_{13}\) (since node \(n_{11}\) is removed) and (2) nodes \(n_2\) and \(n_9\) because of their \(z\) increment to 2. The first set of updates can easily be conducted as they refer to child nodes of the selected path. The second set of nodes cannot be determined without traversing the whole tree. Thus, in \(HF_r\), instead of keeping the frequency of a graph node (i.e., \(fr\)), we keep the set of the OS nodes corresponding to the particular graph node (i.e., \(frS\), which also facilitates the frequency as well). In this example, \(HF_r[\{w\}] . frS = \{n_2, n_9, n_{13}\}\); thus we can find and update all affected nodes in constant time.

The worst case complexity of this algorithm remains the same as LASP. However, in practice it is much faster; for instance, for an exemplary OS of size 735 and \(l=50\), LASP conducts 24,267 updates (in 145ms) whereas 2-LASP performs only 214 updates (in 414ms), resulting to snippets of similar quality. We can easily show that all optimality theorems that hold for LASP also hold for 2-LASP. Empirically, 2-LASP provides snippets of almost the same quality as LASP.

We also investigated the 3-LASP algorithm (i.e., averaging the score of a node with that of its parent and grandparent). We found that 3-LASP is slower than 2-LASP as it requires more operations, while again giving results of similar effectiveness and quality as 2-LASP and LASP. Note that LASP is in fact a 4-LASP algorithm for the DBLP and Google- \(G_{DS}\)s that we use in our experiments, as both graphs have a maximum path length 4.

6. PRELIM-L ALGORITHMS

The aforementioned algorithms operate on the complete OS. Inspired by the prelim-l approach [14], we propose to produce a subset of the OS, denoted as DPPrelim-l (or PPrelim-l) OS, that prunes nodes from the OS which have low probability to be considered for the size-l OS; this saves a lot from the OS generation time and the consequent size-l OS computation time. Note that the prelim-l generation approach of [14] considers the inclusion of the top-l nodes, i.e., the \(l\) nodes with the highest \(li\) in the OS (allowing their repetitions); for clarity, we refer to this algorithm as VPPrelim-l. The direct application of VPPrelim-l is inappropriate here, especially for the PPrelim-l OS, for the following three reasons: (1) it allows the consideration in the top-l set of nodes repeatedly which is against the diversification requirements of PSize and DSsize; (2) it fails to manage the non-monotonic relationship between \(w[1]\) and \(li\) of proportionality (i.e., \(w[1] > li\)), which requires the challenging estimation of frequency per node in the OS; and (3) it does not facilitate further pruning of nodes that are included multiple times.

Recall, however, that the two properties, proportionality and diversity, are based on different equations and thus have different properties in measuring the \(w\) score. More precisely, the proportionality equation is more challenging as it requires the apriori knowledge of \(fr\) per node in order to produce \(w\) scores. Thus we first present more comprehensively a generalized version of the VPPrelim-l approach, which addresses proportionality. This algorithm, denoted as PPPrelim-l, considers the frequency and respective frequency upper bounds of nodes in an OS, in order to produce the so called PPrelim-l OS. Then, we present more synthetically the DPPrelim-l algorithm with the required specializations and simplifications in order to produce DPPrelim-l OS.

6.1 PPrelim-l

The computation of the optimal PSize-l OS is very expensive and, as a consequence, so is the computation of any PPrelim-l OS that is guaranteed to include the optimal PSize-l OS. Thus, we resort to a heuristic that aims to generate a PPrelim-l OS that includes at least the \(l\) distinct graph nodes with the largest \(w[1]\) scores (denoted as the \(top_l\) set). The rationale is that while searching for \(top_l\), and by appending the retrieved nodes to the PPrelim-l OS, we will generate a good superset of the PSize-l OS. The constraint of including only distinct nodes in \(top_l\) is necessary in order to facilitate eventual diversity. We generate the PPrelim-l OS by extending the complete OS generation algorithm (see Algorithm 4, described in [12]) to include three pruning conditions. For this purpose, cheap pre-computed indexes with statistics per \(G_{DS}\) relation and per \(n_i\) node are employed (to be described shortly). More precisely, we traverse the \(G_{DS}\) graph in a breadth first order, according to Algorithm 3. Let \(l\) be the distinct node with the \(l^{th}\) largest \(w[1]\) retrieved so far. If the current tuple \(n_i\) is greater than \(l\), \(n_i\) is added to the \(l\)-sized priority queue \(W_l\) which manages the \(top_l\) set (line 7, function AddNode of Algorithm 3). We also maintain \(HF_r\), a hash table managing the frequency of nodes. By considering the computed frequency \(HF_r[n_i]\) of a node \(n_i\) so far, which is less
than or equal to actual frequency of \( n_i \), in fact we consider the lower bound of \( w \). In the sequel, we first introduce the constructed indexes and then the pruning conditions.

### Index per \( G^{DS} \) Relation

The following indexes are calculated during pre-processing per \( G^{DS} \) relation \( R_i \): (1) \( \max(R_i) \), i.e., the maximum value of \( l_i \) in \( R_i \); (2) \( \max(K_i) \), i.e., the maximum value of \( \max(R_i) \) of all \( R_i \)'s descendant nodes in \( G^{DS} \) or 0 if \( R_i \) has no descendants (i.e., \( R_i \) is a leaf node); (3) \( \max(R_i) \), i.e., the upper bound of joins of a node in \( R_i \) can have with any \( n_i \). During pre-processing, we can determine only for some cases these bounds; e.g., when up to 1 node from a relation (e.g., \( R_{parent} \)) can only join with \( n_i \). Otherwise, we assume infinite joins (e.g., the same co-author may appear in many papers); e.g., we assume \( UBFr(\text{Co-author})=\infty \). In order to facilitate the calculation of \( UBFr(R_i) \), we also introduce the \( c(R_i) \) variable which is the summation of tuples from \( R_i \) that join with \( n_i \). Thus, for N:1 relationships, \( c(R_i) = c(R_{parent}) \), where \( c(R^{DS}) = 1 \) and \( R_{parent} \) is the parent relation of \( R_i \). Thus, given \( c(R_{parent}) \) for cases where \( UBFr(\text{Co-author})=\infty \), we can estimate \( UBFr(R_i) \) as a function of \( c(R_{parent}) \), i.e., \( UBFr(R_i) = c(R_{parent}) \) (denoted as \( \rightarrow c(R_i) \) in Figure 2); this association will be useful later, during the online calculation of tighter frequency bounds. Also note that since, we only need \( c(R_{parent}) \), we do not need to calculate \( c(R_i) \) for leaf nodes (thus we denote their \( c(.) \) as \( 0 \) in Figure 2).

#### Index per \( n_i^{DS} \) node

During pre-processing we also maintain the following indexes per \( n_i^{DS} \): (1) \( \max(n_i^{DS}) \) is the maximum \( l_i \) for all nodes in an OS (excluding \( n_i^{DS} \)); e.g., in Figure 2, \( \max(n_i^{DS}) = 0.6 \) is found in relation Paper. Note that this score overrides the maximum \( l_i \) score of all \( G^{DS} \)relations (i.e., \( \max(R_i) \) and \( \max(R_j) \)). (2) \( m_w(n_i^{DS}) \) is the maximum \( w[1] \) of any node in the OS (excluding \( n_i^{DS} \)). Similarly to \( \max(n_i^{DS}) \), this score is considered as the upper bound of \( w \) of all nodes of all relations (e.g., in Figure 2, \( m_w(n_i^{DS}) = 0.7 \) is found in relation Co-Author). (3) \( mFr(n_i^{DS}) \), i.e., the maximum frequency of any node in the OS that belongs to a relation \( R_i \) with \( UBFr(R_i) \). For instance, in the example of Author \( G^{DS} \) in Figure 2, \( mFr(n_i^{DS}) = 21 < UBFr(R_i) = 33 \), thus we can use this as a tighter bound and thus override the \( UBFr(R_i) \) bound.

### Variables and data structures

Let \( R_i(n_j) \) be the subset of \( R_i \) that joins with \( n_j \), and \( R_i(n_j) \) be the relation where to \( n_j \) belongs. \( HF_r \) is a hash table which indexes for each graph node \( n \) \( f(n) \), the computed frequency of \( n \) in the OS so far.

While processing \( n_i \) (in \( R_i(n_j) \)) against a relation \( R_i \) with \( UBFr(R_i) \), we try to get a tighter bound than \( UBFr(R_i) \) and \( mFr(n_i^{DS}) \), denoted as \( UBFr(n_j, R_i) \). For this purpose we maintain the current maximum frequency, denoted as \( cmFr(R_i) \), a node was found so far from \( R_i \) (i.e., from processing predecessor nodes \( n_1, \ldots, n_{j-1} \) of \( n_j \) against \( R_i \), i.e., from their respective \( R_i(n_1), \ldots, R_i(n_{j-1}) \) sets). For instance, let the node \( n_j \) from \( R_i \) be the most frequent among all \( R_i(n_1), \ldots, R_i(n_{j-1}) \) sets that was found so far and it was found 10 times; thus \( cmFr(R_i) = 10 \). Given \( cmFr(R_i) \), \( UBFr(R_i, n_j) \) assumes that \( n_j \) will appear in all the remaining sets \( R_i(n_j), \ldots, R_i(n_{j+1}) \) after processing \( n_j \). At the beginning, \( UBFr(n_j, R_i) \) can be very loose, so we compare it with \( mFr(n_i^{DS}) \), to keep the minimum of the two (lines 6-12).

Then, we denote by \( UBw(n_j, R_i) \) the upper bound of the \( w[1] \) score that can be obtained from \( R_i(n_j) \) (line 13). We denote as \( UBw(n_j, R_i) \) the upper bound of \( w[1] \) of all nodes form \( R_i \)'s descendant relations that can join with \( n_j \) or if \( R_i \) has no descendants. Similarly to \( UBw(n_j, R_i) \) calculation, \( dUBw(n_j, R_i) \) \( f(.) \) is defined by Equation 4 and \( z = 1 \) (line 14). Also note that, if \( R_i \) is a leaf node on \( G^{DS} \) then \( mmax(R_i) = 0 \) and thus \( UBw(n_j, R_i) = 0 \). \( UBw(n_j, R_i) \) and \( dUBw(n_j, R_i) \) bounds are specializations of \( max(R_i) \) and \( mmax(R_i) \) that have been used in prelim-l [14] in pruning conditions 1 and 2 respectively; however, they are tighter bounds as they are specific for the given \( n_i^{DS} \).

#### Pruning Conditions

Each time we further process a node \( n_j \) we employ three pruning conditions:

**Pruning Condition 1.** If \( t > UBw(n_j, R_i) \) and \( t > dUBw(n_j, R_i) \), then there is no need to traverse the sub-tree starting at \( R_i \) (line 15)

**Pruning Condition 2.** We can limit the amount of tuples returned by an \( R_i(n_j) \) join, up to \( t \) nodes (i.e., avoid computing the entire join of \( n_j \) with \( R_i \) if we can infer that none of \( R_i \)'s descendants (if any) can be fruitful for the topic, i.e., when \( t > UBw(n_j, R_i) \) (line 16). Then, we can extract only up to \( t \) tuples with \( w[1] \) greater than \( t \) (line 17).

**Pruning Condition 3.** When pruning condition 2 holds, we can safely extract only up to \( t \) nodes. However, when \( UBFr(R_i) \), it is still possible that we extracted nodes which have already been added on the PPrelim-l and thus their \( w[1] \) is actually used (where \( z > 1 \)). Thus, we introduce a new pruning condition that checks first if a node \( n_j \) already exists and then whether \( w[1] \) is less than \( t \) (line 21).
6.2 DPrelim-l OS

We simplify the previous algorithm by excluding all work concerning calculating or upper bounding the frequencies of nodes. For instance, indexes such as UBFr(Ri), mFr(nDS) and calculations of UBFr(nj, Ri) are not required. We adjust accordingly our algorithm to include these alterations (e.g., exclude calculations of UBFr(nj, Ri)); functions (line 13, 14) use Eq. 2 and 4, etc.).

6.3 Discussion

In terms of cost, in the worst case, we need up to n extractions of nodes, where n is the amount of nodes in the complete OS. In practice, however, there can be significant savings if the top-l tuples are found early and large sub-trees of the complete OS are pruned. The PPrelim-l (resp. DPrelim-l) OS does not essentially contain the optimal PSize-l (resp. DSize-l) OS; in practice, however, we found that this is the case in most problem instances. This means that the PSize-l (resp. DSize-l) OS computation algorithms most likely give the same results when applied either on the PPrelim-l (resp. DPrelim-l) OS or the complete OS. Similarly to LASP and 2-LASP algorithms, DPrelim-l and PPrelim-l algorithms will produce optimal results (i.e., supersets of the respective optimal size-l OSs) when we have a monotonic OS(l) and monotonic OS(\ell) respectively. Note that PPrelim-l cannot return an optimal solution when we have simply a monotonic OS(\ell) because the proportionality equation (in contrast to to the diversification equation) is not monotonic to li. The following theorem proves that if the li scores of modes are monotonic then the computed DPrelim-l OS will be optimal.

**Theorem 4.** If the local importance scores of nodes (li) are monotonically non-increasing with respect to the distance of the nodes to nDS, then DPrelim will produce the optimal DPrelim-l OS (i.e., a superset of the optimal DSize-l OS).

**Proof.** The DPrelim OS will include the (1) top-l set, i.e., l distinct nodes with the largest w[1] scores where t is the minimum (top-l), and (2) also all repetitions of top-l nodes with w[2] < t, denoted as repw (since we only prune nodes with w[1] < t and w[2] < t in pruning condition 2 and 3 respectively). When we have a monotonic OS, we can produce the optimal DSize-l OS by using the LASP algorithm (Theorem 2) as follows. Initially, we will include j distinct nodes with the largest li scores (as they also correspond to the largest AP since w[1] = li, where j < l); where, a subset of these j nodes may have fr > 1. All these nodes are members of the top-l by definition. Then, consider that if the next node to be added (according to LASP) is a repetition of a node (i.e., we include it considering its w[2]; then, this node is member of repw by contradiction since it should have w[2] > t (as otherwise another distinct node would have been selected). Thus, we conclude that the optimal DSize-l will comprise nodes from either top-l or repw nodes which are included in DPrelim-l by definition.

**Theorem 5.** Similarly, based on Theorems 1 and 4 we can easily see that the PPrelim can produce an optimal PPrelim-l OS if we have a monotonic OS(\ell) and all nodes have fr(nj) < α + 1.

7. EXPERIMENTAL EVALUATION

We experimentally evaluate the proposed snippets and algorithms. We emphasize on effectiveness comparisons between the two types of diversified snippets and also against the non-diversified size-l snippets [14]. Firstly, we thoroughly investigate the effectiveness and usability of the produced snippets with the help of human evaluators. Then, we evaluate the quality of the size-l OSs produced by the greedy heuristics. Finally, we comparatively investigate the efficiency of the proposed algorithms.

We used two databases: DBLP and Google+. The two databases have 3M and 14M tuples and occupy 513MB and 800MB on the disk, respectively. Google+ dataset was constructed by combining real data extracted from Google+ (i.e., users, activities and reactions which are publicly available). Followers and circles which were dealt as private by Google+ (and thus were publicly unavailable) were generated from the synthetic SNAP dataset. We calculate global importance by using global ObjectRank [3]. For the DBLP dataset we use the default setting used in [3] and [14], i.e., the G^A shown in Figure 10(a) and d = 0.85 and for Google+ the G^A presented in Figure 10(b) and also d = 0.85. We calculate αf as in [14], (alternatively an expert can define the G^DS’s and affinity manually, i.e., to select which relations to include in each G^DS and their affinity). For proportionality, we use α = 2. We used Java, MySQL, and a PC with an AMD Phenom 9650 2.3GHz (Quad-Core) processor and 8GB of memory.

7.1 Effectiveness

We conducted an effectiveness evaluation with the help of human evaluators. The evaluators were professors and researchers from Universities in Cyprus and Hong Kong. Since the DBLP database includes data about real people, we asked the DSs themselves where possible (i.e., twelve authors or students listed in DBLP) to suggest their own Author DSize-l and PSize-l OSs. The rationale of this evaluation is that the DSs themselves (even their students) have the best knowledge of their work and can therefore provide accurate summaries. For Google+, we presented 10 random OSs to nine evaluators and asked them to DSize-l and PSize-l them. First, we familiarized them with the concepts of OSs in general and the three types of size-l OSs. 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. In addition, we explained that DSize-l OSs and PSize-l OSs consider diversity and proportionality respectively. However, we avoided to discuss the advantages or disadvantages of the three types as to avoid any bias. In order to assist them with their tasks we provided them useful information per node, such as fr, li, dv[1], ps[1] and w[1]. For instance, we provided them, with the amount of times the co-author C. Faloutsos appears in the M. Faloutsos core) processor and 8GB of memory.

7.1.1 Precision and Recall

We provided evaluators with OSs and asked them to DSize-l and PSize-l them for l = 10, 15, 30. Figure 5 measures the effectiveness of our approach as the average percentage of the nodes that exist in both the evaluators’ size-l OS 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. Figures 5(a) and 5(b) plot the recall of the DSize-l and PSize-l for DBLP Author and Google+ User G^DS’s. On average, the effectiveness of DSize-l and PSize-l OSs ranges from 67% to 82% for all cases, which is very encouraging. The results of Figure 5 are obtained using the LASP algorithm (as the BF algorithm was prohibitively expensive). We omit results obtained by our other algorithms as they do not vary from these results. For instance, the 2-LASP algorithm gave almost identical results as LASP and the use of DPrelim-l OSs or PPrelim-l OSs had no impact on effectiveness. As we show 2 http://snap.stanford.edu/data/egonets-Gplus.html
later, they have very minor impact on the quality of the computed snippets.

7.1.2 Usability Test

We conducted a comparative study of the usability of the three types that verifies users’ preference for DSize-l and PSize-l OSs over size-l OSs. Usability is the ease of use and learnability of a human-made object; namely, how efficient it is to use (for instance, whether it takes less time to accomplish a particular task), how easy it is to learn and whether it is more satisfying to use\(^1\). More precisely, for a given OS, we measured the ease of use of all types through a usability test. We presented to users the three versions of size-l OSs in a random order to avoid any bias and we also gave them four tasks to complete for each OS. Then, we asked them to give a score in a scale of 1 to 10 and also to justify their answers, where possible, the usability of the three approaches when completing these tasks. Namely, to score them considering (1) the ease of accomplishing each task, (2) how easy and (3) satisfying they are to learn and use.

For instance, the first task (T1) was to score the general use of all types; namely which one they prefer as a representative and informative snippet. For this purpose, we emphasized again that snippets should be short, stand-alone and meaningful synopsis of the most important and representative information about the particular DS; we avoided to discuss any advantages/disadvantages. The rest of the tasks were to extract information about the DSs. For instance, task 2 (T2), focusing on the Author relation, was to determine a couple of the most frequent co-authors of a given author (e.g., whether C. Faloutsos and S. Krishnamurthy are among the most frequent collaborators of M. Faloutsos); task 3 (T3) to determine a couple of the most important co-authors (e.g., whether C. Faloutsos and S. Madden are among the most important co-authors of M. Faloutsos); and task 4 (T4) to determine the most frequent journal/conference the DS has published. Similar tasks were used for Google+.

Figure 6 averages the evaluators’ usability scores of the three methods per \(G^{DS}\) and per task. The results show that evaluators preferred firstly PSize-l OSs, then DSize-l OSs and lastly size-l OSs for both datasets; they also preferred size \(l=30\) over \(l=15\). For instance, the average scores of all tasks and both values of \(l\) for PSize-l, DSize-l, and size-l OSs are 7.5, 6.9, and 6.0 respectively for the Author \(G^{DS}\). The evaluators also provided justifications for their scores; we summarize them for each type and \(l\). They explained that in general they prefer the concept of PSize-l OS as it also considers frequent nodes; which is a property other types do not consider. However, they pointed out although the inclusion of repeated frequent items is informative, it comes with the cost of excluding other important nodes. They found DSize-l very useful in covering the most important elements of an OS; however, they pointed out that rare but important elements may appear which again can be to some extend misleading. They found the non-diversified size-l summaries [14] more misleading as very important nodes are too dominant in them. The evaluators explained that \(l = 30\) results include more information in the snippets, giving a better representation of frequent and important information of an OS.

7.2 Quality of Snippets

We now compare the holistic importance \(Im()\) scores of DSize-l and PSize-l OSs produced by the greedy methods. More precisely, the results of Figure 7 represent the average holistic scores for 10 random OSs per \(G^{DS}\). The average size (i.e., the amount of nodes) of OSs is also indicated (denoted as \(|OS|\)). The results show that in most cases the results of LASP and 2-LASP are of very similar quality, i.e., they have similar holistic \(Im()\) scores. The evaluation also reveals that using the DPrelim-l and PPrelim-l OSs result very minor quality loss compared to using the complete respective OSs; e.g., by using LASP on either a complete OS or the corresponding DPrelim-l OS, we obtain a DSize-l OS of the same \(Im(DS)\). In
the case of PSize-l OSs, the maximum score difference between 2-LASP and LASP is between 20 and 15.5. We did not compare with the optimal results as the BF-l algorithm is too expensive.

7.3 Efficiency

We compare the run-time performance of our greedy algorithms in Figure 8. We used the same OSs as in Section 7.2 (i.e., the same 10 OSs per $G^5$). The Figures (a)-(e) show the costs of our algorithms for computing DSize-l (resp. PSize-l), excluding the time required to generate and pre-process OSs (i.e., the generation of scores (e.g., $u$ and $AP$ scores), priority queue $W$, etc.), where each algorithm operates on. More precisely, Figures 8(a)-(d) show the costs of our algorithms for computing size-ls from OSs of the two $G^5$s with various sizes and using a range of $l$ values. The average sizes of the OSs on which the algorithms operate are indicated in brackets for each $G^5$. Figure 8(e) shows the scalability for Author PSize-l of different sizes, after fixing $l=10$ (analogous results were obtained from User $G^5$ and DSize-l and thus we omit them). Each value on the x-axis represents an OS size (and the corresponding PPrelim-10 size). Comparing these numbers, we can get an indication of preliminary OSs savings; e.g., the OS with size 1,300 has a PPrelim-10 size 157 (i.e. 11% of the size of the complete OS). Finally, Figure 8(f) breaks down the cost to OS generation and pre-processing time (bottom of the bar) and size-l computation (top of the bar) for each method for PSize-l. The figure also shows the average sizes of the complete OSs and the PPrelim-l for $l = 10$ and $l = 50$, respectively. The preliminary OS generation is always faster than that of the complete OS; for instance the PPrelim-5 OS’s size is approximately 10% of the size of the complete OS and its generation can be done up to 2.5 times faster. Also, 2-LASP is always faster at both phases (i.e. during OS generation and pre-processing and during size-l calculation) as at both phases more operations are required by LASP (recall that during pre-processing, the $AP$ of a node in LASP corresponds is the path to the root, whereas in 2-LASP is the node with its parent only). In general as expected, both the OS size and $l$ negatively affect the cost.

The cost of the BF-l algorithm becomes unbearable for moderate OSs sizes and values of $l$. For instance, although using BF-l we could get results for $l=5$ (e.g., 16ms for the Author $R^5$), we had to terminate the algorithm for $l\geq10$ as it exceeded 30min of running. In summary, the BF-l algorithm is not practical at all whereas our greedy algorithms are very fast and as we showed in Section 7.2, their results are snippets of high quality. In addition, the use of preliminary OSs and 2-LASP is constantly a better choice over the complete OSs and LASP respectively since they are always faster with a negligible quality loss.

8. CONCLUSION AND FUTURE WORK

In this paper, we introduced and investigated the effectiveness and efficiency of two novel types of size-l OSs, namely DSize-l OSs and PSize-l OSs. We proposed a brute force algorithm, two efficient greedy heuristics and a preprocessing strategy that restricts processing on only a subset of the OS. Finally, we conducted a systematic experimental evaluation on the DBLP and Google+ datasets that verifies the effectiveness, approximation quality and efficiency of our techniques. The evaluation verified that the two novel snippets are preferred by human evaluators over non-diversified size-l OSs [14]. A direction of future work concerns the investigation of inter-diversity and inter-proportionality among a set of query results. For instance, for $Q_1$ we get three OSs, one per Faloutsos brother; we can diversify M. Faloutsos DSize-l by avoiding information included in the C. Faloutsos DSize-l. Another challenging problem is the combined size-l and top-k ranking of OSs.

9. ACKNOWLEDGMENTS

We would like to thank our evaluators for their time and generous help during the experimental evaluation of our work. George Fakas was supported by grants HKU 715413E and HKUST 617412 from Hong Kong RGC. Zhi Cai was supported by Beijing Natural Science Foundation grants 4144068 and SRF for ROCS, SEM.

10. REFERENCES

Figure 9: The Google+ User $G^D$

(a) The DBLP $G^A$

(b) The Google+ $G^A$

Appendix

Algorithm 4 The OS Generation Algorithm

\begin{enumerate}
\item \textbf{enQueue}($Q, n^{DS}$)  \quad \triangleright \text{Queue } Q \text{ facilitates breadth first traversal}
\item add $n^{DS}$ as the root of the OS
\item while \textbf{isEmptyQueue}(Q) do
\item \textbf{deQueue}(Q)
\item \textbf{for each} child relation $R_i$ of $R(n_j)$ in $G^{DS}$ do
\item get $R_i(n_j)$
\item \textbf{for each} tuple $n_i$ of $R_i(n_j)$ do
\item \textbf{enQueue}(Q, $n_i$)
\item \textbf{add} $n_i$ on OS as child of $n_j$
\item \textbf{return} OS
\end{enumerate}