ENCYCLOPEDIA WITH SEMANTIC COMPUTING AND ROBOTIC INTELLIGENCE
Vol. 2, No. 2 (2018) 1750002 (20 pages)

DOI: 10.1142/S2529737617500022

World Scientific
www.worldscientific.com

Object summaries for keyword search

Georgios J. Fakas*,‡ and Zhi Cai†,§
*Department of Information Technology, Uppsala University, Sweden
†Beijing University of Technology, P. R. China
‡georgios.fakas@it.uu.se
§caiz@bjut.edu.cn

Accepted 20 September 2016; Published 29 June 2017

The abundance and ubiquity of graphs (e.g., semantic knowledge graphs, such as Google's knowledge graph, DBpedia; online social networks such as Google+, Facebook; bibliographic graphs such as DBLP, etc.) necessitates the effective and efficient search over them. Thus, we propose a novel keyword search paradigm, where the result of a search is an Object Summary (OS). More precisely, given a set of keywords that can identify a Data Subject (DS), our paradigm produces a set of OSs as results. An OS is a tree structure rooted at the DS node (i.e., a node containing the keywords) with surrounding nodes that summarize all data held on the graph about the DS. An OS can potentially be very large in size and therefore unfriendly for users who wish to view synoptic information about the data subject. Thus, we investigate the effective and efficient retrieval of concise and informative OS snippets (denoted as size-*l* OSs). A size-*l* OS is a partial OS 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, 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, besides the importance of each node, we also consider its pairwise relevance (similarity) to the other nodes in the OS and the snippet. We conduct an extensive evaluation on two real graphs (DBLP and Google+). We verify effectiveness by collecting user feedback, e.g., by asking DBLP authors (i.e., the DSs themselves) to evaluate our results. In addition, we verify the efficiency of our algorithms and evaluate quality of the snippets that they produce.

Keywords: Diversity; proportionality; snippets; summaries.

1. Introduction

Keyword search on the web (W-KwS) has dominated our lives, as it facilitates users to find effectively information using only keywords. For instance, the result for query Q_1 = "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 searcher's actual information needs, thus preventing the need to retrieve the actual result.[1]

The keyword search paradigm has also been introduced in relational databases (R-KwS). (e.g., Ref. 2). 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 Q_2 = "Faloutsos" + "Agrawal" over the DBLP database returns tuples Faloutsos and Agrawal from the Author table and their associations through co-authored papers (Example 2). 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.; Example 3). A DS is an entity (e.g., an individual, paper, product, etc.) which has its identity in a tuple which is the result (i.e., subject) of the keyword search. R-KwS only returns tuples containing the keywords (in this case only Faloutsos Author tuples), and hence fails to address search for the *context* of most important tuples around a central tuple (i.e., a DS).

Example 1. Q_1 "Faloutsos" using a W-KwS (Google)

Christos **Faloutsos**
SCS CSD Professor's affiliatons, research, projects, publications and teaching.
www.cs.cmu.edu/∼christos/ - 9k
Michalis **Faloutsos**
The Homepage of Michalis **Faloutsos** . . . Interesting and Miscallaneous Links · Fun pictures · Other **Faloutsos** on the web; The Teach-To-Learn Initiative:
www.cs.ucr.edu/∼michalis/ - 5k

(*Continued*)

(*Continued*)

Petros **Faloutsos**
Courses · Press Coverage · Publications · Research Highlights · Awards · MAGIX Lab · Curriculum Vitae · Family · Other **Faloutsos** on Web.
www.cs.ucla.edu/∼pfal/ - 4k
. . .

Example 2. Q_2 using an R-KwS

Author: Christos **Faloutsos**, **Paper**: Efficient similarity search in sequence databases, **Author**: Rakesh **Agrawal**.

Author: Christos **Faloutsos**, **Paper**: Method for high-dimensionality indexing in a multi-media database, **Author**: Rakesh **Agrawal**.

Author: Christos **Faloutsos**, **Paper**: Quest: A project on database mining, **Author**: Rakesh **Agrawal**.

Example 3. Q_1 using an R-KwS

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

Example 4. The OS for Michalis Faloutsos

Author: Michalis **Faloutsos**
Paper: On Power-law Relationalships of the Internet Topology.
Conference: SIGCOMM. **Year**: 1999.
Co-Author(s): Christos Faloutsos, Petros Faloutsos.
Cites: Building Shared. . ., **Cited by**: The Structure. . .,
Paper: BLINC: Multilevel Traffic Classification in the Dark.
Conference: ACM SIGCOMM Computer Comm. Review **Year**:2005.
Co-Author(s): T. Karagiannis, K. Papagiannaki.
Cites: A Parametrizable methodology. . ., **Cited by**: P4P: Provider. . .,
Paper: Transport Layer Identification of P2P Traffic.
Conference: SIGCOMM. **Year**:2004.
Co-Author(s): T. Karagiannis, A. Broido.
Cites: Their Share: Diversity. . ., **Cited by**: Internet Traffic. . . . . .
. . .
. . .

Example 5. The size-15 OS for Michalis Faloutsos

Author: Michalis **Faloutsos**
Paper: On Power-law Relationalships of the Internet Topology.
Co-Author: *Christos Faloutsos*,. . .
Paper: Power Laws and the AS-Level Internet Topology.
Co-Author: *Christos Faloutsos*,. . .
Paper: ACM SIGCOMM' 99. **Co-Author**: *Christos Faloutsos*,. . .
Paper: Information survival thr.... **Co-Author**: *Christos Faloutsos*,. . .
Paper: The Connectivity and Fault. . . **Co-Author**: *Christos Faloutsos*,. . .
Paper: BGP-lens: Patterns and An. . . **Co-Author**: *Christos Faloutsos*,. . .
Paper: The eBay Graph: How Do.... **Co-Author**: *Christos Faloutsos*,. . .

Example 6. The DSize-15 OS for Michalis Faloutsos

Author: Michalis **Faloutsos**
Paper: On Power-law Relationalships of the Internet Topology.
Conference: SIGCOMM. **Year**: 1999.
Co-Author: Christos Faloutsos,....
Paper: Information Survival Threshold in Sensor and P2P Networks.
Co-Author: S. Madden, ..., **Conference**: INFOCOM.
Paper: Power Laws and the AS-Level Internet Topology.
Conference: IEEE/ACM Tr. Netw. **Year**: 2003.
Co-Author: Christos Faloutsos,..
Paper: Network Monitoring Using Traffic Dispersion Graphs.
Co-Author: M. Mitzenmacher, ...**Conference**: SIGCOMM.

Example 7. The PSize-15 OS for Michalis Faloutsos

Author: Michalis **Faloutsos**
Paper: On Power-law Relationalships of the Internet Topology.
Conference: SIGCOMM. **Year**: 1999.
Co-Author: Christos Faloutsos,....
Paper: Denial of Service Attacks at the MAC Layer. . .
Co-Author: S. Krishnamurthy,. . ., **Conference**: MILCOM.
Paper: Power Laws and the AS-Level Internet Topology.
Conference: IEEE/ACM Tr. Netw.
Co-Author: Christos Faloutsos,..
Paper: Reducing Large Internet Topologies for Faster Simulations
Co-Author: S. Krishnamurthy, L. Cui,. . . **Conference**: NETWORKING.

In view of this, in Refs. 3 and 4 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 n^{DS} containing the keywords (e.g., Author tuple Faloutsos) as the root node and its neighboring tuples, containing additional information (e.g., his papers, co-authors, etc.), as child or descendant nodes. 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 n^{DS} via a short path. For instance, the result for q is a set of OSs: one for each Faloutsos brother. Example 4 illustrates the OS for Michalis Faloutsos. Note that the OS paradigm is in more analogy to W-KwS, compared to R-KwS. For instance, Example 4 resembles a web page (as it includes comprehensive information about the DS). Therefore, for the non-technical users with experience only on web keyword search, the OS paradigm will be friendlier and also closer to their expectations. 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. Another application of this summarization concept is on semantic knowledge graphs.[5,6]

In Refs. 7–9, OS snippets were proposed (denoted as size-l OSs). Size-l OSs are composed of only l important nodes so that (1) the summation of their scores is maximized and (2) all l nodes are connected to the OS root (i.e., n^{DS}). Example 5 illustrates the size-l OS for M. Faloutsos with $l = 15$ on the DBLP database. According to Ref. 7, a size-l OS should be a standalone sub-graph of the complete OS 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.

However, this selection criterion (i.e., maximizing importance score) can render such snippets ineffective. For instance, in Example 5, 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, we also propose two novel snippets, namely *diverse* and *proportional* size-l OSs denoted as *DSize-l* OS and *PSize-l* OS, respectively. More precisely, in a DSize-l OS, we favor diversity by penalizing repetitions of relevant nodes. For instance, the DSize-l OS of Example 6 includes C. Faloutsos only twice, allowing the appearance of other important co-authors as well. In a PSize-l OS, we favor proportionality, i.e., a frequent relevant node should be analogously represented, facilitating diversity at the same time. Similarly, the PSize-l OS of Example 7 includes also frequent co-authors S. Krishnamurthy and L. Cui who do not appear at all in the DSize-l OS. To compute them, we calculate a *combined* score per node, which integrates (1) importance, (2) affinity to the data subject node n^{DS} and (3) diversity or proportionality.

For the diversity and proportionality scores calculation, we employ two types of pairwise relevance: *Similarity* (denoted also as *sim*) and *Equality* (denoted as *equi*). More precisely, *sim* is the textual similarity between nodes (e.g., Jaccard similarity); e.g., two papers with common keywords are similar. Note that textual similarity on Author's names makes little sense here; e.g., two authors with a common surname are still two different persons. Thus, we also use *equi* as a *binary* relevance function, i.e., two OS nodes that correspond to the same graph node (e.g., the same Author appearing many times in an OS) have *equi*-relevance 1, otherwise their *equi*-relevance is 0. We say that a snippet is an *equi* size-l OS if it considers only *equi* relevance (e.g., an *equi* DSize-l); we say that a snippet is a *sim* size-l OS if it considers both *sim* and *equi* relevance (e.g., *sim* DSize-l).

The efficient generation of DSize-l or PSize-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[7]).

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 that DSize-l and PSize-l are more usable than the respective size-ls OSs, which disregard diversity. In addition, we investigated in detail and verified the efficiency and approximation quality of our algorithms.

2. Object Summaries

According to the keyword search paradigm of Ref. 4, an object summary (OS) is generated for each node (tuple) n^{DS} found in a graph (database) that contains the query keyword(s) (e.g., "Michalis Faloutsos" node of Author relation in the DBLP database). An OS is a tree having n^{DS} as a root, the nodes that link to n^{DS} through foreign keys as its children, the nodes that link to the children recursively as descendant nodes. To construct an OS, the relation R^{DS} (e.g., the Author relation) that holds n^{DS} and those that link to R^{DS} via foreign keys are used. First, a Data Subject Schema Graph G^{DS} is generated. Figure 2 illustrates the G^{DS} for the Author relation of the DBLP database, whose schema is shown in Fig. 1. 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. Examples of such replications are relations PaperCitedBy, PaperCites and co-author on author G^{DS} (see Fig. 2). (User evaluation in Ref. 4 verified that the tree

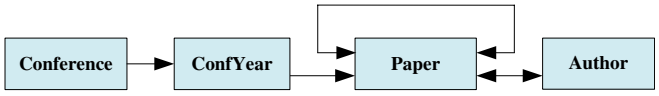


Fig. 1. The DBLP database schema.

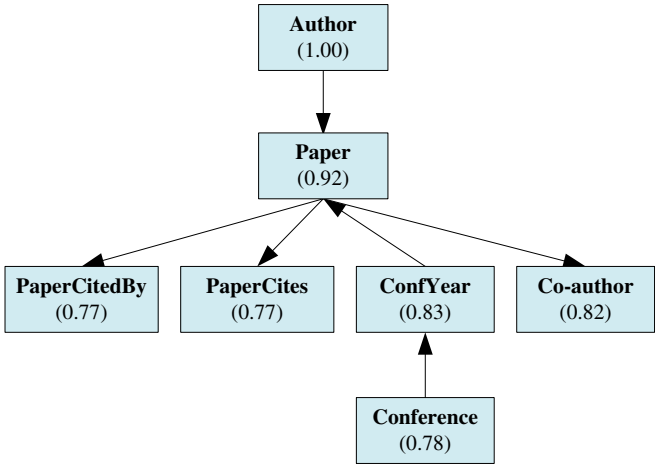


Fig. 2. The DBLP author G^{DS} (affinity).

format (achieved via such replications) increases significantly friendliness and ease of use of OSs.)

In other words, a G^{DS} is a "*treelization*" of the database schema, where R^{DS} becomes the root, R^{DS}'s neighboring relations become child nodes and so on. In order to generate the OS, the relations from G^{DS}, which have high *affinity* with R^{DS} are used. The affinity of a relation R_i to R^{DS} can be calculated by the formula:

$$af(R_i) = \sum_j mw_j \cdot m_j \cdot af(R_{Parent}), \qquad (2.1)$$

where j ranges over a set of measures $(m_1, m_2, \ldots, m_n)$ and their corresponding weights $(mw_1, mw_2, \ldots, mw_n)$, and $af(R_{Parent})$ is the affinity of R_i's *parent* to R^{DS}. The measures' 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 measures: 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 measures 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 a threshold θ, a subset of G^{DS} can be produced that includes only the relations of affinity at least θ to R^{DS}. The OS for a tuple n^{DS} in R^{DS} is generated by traversing the G^{DS} starting from n^{DS} (e.g., by joining n^{DS} with the neighboring relations of R^{DS}; Algorithm 1). 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 4 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[10] and ValueRank.[11] Due to the "treelization" 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_i of an OS, we use function $g(n_i)$ to denote the corresponding tuple v in the database. Thus, for two OS nodes n_i and n_j, we may have $g(n_i) = g(n_j) = v$. We also denote as $fr(v)$ (i.e., $fr(g(n_i))$, or simply $fr(n_i)$) the frequency of tuple v in the given OS.

Algorithm 1. The OS Generation Algorithm

OS Generation (n^{DS}, G^{DS})

```
1:  enQueue(Q, n^DS)            ▷ Queue Q facilitates breadth first traversal
2:  add n^DS as the root of the OS
3:  while !(isEmptyQueue(Q)) do
4:      n_j=deQueue(Q)
5:      for each child relation R_i of R(n_j) in G^DS do
6:          get R_i(n_j)
7:          for each tuple n_i of R_i(n_j) do
8:              enQueue(Q, n_i)
9:              add n_i on OS as child of n_j
10: return OS
```

3. Size-l OSs

According to Ref. 7, given an OS and an integer l, a candidate size-l OS is any subset of the OS composed of l nodes, such that the l nodes form a tree rooted at n^{DS}. In Ref. 7, we 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 is, 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 5, if we exclude the 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 the sum of the local importance scores of its nodes, i.e., $Im(Sl) = \sum_{n_i \in Sl} li(n_i)$. The local importance of a node n_i is the affinity-weighted global importance of n_i, i.e., $li(n_i) = af(n_i) \cdot gi(n_i)$. The affinity $af(n_i)$ of a node n_i equals the affinity of the relation where n_i belongs (Eq. (2.1)); global importance was defined in Sec. 2.

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 $O(nl^2)$ time where n is the amount of nodes in the OS) and greedy algorithms were proposed in Ref. 7.

3.1. *Greedy algorithms*

Since the DP algorithm does not scale well, in this section, we investigate greedy heuristics that aim at producing a high-quality size-l OS, not necessarily being the optimal. A property that the algorithms exploit is that the local importance of tuples in the OS (i.e., $Im(\mathrm{OS}, t_i)$) usually decreases with the node depth from the root t^{DS} of the OS. Recall that $Im(\mathrm{OS}, t_i)$ is the product $Im(t_i) \cdot Af(t_i)$, where $Im(t_i)$ is the global importance of tuple t_i and $Af(t_i)$ is the affinity of the relation that t_i belongs to. $Af(t_i)$ monotonically decreases with the depth of the tuple since $Af(R_i)$ is a product of its parent's affinity and $Af(R_i) \leq 1$ (cf. Eq. (2.1)). On the other hand, the global importance for a particular tuple is to some extent unpredictable. Therefore, even though the local

importance is not monotonically decreasing with the depth of the tuple on the OS tree, it has higher probability to decrease than to increase with depth. Hence, it is more probable that tuples higher on the OS to have greater local importance than lower tuples. Moreover, note that due to the non-monotonicity of OSs, existing top-k techniques such as[11–13] cannot be applied.

3.2. *Bottom-up pruning size-l algorithm*

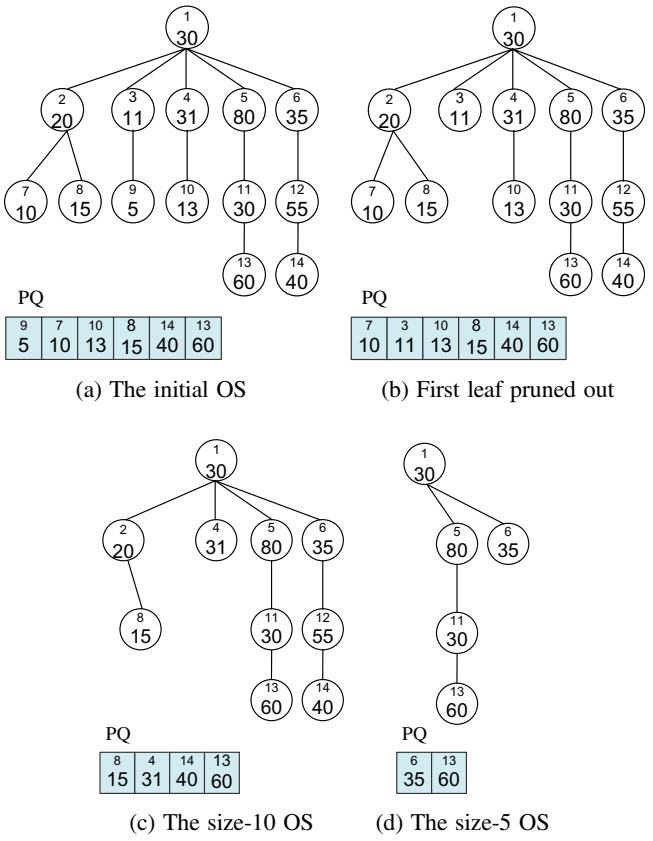
This algorithm, given an initial OS (either a complete or a prelim-l OS) iteratively prunes from the bottom of the tree the $n - l$ leaf nodes with the smallest $Im(\mathrm{OS}, t_i)$, where n is the number of nodes in the complete OS. The rationale is that since tuples need to be connected with the root and lower tuples on the tree are expected to have lower importance, we can start pruning from the bottom. A priority queue (PQ) organizes the current leaf nodes according to their local importance. Algorithm 2 shows a pseudocode of the algorithm and Fig. 3 illustrates the steps.

More precisely, this algorithm firstly generates the initial OS (line 1; e.g., the complete OS using Algorithm 1). The OS Generation algorithm generates the initial size-l OS and also the initial PQ (initially holding all leaves of the given OS). Then, the algorithm iteratively prunes the leaves with the smallest $Im(\mathrm{OS}, t_i)$. Whenever a new leaf is created (e.g., after pruning node 9 in Fig. 3, node 3 becomes a leaf), 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(log$n$). Thus, the overall cost of the algorithm is O($n$log$n$). This is much lower than the complexity of the DP algorithm, which gives the optimal solution.

On the other hand, this method will not always return the optimal solution; e.g., the optimal size-5 OS should include nodes 1, 5, 6, 12 and 14 instead of 1, 5, 6, 11 and 13 (Fig. 3(d)). In practice, it is very accurate (see our experimental results in Sec. 8.2), due to the aforementioned property of $Im(\mathrm{OS}, t_i)$, which gives higher probability to nodes closer to the root to have a high local importance.

Algorithm 2. The Bottom-Up Pruning Size-l Algorithm

Bottom-Up Pruning Size-l (l, t^{DS}, G^{DS})
1: **OS Generation**(t^{DS}, G^{DS}) {g}enerates initial size-l (i.e., complete or prelim-l) OS and initial PQ
2: **while** (|size-l OS| $> l$) **do**
3: t_{tem}=deQueue(PQ) ▷ the smallest value from PQ
4: **if!**(hasSiblings(size-l OS, t_{tem}))
5: enQueue(PQ, parent(size-l OS, t_{tem})) ▷ check whether after
6: pruning t_{tem}, its parent becomes a leaf node
7: prune t_{tem} from size-l OS
8: **return** size-l OS



(a) The initial OS (b) First leaf pruned out
(c) The size-10 OS (d) The size-5 OS

Fig. 3. The Bottom-Up Pruning Size-l Algorithm: Size-l OSs and their corresponding PQs (annotated with tuple ID and local importance).

4. DSize-*l* and PSize-*l* Snippets

We propose two types of size-l OSs, namely *diverse* DSize-l OSs and *proportional* PSize-l OSs, which extend the size-l OS definition[7] to capture diversity and proportionality, respectively. Similarly to size-l OSs, both DSize-l OSs and PSize-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^{DS}). We argue that an effective DSize-l (PSize-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 $dv(n_i), pq(n_i)$ and $li(n_i)$, respectively. $dv(n_i)(pq(n_i))$ and $li(n_i)$ are combined to a single score $dw(n_i)(pw(n_i))$ for a DSize-l (PSize-l) OS, simply denoted by $w(n_i)$ when the context (i.e., diversity or proportionality) is clear. The notations frequently used throughout this section and in the rest of the paper are summarized in Table 1.

Table 1. Notations for DSize-l and PSize-l OSs.

Notation	Definition
$gi(v_i)$	The global importance of a graph node v_i
$li(n_i)$	The local importance of an OS node n_i
$fr(v_i)$	The frequency a graph node v_i appears in an OS
$z(v_i)$	The amount of times that a graph node v_i has been added on the snippet
$dv(n_i)$	The diversity score of an OS node n_i
$dw(n_i)$	The diversity weight score: $li(n_i) * dv(n_i)$
$pq(n_i)$	The proportionality quotient of an OS node n_i
$pw(n_i)$	The proportional weight score: $li(n_i) * pq(n_i)$
$w(n_i)$	A weight score that may represent either $dw(n_i)$ or $pw(n_i)$ (when the context is clear)
$ap(n_i)$	The average $w(.)$ score of nodes of path n_i to root
$Im(Sl)$	The Importance score of a size-l Sl

Then, the objective is to select the l nodes that (1) include n^{DS} and form a connected subtree of the OS and (2) the sum of their $w(.)$ scores is maximized. Local importance (i.e., $li(.)$) can be calculated as in the original size-l OS problem (i.e., by multiplying affinity with global importance[7]), thus hereby we discuss only diversity and proportionality.

We investigate two relevance types among OS nodes, namely similarity (denoted as *sim*) and equality (denoted as *equi*). More precisely, we consider relevance among nodes belonging to the same relation; thus, we classify each G^{DS} relation either as *sim* or *equi*. We can use a domain expert to classify each relation in these types. When two nodes belong to different relations, then they have relevance 0 (e.g., the similarity between a conference and a paper is always 0). For nodes belonging to the same *sim* relation, we use Jaccard similarity (i.e., $sim(n_i, n_j) = \frac{|n_i \cap n_j|}{|n_i \cup n_j|}$; we treat n_i and n_j as sets of words). We can use an expert to define which attributes to compare per relation. For instance, Paper is a *sim* relation (Author G^{DS}, Fig. 2) and we define similarity between papers using their titles (e.g., "On Power-law relationships of the Internet Topology" versus "Power laws and the AS-Level Internet topology"). (Note that Jaccard similarity is symmetric (i.e., $sim(n_i, n_j) = sim(n_j, n_i)$) and the respective distance (i.e., $1 - sim(n_i, n_j)$) is a metric.) However, we observe that textual similarity cannot be applied on all relations meaningfully. For instance, consider Authors; it is not meaningful to define textual similarity between author names "Christos Faloutsos" versus "Michalis Faloutsos". Thus, for such cases (e.g., DBLP relations Author and ConfYear), we consider equality relevance, where two different OS nodes may either correspond to the same tuple (e.g., $g(n_i) = g(n_j) = v_p$) or to two different tuples (e.g., $g(n_i) \neq g(n_j)$). In the former case, we have $sim(n_i, n_j) = 1$, whereas in the latter case, we have $sim(n_i, n_j) = 0$. Note that in both relevance types, we measure relevance between two nodes using the same notation, i.e., $sim(n_i, n_j)$ (even for *equi* relations). Also note that $sim(n_i, n_j) = 1$ indicates that the two nodes are equal even in the case of *sim* relations; e.g., although Conference is a *sim* relation, an author may have papers appearing in the same conference more than once.

4.1. *Diversity (DSize-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 n_i in a DSize-l DSl, we suggest to estimate diversity as follows:

$$dv(n_i) = \begin{cases} 1 - \dfrac{\sum_{n_j \in \mathrm{DS}l, n_i \neq n_j} sim(n_i, n_j)}{l-1} & R(n_i) \textit{ is sim} \text{ relation} \\ 1 - \dfrac{z(g(n_i)) - 1}{l-1} & R(n_i) \textit{ is equi} \text{ relation} \end{cases}, \tag{4.1}$$

where $R(n_i)$ is the relation n_i belongs to, n_j is any other node in DSl and $sim(n_i, n_j)$ is the similarity between n_i and n_j. When $g(n_i) = g(n_j)$ (i.e., n_i and n_j correspond to the same tuple), then $sim(n_i, n_j) = 1$ for both *sim* and *equi* relations. For an *equi* relation, if $g(n_i) \neq g(n_j)$ then $sim(n_i, n_j) = 0$. Recall also that, if n_i and n_j do not belong to the same relations (i.e., $R(n_i) \neq R(n_j)$), then $sim(n_i, n_j) = 0$. The summation of similarities of n_i to the rest of the nodes in the snippet will give us the respective $dv(n_i)$ score. For an *equi* relation, that will be $z(g(n_i)) - 1$, where $z(g(n_i))$ is the amount of times $g(n_i)$ appears in the snippet (since for all nodes n_j such that $g(n_i) = g(n_j), sim(n_i, n_j) = 1$). Dividing by $l - 1$, we normalize $dv(n_i)$ in the range [0,1].

Given a set of nodes, $n_{j1}, \ldots, n_{jx}$, that have been added to the DSize-l OS, we denote as $dv(n_i|n_{j1}, \ldots, n_{jx})$ the diversity score of an unselected node n_i considering these added nodes. For instance, the score $dv(P_1|P_5)$ in Table 2 denotes the score of P_1 after the addition of P_5 in the snippet. For short, when the context is clear, we also denote as $dv(n_i|n_{jx})$ the score of n_i given that n_{jx} has been appended as the last (i.e., xth) node on the snippet. We also denote as $dv(n_i|\emptyset)$ the maximum diversity score a node n_i can get, i.e., $dv(n_i|\emptyset) = 1$; e.g., when n_i is the first to be added. This notation will be useful when describing our greedy algorithms; where after each node addition, the score of unselected nodes is affected accordingly.

For *equi* nodes for short, we also denote as $dv[z](g(n_i))$ the diversity score of a graph node n_i considering it appears for the zth time in the snippet. For instance, in Table 3, $dv[1]$ indicates the score of a node assuming it appears for the first time; where $dv[1](n_i) = 1$ is the maximum diversity score (which corresponds to $dv(n_i|\emptyset) = 1$). As another example, consider $l = 10$ and that C. Faloutsos appears two times (i.e.,

Table 2. *Sim* relevance of selected papers of Michalis Faloutsos.

Name	$li(.)$	dv $(.\|\emptyset)$	dw $(.\|\emptyset)$	dv $(.\|P_5)$	dw $(.\|P_5)$	dv $(.\|P_8)$	dw $(.\|P_8)$	pq $(.\|\emptyset)$	pw $(.\|\emptyset)$	pq $(.\|P_5)$	pw $(.\|P_5)$	pq $(.\|P_8)$	pw $(.\|P_8)$
P_1: Aggregated Multicast for Scalable QoS Mu..	0.17	1.00	0.17	1.00	0.17	0.99	0.16	0.69	0.11	0.69	0.11	0.59	0.10
P_2: Aggregated Multicast with Inter-Group Tr..	0.18	1.00	0.18	1.00	0.18	0.99	0.18	0.56	0.10	0.56	0.10	0.48	0.09
P_3: BGP-lens: Patterns and Anomalies in Inte..	0.16	1.00	0.16	0.99	0.16	0.98	0.16	0.58	0.09	0.49	0.08	0.43	0.07
P_4: Bounds for the On-line Multicast Problem..	0.19	1.00	0.19	1.00	0.19	0.99	0.19	0.68	0.13	0.68	0.13	0.59	0.11
P_5: On Power-law Relationships of the Intern..	0.61	1.00	0.61	—	—	—	—	1.15	0.71	—	—	—	—
P_6: Power Laws and the AS-Level Internet Top..	0.19	1.00	0.19	0.93	0.17	0.92	0.17	1.15	0.21	0.49	0.09	0.46	0.08
P_7: QoS-aware Multicast Routing for the Inte..	0.18	1.00	0.18	0.99	0.18	0.96	0.18	1.15	0.21	0.99	0.18	0.69	0.13
P_8: QoSMIC: Quality of Service Sensitive Mu..	0.31	1.00	0.31	0.99	0.31	—	—	0.93	0.29	0.79	0.24	—	—
P_9: Reducing Large Internet Topologies for F..	0.27	1.00	0.27	0.98	0.26	0.97	0.26	0.69	0.19	0.48	0.13	0.43	0.12
P_{10}: The Effect of Asymmetry on the On-Line..	0.17	1.00	0.17	1.00	0.17	0.99	0.16	0.86	0.14	0.86	0.14	0.74	0.12

Table 3. *Equi* relevance of selected co-authors of michalis (Ranked descending their $pw[1]$).

Name	$li(.)$	$fr(.)$	$dv[1](.)$	$dw[1](.)$	$dv[2](.)$	$dw[2](.)$	$pq[1](.)$	$pw[1](.)$	$pq[2](.)$	$pw[2](.)$
Srikanth V. Krishnamurthy	0.60	37	1.00	0.60	0.89	0.53	12.33	7.40	7.40	4.44
Christos Faloutsos	1.80	12	1.00	1.80	0.89	1.60	4.00	7.20	2.40	4.32
Jun-Hong Cui	0.81	11	1.00	0.81	0.89	0.72	3.67	2.97	2.20	1.78
Thomas Karagiannis	0.70	10	1.00	0.70	0.89	0.62	3.33	2.33	2.00	1.40
Michael Mitzenmacher	1.40	3	1.00	1.40	0.89	1.24	1.00	1.40	0.60	0.84
George Varghese	1.38	2	1.00	1.38	0.89	1.23	0.67	0.92	0.40	0.55
Konstantina Papagiannaki	0.61	4	1.00	0.61	0.89	0.54	1.33	0.81	0.80	0.49
Samuel Madden	1.61	1	1.00	1.61	0.89	1.43	0.33	0.54	0.20	0.32
Marek Chrobak	0.33	4	1.00	0.33	0.89	0.29	1.33	0.44	0.80	0.27
Jakob Eriksson	0.15	7	1.00	0.15	0.89	0.13	2.33	0.35	1.40	0.21

$z = 2$); $dv[2] = 1 - \frac{2-1}{10-1} = \frac{8}{9} = 0.89$ (Table 3). 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 alterative way would be to score the first occurrence as 1 and the second as 0.78, since $1 + 0.78 = 0.89 + 0.89$).

Our 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.[15] 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 measures such as Jaccard similarity. This is not the case here, since we do not consider the similarity of nodes to the query but a local importance score in relation to n^{DS}. Thus, local importance and 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 Sec. 4.3) we do not sum up local importance and dissimilarity, but multiply them.

4.2. *Proportionality (PSize-l OSs)*

We observe that in an OS we often find *equi* graph nodes (i.e., database tuples) multiple times. For instance, in the Michalis Faloutsos OS (see Table 3), 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_i in an OS as $fr(v_i)$ (or simply by fr when the context is clear). Graph nodes 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 $li(.)$), we should include nodes in proportion of their frequency. For example, if a graph node v_i appears 37 times in the total of 1,259 OS nodes, then v_i should ideally appear $\lfloor l \cdot 37/1,259 \rfloor$ 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.)

Analogously, we observe that the topic of *sim* nodes may appear multiple times; a node may be very similar to many

other nodes in the OS. For instance, in Table 2, we find 6 out of 10 papers including the word "Multicast" (e.g., P_1, P_2, P_4, etc.) and two papers including a pair of words "Aggregated Multicast". Thus, papers have some similarity due to the frequent common topics (e.g.,"Aggregated Multicast") and hence they should also be analogously represented.

For this purpose, for a given graph node n_i in a PSize-l PSl, we propose the use of the *proportional quotient* as follows:

$$pq(n_i) = \begin{cases} \dfrac{\sum_{n_j \in \text{OS}} sim(n_i, n_j)}{\alpha \cdot \sum_{n_j \in PSl} sim(n_i, n_j) + 1} & R(n_i) \textit{ is sim} \text{ relation} \\ \dfrac{fr(g(n_i))}{\alpha \cdot z(g(n_i)) + 1} & R(n_i) \textit{ is equi} \text{ relation} \end{cases}, \tag{4.2}$$

where $R(n_i)$ is the relation where n_i belongs, $sim(n_i, n_j)$ is the similarity between the two nodes (as defined in Eq. (4.1)) and α is a constant that can tune proportionality. For *equi* relations, $z(g(n_i))$ is the amount of times that node n_i appears in the snippet and $fr(g(n_i))$ is the frequency that the node appears in the OS. We use analogous notations as in $dv(n_i)$. We denote by $pq(n_i|n_{j1}, \ldots, n_{jx})$ the proportional quotient of n_i when nodes $n_{j1}, \ldots, n_{jx}$ have been appended to the snippet. For *equi* nodes, we also denote as $pq[z](n_i)$ the proportional score considering n_i appears z times (Table 3).

This equation is inspired by the Sainte-Laguë Algorithm 4[16] (with $\alpha = 2$) and empirically we found that it is very effective for our problem (other equations can also be considered, e.g., Refs. 17 and 18). The rationale of this quotient is to favor a frequent node (or nodes including frequent topical words) and each time a node is added to the snippet its proportional score is significantly decayed so that other frequent nodes 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 once we get $pq[1](n_i) = 12/3 = 4$ and twice we get $pq[2](n_i) = 12/5 = 2.4$.

4.3. *DSize-l and PSize-l definitions*

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

$$dw(n_i) = li(n_i) \cdot dv(n_i), \tag{4.3}$$

where $li(n_i) = af(n_i) \cdot gi(n_i)$ is the local importance of n_i and $dv(n_i)$ is the diversity factor (Eq. (4.1)). Tables 2 and 3 depict 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 five authors (and thus an intermediary paper), then we will select twice C. Faloutsos (i.e., $0.89 \cdot 1.8 + 0.89 \cdot 1.8 = 1 \cdot 1.8 + 0.78 \cdot 1.8$) and once S. Madden ($1 \cdot 1.6$), M. Mitzenmacher ($1 \cdot 1.4$) and G.Varghese ($1 \cdot 1.4$). Note that a third addition of C. Faloutsos cannot compete the total 1.4 score, as the additional score is only 1.12 (i.e., $(3 \cdot 0.78 - 1 - 0.78) \cdot 1.8 = 0.56 \cdot 1.8 = 1.08$).

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 $l \leq |\text{OS}|$)
(2) The l nodes form a connected tree rooted at n^{DS}
(3) Each node n_i carries a weight $dw(n_i)$ (obtained by Eq. (4.3))
(4) The aggregated score of a DSize-l OS DSl can be calculated by:

$$Im(\text{DS}l) = \sum_{n_i \in \text{DS}l} dw(n_i). \tag{4.4}$$

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 $Im(\text{DS}l)$ among all such candidates.

Find an optimal DSize-*l* OS. Given an OS and l, find a candidate DSize-l OS of maximum score (according to Definition 1). Analogously, we define the proportionality score per node (i.e., $pw(n_i) = li(n_i) \cdot pq(n_i)$, instead of Eq. (4.3)), 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. Krishnamurthyan 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.

Problem Definitions and Algorithms. We have two types of problems, namely PSize-l and DSize-l generation. In addition, an OS may have only *equi* relevance or both *equi* and *sim* relevance. Thus, we have four combinations of problems and thus we propose algorithms that are general to address all these combinations. Firstly, we propose a brute-force algorithm which is prohibitively slow. Then, we propose two greedy algorithms LASP and 2-LASP*e* (which is LASP's optimization). Finally, we propose pruning algorithms that can produce pruned preliminary results. We can apply all aforementioned algorithms on these preliminary OSs.

Notation. For simplicity, we unify $dw(.)$ and $pw(.)$ into a single notation $w(.)$ and use $w(n_i)$ to refer to the corresponding diversity or proportionality score of a node n_i in a DSize-l OS or PSize-l OS, respectively. Analogously to diversification and proportionality scores notations, we denote $w(n_i|n_{j1}, \ldots, n_{jx})$ as the score given $n_{j1}, \ldots, n_{jx}$ have been added and $w[z](n_i)$ as the score when n_i is added for the zth time. In the rest of the paper, whenever the context is clear, we drop n_i or $(n_i|n_{j1}, \ldots, n_{jx})$ from the notation and denote the diversity/proportionality score of a node simply by $w(.)$.

5. Largest Averaged Score Path

The BF-l algorithm can be very expensive even for moderate values of l or $|\text{OS}|$. Thus, we propose Largest averaged score

Algorithm 3. The Largest Averaged Path Algorithm

LASP (l, n^{DS})

1: **OS Generation** (n^{DS}){g}enerates OS, $w(.)$, $ap(.)$, HFr and W
2: **while** (|size-l| $< l$) **do**
3: p_i=path from maximum W node
4: add first ($l-$|size-l|) nodes of p_i to size-l
5: **if**(|size-l| $< l$) **then**
6: remove selected path p_i from the OS tree and from W
7: **UpdateRemPaths** (p_i)
8: **UpdateRelScores** (p_i)
9: **return** size-l

UpdateRemPaths (p_i)

1: **for each** child v of nodes in p_i **do**
2: **for each** node n_j in the subtree rooted at v **do**
3: update $ap(n_j)$ on the OS tree
4: update $ap(n_j)$ on W

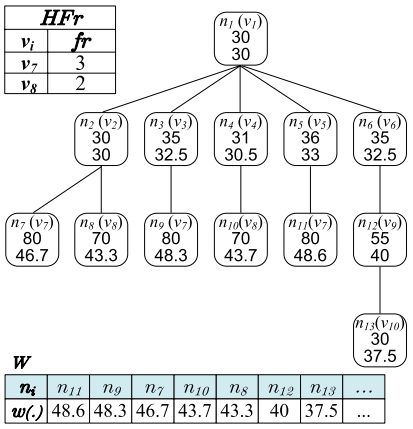
UpdateRelScores (p_i)

1: **for each** node n_i in p_i **do**
2: **for each** unselected OS node n_j **do**
3: **if**($sim(n_j, n_i) > 0$) **then**
4: update $w(n_j)$ considering $sim(n_j, n_i)$
5: **for each** node n_k in the subtree rooted at n_j **do**
6: update $ap(n_k)$ on OS tree using $w(n_j)$
7: update $ap(n_k)$ on W

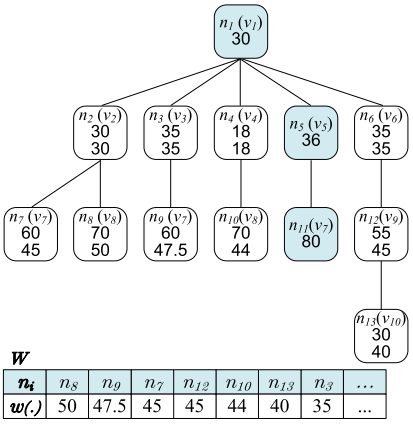
path (LASP), a greedy algorithm, that can produce a size-l OS of high quality at a much lower cost. In a nutshell, LASP firstly generates the OS. It also calculates for each node n_i an initial $w(n_i)$ score (i.e., $w(n_i|\emptyset)$, using Eq. (4.3)) and its corresponding *average* $w(n_i)$ *score per node* (denoted as $ap(n_i)$) of the path from n_i to the root. Then, the algorithm iteratively selects and adds to the size-l OS the path p_i 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. Algorithm 3 is a pseudocode of the heuristic and Fig. 4 illustrates an example.

LASP is a general algorithm that (1) can compute both types of size-l OSs (i.e., DSize-l and PSize-l OSs) and (2) can process both relevance types (i.e., equality and similarity). The difference between the two size-l types is that the proportionality equation also considers the similarity/equality (frequency) of each node against all other nodes, which is calculated during the OS generation process (to be described in more detail shortly). LASP can process both relevance types by using the pre-calculated *sim matrix* (a matrix storing the similarity among all nodes), which facilitates the initial calculation of $w(.)$ and $ap(.)$ and the consequent updates (to be described in more detail shortly). Thus, given an OS annotated with $w(.)$ and $ap(.)$ scores, the problem of determining either DSize-l or PSize-l using either equality or similarity relevance type remains the same for LASP.

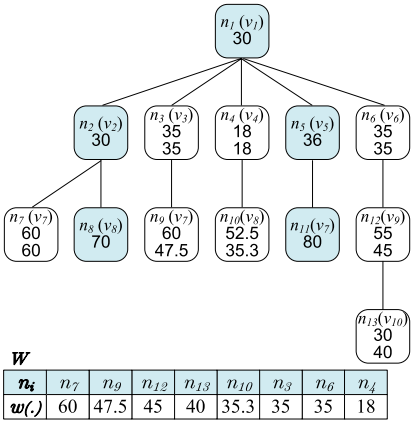
More specifically, the LASP algorithm firstly generates the OS (line 1). During **OS Generation()**, LASP also calculates the $w(.)$ score and the respective $ap(.)$ score per node.



(a) The initial OS



(b) First update



(c) The final update

Fig. 4. The LASP algorithm: The Size-5 OS (annotated with OS and (graph) node ID, $w(.)$ and $ap(.)$; selected nodes are shaded).

For the DSize-l, the calculation of $dv(.)$ (and thus $w(.)$) is straightforward; whereas for the PSize-l, the calculation of $pq(.)$ is more demanding, as it requires the comparison of each node against all other nodes. Thus, for *equi* relations, in order to facilitate faster calculation of $pq(.)$ scores, we also

maintain a hash table of graph nodes (denoted as *HFr*) containing the frequency of a graph node in the OS tree (denoted as *fr*). *HFr* can easily be compressed by excluding nodes appearing only once in the OS; thus, if a node does not exist in *HFr*, we can infer that it only appears once. For *sim* relations, we compare each node against all other nodes as to obtain their $pq(.)$. Note, that *sim* comparisons are more expensive (i.e., $\frac{(n+1)\cdot n}{2}$ time) in contrast to *equi* *HFr*-based comparisons that require only n time. During the OS generation, we also generate a priority queue W of the initial $ap(.)$, in order to better manage nodes.

We then select the node with the largest $ap(.)$ and add the corresponding path to the size-l OS. We remove this p_i from the OS and from W (lines 6). By removing the nodes of p_i 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 (**UpdateRemPaths()**) (2) and also to consider the revised $w(.)$s due to relevance to newly added nodes (**UpdateRelScores()**). Note that the $ap(.)$ of an unselected node corresponds to the $w(.)$ score, the node will have if included in the snippet (considering also the similarity loss of already added nodes for the diversity case, i.e., *dv*). Thus, $w(.)$ and $ap(.)$ scores of all unselected nodes should be updated each time a new node is added (by function **UpdateRelScore()**). Also note that this general LASP can be significantly more expensive than the version of LASP presented in Ref. 19, since for the latter it suffices to simply count *fr* and z of added nodes for the estimation of $ap(.)$ and $w(.)$ scores.

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 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.

Take for instance the example of Fig. 4, where nodes at level one have similarity relevance and nodes at level two have equality relevance. More precisely, consider $sim(n_3, n_4) = sim(n_4, n_5) = 0.6$; whereas equality relevance is annotated on the example of figure (e.g., $g(n_7) = g(n_9) = g(n_{11}) = v_7$). Node n_{11} (i.e., graph node v_7) has $ap(n_{11})$=48.6, because its path includes nodes n_1, n_5 and n_{11} with average $w(.)$ 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, $w(.)$ and $ap(.)$ are updated to disregard the removed nodes and also to consider the inclusion of newly added nodes (Fig. 4(b)). For instance, the revised $ap(n_{12})$ is $(35 + 55)/2 = 45$, because its path now includes only n_6 and n_{12}. Also, nodes n_7 and n_9 which correspond to the same graph node as n_{11} and node n_4 which has similarity with node n_5 which have just been added to the size-l also need to be updated with new $w(.)$ and $ap(.)$ scores. 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_8, which adds two more nodes to the size-l OS (Fig. 4(c)). Note that $ap(n_i)$ for each node n_i corresponds to the path starting from n_i to the root of the corresponding unselected tree (from the unselected forest). For instance, during the second update, p_8 comprises n_2 and n_8. Note also that the path's root (e.g., n_2) 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.

6. 2-LASP*e*

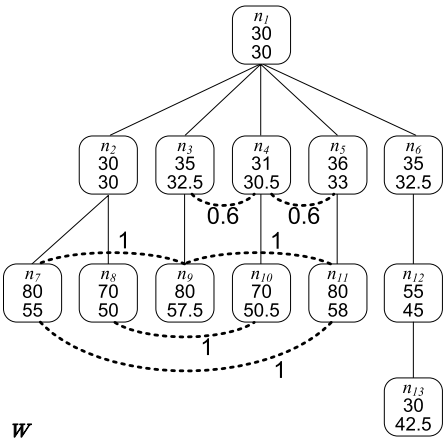
The runtime cost of LASP is dominated by the numerous updates it applies; each time we add a node (or path) to the snippet, we have to update up to twice each of the remaining nodes. Thus, we introduce the 2-LASP*e* algorithm, an enhancement of LASP, that aims to reduce where possible such updates. In a nutshell, 2-LASP*e* facilitates update reductions at both **UpdateRemPaths()** and **UpdateRelScores()** phases. The algorithm remains, like LASP, general and can address both types of size-l and relevance. Algorithm 4 illustrates the differences of the two functions from the respective original functions of the LASP algorithm; whereas Fig. 5 illustrates an example.

We propose to relax LASP by averaging $w(.)$ pairs of nodes (hence the prefix 2 to the name of the algorithm). 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. 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. Recall that, $li(.) = af(.) \cdot gi(.)$ where $af(.)$ is monotonic by definition and

Algorithm 4. 2-LASPe Algorithm

2-***LASP****e* (l, n^{DS}) **UpdateRemPaths** (p_i)
1: **for each** child n_j of nodes in p_i **do**
2: update $ap(n_j)$ on the OS tree
3: update $ap(n_j)$ on W

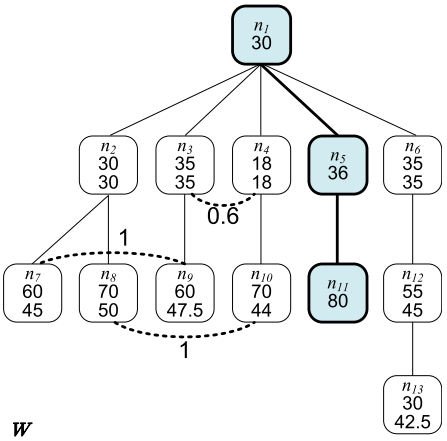
UpdateRelScores (p_i)
1: **for each** node n_i in p_i **do**
2: **for each** similarity edge $se(n_i, n_j)$ (of n_i with an unselected n_j node) **do**
3: update $w(n_j)$ considering $sim(n_j, n_i)$
4: deleteEdge($se(n_i, n_j)$)
5: **for each** child n_k of n_j **do**
6: update $ap(n_k)$ on OS tree using $w(n_j)$
7: update $ap(n_k)$ on W



W

n_i	n_{11}	n_9	n_7	n_{10}	n_8	n_{12}	n_{13}	...
$w(.)$	58	57.5	55	50.5	50	45	42.5	...

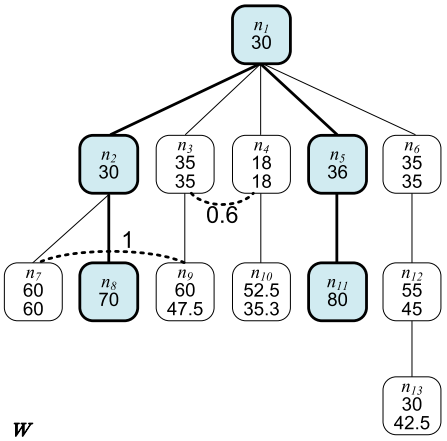
(a) The initial OS (and *sim*OS)



W

n_i	n_8	n_9	n_7	n_{12}	n_{10}	n_{13}	n_3	...
$w(.)$	50	47.5	45	45	44	42.5	35	...

(b) First update



W

n_i	n_7	n_9	n_{12}	n_{13}	n_{10}	n_3	n_6	n_4
$w(.)$	60	47.5	45	42.5	35.3	35	35	18

(c) Final update

Fig. 5. 2-LASP*e* algorithm (annotated with similarity edges of unselected nodes).

additionally both $dv(.)$ and $pq(.)$ equations are monotonic to z (and additions of similar nodes), where z is expected to be larger at the bottom levels of the OS tree.

Secondly, we introduce a similarity index on the OS tree, denoted as *sim*OS tree (see Fig. 5) (instead of using the *HFr* of LASP and 2-LASP[19]). This *sim*OS tree is generated during the **OS Generation()** function (line 1). For each pair of nodes that there exists a similarity, we add a similarity edge carrying the similarity value, denoted as $se(n_i, n_j)$. Using the similarity edges, we can limit checks of newly added nodes against only unselected nodes that we know they have a similarity (thus the suffix *e* for edge to the name of the algorithm). This is in contrast to LASP which checks against all unselected nodes.

Let us first demonstrate the updates due to removals of paths. At each addition, we update only pairs of scores, i.e., $ap(.)$ between the affected node and its parent (instead of all remaining paths towards the root). For example, after adding path p_{11}, in 2-LASP*e*, we only need to update nodes at level 1, except the included n_5 node (since node n_1 is removed). Let us now demonstrate the updates due to relevance between the selected path and unselected nodes. We can easily determine from the *sim*OS tree that p_{11} path's nodes have similarity with n_4, n_7 and n_9 nodes and thus update them according to their similarities. Since at each iteration, we only need to check the similarity between a newly selected node against all unselected nodes, we delete all similarity edges of a newly selected node (line 4). Thus, after the first update and removal of p_{11} path, similarity edges of n_5 and n_{11} nodes are deleted from the *sim*OS tree.

7. Prelim-*l* Algorithms

The aforementioned algorithms operate on the complete OS. Inspired by the prelim-*l* approach,[7] we propose to produce a subset of the OS, denoted as DPrelim-*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 Ref. 7 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 VPrelim-*l*. The direct application of VPrelim-*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 DSize; (2) it fails to manage the nonmonotonic relationship between $w(n_i|.)$ and $li(n_i)$ of proportionality (e.g., $w(n_i|\emptyset) > li(n_i)$), which requires the challenging estimation of similarity among nodes (and frequency per node) in the OS; and (3) it does not facilitate further pruning of nodes that have similarity with already added nodes (or 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 similarity between nodes (and frequency of nodes) in an OS in order to produce $w(.)$ scores. Thus, we first present more comprehensively a generalized version of the VPrelim-*l* approach, which addresses proportionality. This algorithm, denoted as PPrelim-*l*, considers the similarity/frequency and respective upper bounds of nodes in an OS, in order to produce the so-called PPrelim-*l* OS. Then, we present more synoptically the DPrelim-*l* algorithm with the required specializations and simplifications in order to produce DPrelim-*l* OS.

7.1. *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* diverse graph nodes (i.e., nodes that their similarity is less than a threshold $d(\theta)$) with the largest $w(n_i|\emptyset)$ scores (denoted as the $top_w l$ set).

The rationale is that while searching for $top_w l$ 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 diverse nodes in $top_w l$ is necessary in order to facilitate eventual diversity.

We generate the PPrelim-*l* OS by extending the complete OS generation algorithm (Algorithm 1, described in Sec. 2 and in Ref. 4) to include three pruning conditions. We traverse the G^{DS} graph in a breadth first-order, according to Algorithm 5. For this purpose, cheap pre-computed indexes, variables and data structures are employed. Hereby, we describe the algorithm by introducing the (1) pre-computed indexes per relation and n^{DS}, (2) variables and data structures and (3) the pruning conditions that are used as to construct the algorithm. We try to describe these terms, where possible, in the order they appear in the algorithm. Figures 6 and 7 illustrate an example and Table 4 summarizes the notation we are using.

The calculation of *sim* relevance requires more time (i.e., quadratic; as we have to compare each node against all remaining nodes); in comparison to *equi* relevance which simply requires counting the frequency of graph nodes. Thus, we address the two relevance types separately. For instance, for *sim* relations, we rely mainly our pruning on pre-computed indexes (e.g., $mw(n^{\mathrm{DS}}, R_i)$ to be described shortly); whereas for *equi* relations, we achieve further pruning by using online retrieved information (e.g., $cmFr(R_i)$ to be described shortly).

Index per G^{DS} Relation. Our PPrelim-*l* OS generation technique uses three indexes (i.e., bounds) for each G^{DS} relation R_i, which are pre-computed. $max(R_i)$ is the maximum value of $li(.)$ in R_i. $mmax(R_i)$ is 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 G^{DS} leaf node). Finally, UBFr(R_i) is the upper bound of joins a node in R_i can have with any n^{DS}. 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_{Paper}) can only join with n^{DS}. Otherwise, we assume infinite joins (e.g., the same co-author may appear in an unbounded number of papers) and set UBFr(Co-author) $= \infty$.

Algorithm 5. The PPrelim-*l* OS Generation Algorithm

PPrelim-*l* (l, G^{DS})

1: $t = 0$; $n_j = n^{\mathrm{DS}}$; $W_l = \{\}$; $Q = \{\}$
2: **addNode**(n_j)
3: **while** !IsEmptyQueue(Q) **do**
4: $Xn_j = n_j$; n_j = deQueue(Q)
5: **for each** child relation R_i of $R(n_j)$ in G^{DS} **do**
6: **if** (UBFr(R_i)>1) **then**
7: **if** ($R(n_j) \neq R(Xn_j)$) **then** {$n_j \in$ new relation}
8: cmFr(R_i) = 0; $i = c(RPar_i)$
9: UBFr(n_j, R_i)=min{$--i$+cmFr(R_i), $mFR(n^{\mathrm{DS}})$}
10: **else**
11: UBFr(n_j, R_i) = 1
12: UBw(n_j, R_i)= min{$mw(n^{\mathrm{DS}}, R_i)$, f_1(min{$max(R_i)$, $max(n^{\mathrm{DS}})$}, min{($mFr(n^{\mathrm{DS}})$, UBFr(R_i), UBFr(n_j, Ri)})}
13: dUBw(n_j, R_i)= min{$mmw(n^{\mathrm{DS}}, R_1)$, f_2(min{$mmax(R_i)$, $max(n^{\mathrm{DS}})$}, min{($mFr(n^{\mathrm{DS}})$, dUBFr(R_i)})}
14: **if** !($t \geq$UBw(n_j, R_i) && ($t \geq$dUBw(n_j, R_i))) **then** {Prun. Cond.1}
15: **if** ($t \geq$dUBw(n_j, R_i)) **then** {Prun. Cond.2}
16: $R_i(n_j)$: get up to diverse l nodes with $UBw(.) > t$ where $UBw(.) = f_3(.)$
17: **else**
18: get $R_i(n_j)$
19: **for each** tuple n_i of $R_i(n_j)$ **do**
20: **if** R_i is *equi* relation **then**
21: **if** (UBFr(R_i)> 1) **then**
22: UpdateHFr(n_i)
23: UpdatecmFr(n_i)
24: $w(n_i)[1]$=$f(li(n_i)$, HFr[$g(n_i)$].fr, 1){E}q. 4.3
25: **if** !((HFr[$g(n_i)$].fr>1) && ($w(n_i)[2]<t$) && ($t \geq$ dUBw(n_j, R_i))) **then** {Prun. Cond.3}
26: **addNode**(n_i)
27: **else if** (R_i is sim relation)
28: addOnHSi(n_i)
29: **for each** n_k in HSi **do**
30: **if** $sim(n_i, n_k)$ **then**
31: Update ($w(n_i|\varnothing)$, $sim(n_i, n_k)$)
32: Update ($w(n_k|\varnothing)$, $sim(n_k, n_i)$)
33: $cSim(n_i)$ =**true**
34: $w(n_i)[2] = f(w(n_i|\varnothing), 2)$){E}q. 4.3
35: **if** (!($cSim(n_i)$ && ($w(n_i)[2] < t$) && ($t \geq$ dUBw(n_j, R_i)))) **then** {Prun. Cond.3}
36: **addNode**(n_i)
37: **return** PPrelim-*l*

addNode (n_i)

1: EnQueue (Q, n_i)
2: add n_i on PPrelim-*l* as child of n_j or as root if n_i is n^{DS}
3: **if** $(w(n_i)[1] > t)$ **then**
4: **if** n_i is not similar with any nodes in W_l **then**
5: EnQueue (W_l, n_i)
6: **if** $(|W_l| > l)$ **then**
7: DeQueue(W_l)
8: **if** $((|W_l| < l))$ **then**
9: $t = 0$
10: **else**
11: t =minimum(W_l)

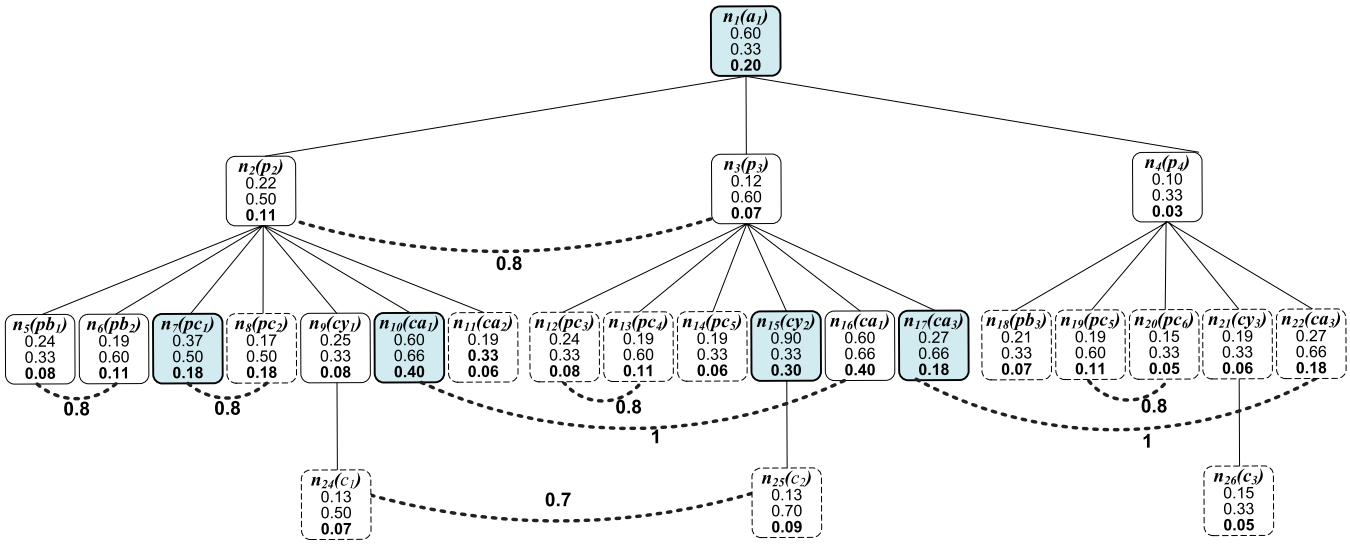


Fig. 6. PPrelim-l Example for $l = 5$ (The complete OS, the PPrelim-l OS and the $top_w l$ set). Edges between nodes indicate their similarity. Nodes are annotated with their $li(.), pq(.)$ and $w(.)$. Nodes with low transparency are pruned nodes (e.g., n_8, n_{11}, etc.), shaded nodes are the $top_w l$ set (e.g., n_1, n_7, etc.) and the rest are the remaining tuples of the PPrelim-l OS. (e.g., n_2, n_3, etc.)

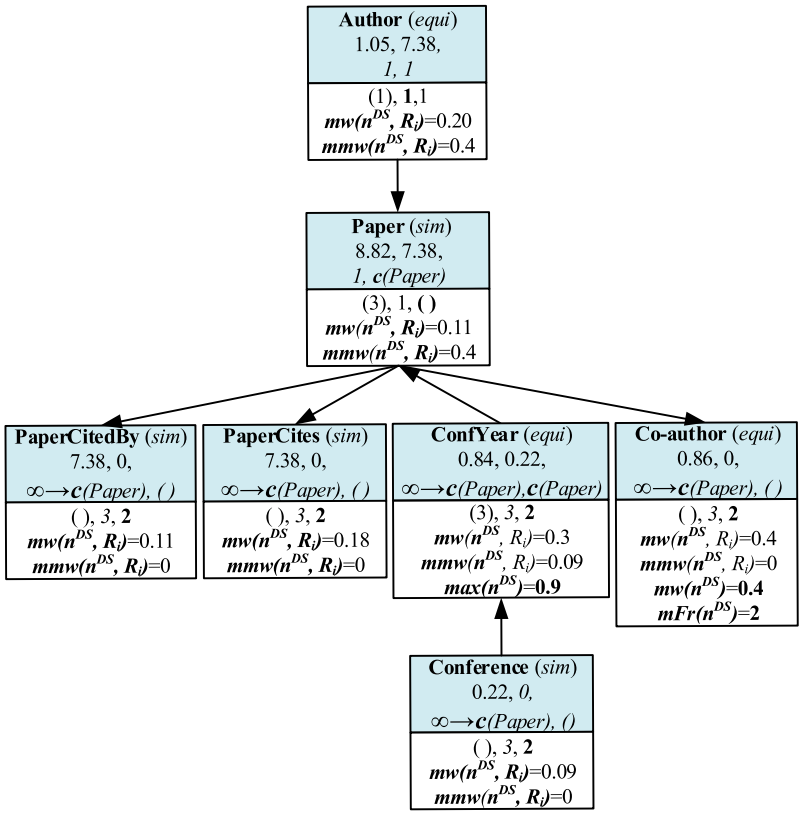


Fig. 7. The DBLP Author G^{DS}; annotated with relation indexes: (relevance type), $max(R_i)$, $mmax(R_i)$, UBFr(R_i), $c(R_i)$, and n^{DS} indexes for the example of Fig. 6: ($c(R_i)$), UBFr(R_i), $mFr(n^{DS}), mw(n^{DS}, R_i)$, etc.)

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 can join with n^{DS}. Thus, for N:1 relationships, $c(R_i) = c(RPar_i)$, where $c(R^{\mathrm{DS}}) = 1$ and $RPar_i$ is the parent relation of R_i. Thus, given $c(RPar_i)$ for cases where UBFr (Co-author) $= \infty$, we estimate UBFr(R_i) as a function of $c(RPar_i)$, i.e., UBFr$(R_i) = c(RPar_i)$ (denoted as $\infty \rightarrow c(R_i)$ in Fig. 7); this association will be useful later, during the

Table 4. Notations of the PPrelim-l Algorithm.

Notation	Definition
$top_w l$ set	The l diverse graph nodes with the largest w scores
$max(R_i)$	The maximum value of $li(.)$ in R_i
$mmax(R_i)$	The maximum $max(R_i)$ in all R_i's descendentant nodes or 0 if R_i has no descendants
UBFr(R_i)	The upper bound of joins a node in R_i can have with any n^{DS}
$mw(n^{\mathrm{DS}}, R_i)$	The maximum w score of R_i nodes in the n^{DS} OS
$mmw(n^{\mathrm{DS}}, R_i)$	The maximum $mw(n^{DS}, R_i)$ of all R_i's descendents or 0 if R_i has no descendants
$mw(n^{\mathrm{DS}})$	The maximum w score of nodes in the n^{DS} OS
$mFr(n^{\mathrm{DS}})$	The maximum frequency of any node in the OS
$R(n_i)$	The relation n_i belongs to
$R_i(n_j)$	The subset of R_i that joins with n_j
$c(R_i)$	The summation of nodes in R_i that can join n^{DS}
UBw(n_j, R_i)	The upper bound of w in $R_i(n_j)$
dUBw(n_j, R_i)	The upper bound of w from all R_i's descendentants that join with n_j

online calculation of tighter frequency bounds. Also note that since, we only need $c(RPar_i)$, we do not need to calculate $c(R_i)$ for leaf nodes (thus we denote their $c(.)$ as () in Fig. 7). Finally, we define and use index dUBFr(R_i), which is the upper bound of joins of a node belonging to any descendant relation of R_i that can have with any n^{DS}.

Index per n^{DS} node. During pre-processing, we also maintain a number of indexes per n^{DS}. $mw(n^{\mathrm{DS}}, R_i)$ is the maximum $w(n_i|\emptyset)$ of any node in R_i (note that this can be prohibitively expensive to calculate online and can render the pruning ineffective). For instance, in our running example for Paper, we have $mw(n^{\mathrm{DS}}, R_i) = 0.11$. $mmw(n^{\mathrm{DS}}, R_i)$ is the maximum value of $mw(n^{\mathrm{DS}}, R_i)$ of all R_i's descendants or 0 if R_i has no descendants (leaf node) (it can cheaply be obtained from descendants' $mw(n^{\mathrm{DS}}, R_i)$. For example, for the Paper relation, $mmw(n^{\mathrm{DS}}, R_i) = 0.4$ due to $mw(n^{\mathrm{DS}}, R_i) = 0.4$ of the Co-Author relation. $max(n^{\mathrm{DS}})$ is the maximum $li(.)$ for all nodes in an OS (excluding n^{DS}); e.g., in Fig. 7, $max(n^{\mathrm{DS}}) = 0.9$ is found in the ConfYear relation. Note that this score overrides the maximum $li(.)$ score of all G^{DS} relations (i.e., $max(R_i)$ and $mmax(R_i)$). $mw(n^{\mathrm{DS}})$ is the maximum $w(n_i|\emptyset)$ of any node in the OS (excluding n^{DS}). Similarly to $max(n^{\mathrm{DS}})$, this score is considered as the upper bound of $w(.)$ of all nodes of all relations (e.g., in our running example, $mw(n^{\mathrm{DS}}) = 0.4$ is found in relation Co-Author). $mFr(n^{\mathrm{DS}})$ is the maximum frequency of any node in the OS that belongs to a relation with UBFr(R_i) > 1, where $mFR(n^{\mathrm{DS}}) \leq$ UBFr(R_i). For instance, in our example, $mFr(n^{\mathrm{DS}}) = 2$ (since ca_1 and ca_3 appear twice); which is less than UBFr(R_i) $= 3$, 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 $R(n_i)$ be the relation where to n_i belongs. While processing n_j (in $R(n_j)$) against a relation R_i with UBFr(R_i) > 1, we try to get a tighter bound than UBFr (R_i) and $mFr(n^{\mathrm{DS}})$, 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, consider we are processing node $n_4(p_4)$ against the Co-Author relation, node ca_1 is the most frequent among all $R_i(n_1), \ldots, R_i(n_{j-1})$ sets that was found so far since it was found twice; thus $cmFr(R_i) = 2$. Given $cmFr(R_i)$, UBFr (n_j, R_i) assumes that ca_1 will appear in all the remaining sets $R_i(n_j), \ldots, R_i(n_{|R_i|})$ after processing n_j. At the beginning, UBFr(n_j, R_i) can be very loose, so we compare it with $mFr(n^{\mathrm{DS}})$, to keep the minimum of the two (lines 6–11).

Another bound, we use is UBw(n_j, R_i), which is the upper bound of the $w(n_i|\emptyset)$ score that can be obtained from $R_i(n_j)$ (line 12) (this value will be useful as to facilitate Pruning Condition 1). It is defined as the minimum of $f_1(.)$ and $mw(n^{\mathrm{DS}}, R_i)$. The $mw(n^{\mathrm{DS}}, R_i)$ index can be a very effective pruning tool for both relevance types. $f_1(.)$ aims to facilitate further pruning for *equi* relations thus is defined as follows: for *sim* relations is set to ∞ (as we do not expect to achieve a better bound than $mw(n^{\mathrm{DS}}, R_i)$ that can be practically useful); whereas for *equi* relations is calculated using Eq. (4.3) for $z = 1$.

We denote as dUB(n_j, R_i) the upper bound of $w(n_i|\emptyset)$ of all nodes from R_i's descendant relations that can join with n_j or 0 if R_i has no descendants (and it will be useful in facilitating Pruning Condition 2). Similarly to UBw(n_j, R_i) calculation, dUBw(n_j, R_i) can be defined as the minimum of $mmw(n^{\mathrm{DS}}, R_i)$ and $f_2(.)$. $f_2(.)$ is also defined by Eq. (4.3) for $z = 1$ (line 13). The $mmw(n^{\mathrm{DS}}, R_i)$ index can be a very effective pruning tool for both relevance types. Analogously, $f_2(.)$ aims to facilitate further pruning for *equi* relations thus is defined as follows: for *sim* relations is set to ∞; whereas for *equi* relations is calculated using the given parameters. Also note that, if R_i is a leaf node on G^{DS} then $mmax(R_i)$=0 and thus dUBw(n_i, 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[7] in Pruning Conditions 1 and 2, respectively; however, they are tighter bounds as they are specific for the given n^{DS}.

Finally, we define the upper bound of $w(n_i|\emptyset)$ score of a node as $UBw(n_i)$ (which will be useful during Pruning Condition 2; line 16). We calculate $UBw(n_i)$ using the respective $pq(n_i|\emptyset)$ produced by function f_3 which is defined as follows. For an R_i being an *equi* relation, $pq(n_i)[1] = f_3(li(n_i),$ UBFr(n_j, R_i)). Whereas, for an R_i being a *sim* relation, $pq(n_i|\emptyset) = f_3(((|\mathrm{OS}($ $R_i)|-$ UBFr(n_j, R_i)) $\cdot\, msim(n_i)+$ UBFr(n_j, R_i) $\cdot\, 1) \cdot li(n_i))$; where $|\mathrm{OS}(R_i)|$ is for UBFr(R_i) $= 1$ the amount of nodes of R_i in the OS and for UBFr(R_i) > 1 $c(RPar_i)$. $msim(n_i)$ is the maximum similarity of an n_i node with any other node in the OS. Namely, we assume that n_i appears UBFr(n_j, R_i) times (thus UBFr(n_j, R_i) $\cdot\, 1$ similarity) and has the maximum similarity with all OS(R_i) nodes.

As we have already explained, we process *sim* and *equi* relations separately. Thus, while processing *equi* relations we maintain *HFr*, a hash table, which indexes for each graph

node the computed frequency in the OS so far (lines 22 and 23). And while processing *sim* relations, we maintain *HSi* hash table which indexes the similarity of each OS node against all other OS nodes (i.e., $\sum_{n_j \in \text{OS}} sim(n_i, n_j)$). The update of *HSi* requires quadratic time as we need to compare the similarity of each node against all previous nodes already on *HSi* (lines 29–32). We also use the $cSim(n_i)$ flag variable to indicate whether n_i is similar to other nodes (line 33). We also denote as $w(n_i)[2]$, the $w(n_i)$ score given another node with maximum similarity with n_i has previously been added (lines 16 and 34); maximum similarity is the equality similarity (thus the common use of the *equi* notation).

We manage the retrieval of $top_w l$ set as follows. Let t be the current smallest value of the $top_w l$ set (or 0 if $top_w l$ does not contain l values yet). If the current tuple n_i is greater than t (line 3, function AddNode) and if n_i is diverse to all $top_w l$ nodes then is added to the PPrelim-l and the l-sized priority queue W_l which manages the $top_w l$ set (AddNode(), lines 4–11). For instance, in Fig. 6 the shaded nodes comprise the final $top_w l$ set for the given OS. Note also that although node n_{16} has score larger than $t = 0.18$, it is excluded as it is similar (equal) with node n_{10}. Note that by considering the computed similarity/frequency of a node n_i so far, which is less than or equal to actual similarity/frequency of n_i, in fact we consider the lower bound of $w(.)$.

Pruning Conditions. Each time we further process a node n_j, we employ three pruning conditions:

- *Pruning Condition* 1. If t is greater than or equal to the $w(n_i|\emptyset)$ of all tuples of the current relation R_i and all its descendants (i.e., $t > \text{UBw}(n_j, R_i)$ and $t > \text{dUBw}(n_j, R_i)$), then there is no need to traverse the sub-tree starting at R_i (line 14).
- *Pruning Condition* 2. We can limit the amount of tuples returned by an $R_i(n_j)$ join (i.e., by avoiding 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 $top_w l$ (i.e., when $t > \text{dUB}(n_j, R_i)$; line 15). Then, we can extract only nodes that their upper bound $w(n_i|\emptyset)$ score, $UBw(n_i)$, is greater than t (line 16).
- *Pruning Condition* 3. When Pruning Condition 2 holds, we can safely extract only part of the join. However, it is still possible that we extracted nodes which are similar to already added on the PPrelim-l nodes; and thus their $w(n_i|n_{j1}, \ldots, n_{jx})$ will be actually used. Thus, we introduce a new pruning condition that checks first if a node n_i is similar to an added node and then consider adding it. For *equi* relations, we can easily detect equality by accessing *HFr*. Whereas for *sim* relations this is more demanding (requiring quadratic time; lines 29–33), thus we use the $cSim(n_i)$ flag variable to detect similarity. For both cases, we use a safe bound, i.e., $w(n_i)[2]$, and we do not add the node unless it is greater than t (lines 25 and 35). Recall that these scores are actually lower bounds as they are produced by comparison against only already retrieved nodes.

8. Experimental Evaluation

We experimentally evaluate the proposed snippets and algorithms. We emphasize on effectiveness comparisons between the two types of diversified snippets, the two types of relevance and also against the nondiversified size-l snippets.[7] 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 513 MB and 800 MB 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.[a] We calculate global importance by using global ObjectRank.[9] For the DBLP dataset, we use the default setting used in Refs. 9 and 7, i.e., the G^A shown in Fig. A.1(a) and $d = 0.85$ and for Google+ the G^A presented in Fig. A.1(b) and also $d = 0.85$. We calculate $af(.)$ as in Ref. 7. We used an expert to classify each relation as a *sim* or *equi*. For *sim* relations, we compare the respective *naming attributes* only (where naming attributes are as defined in Ref. 8, e.g., names, paper's title). We used an expert to define these naming attributes (alternatively, we can semi-automate this by using the attribute clustering approach of Ref. 4). More precisely, we used Jaccard distance on the respective naming attributes (preliminary experimentation revealed that alternative techniques (such as IR) have insignificant impact on the overall effectiveness results; thus we present results only using Jaccard). Recall that an *equi* size-l considers only *equi* relevance whereas a *sim* size-l considers both *sim* and *equi* relevance. For proportionality, we use $\alpha = 2$. We used Java, MySQL, and a PC with an AMD Phenom 9650 2.3 GHz (Quad-Core) processor and 8 GB of memory.

8.1. *Effectiveness*

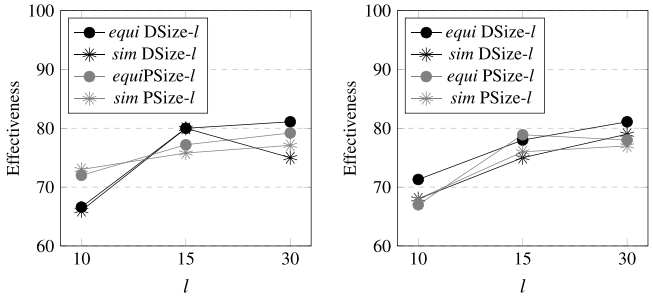
We conducted an effectiveness evaluation with the help of human evaluators. The evaluators were professors and researchers from our Universities. None of our evaluators were involved in this paper. Because of the complexity of the evaluation (we have to compare five different types of snippets), we used evaluators with expertise in the topics, we investigate. In particular, since the DBLP database includes data about real people, we asked the DSs themselves where possible (i.e., eleven authors or students of authors listed in DBLP) to participate in this evaluation. The rationale of this evaluation is that the DSs themselves (even their students) have the best knowledge of their work and can therefore

[a]http://snap.stanford.edu/data/egonets-Gplus.html.

provide accurate summaries. For Google+, we presented 10 random OSs to nine evaluators. First, we familiarized them with the concepts of OSs in general and the five types of size-l OSs. Specifically, we explained that a good size-l OS should be a standalone 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 and the difference between the two relevance types. However, we avoided to discuss the advantages or disadvantages of these combinations of types as to avoid any bias. In order to assist them with their tasks, we provided them useful information per node, such as $fr(.), li(.), dv(n_i|\emptyset), pq(n_i|\emptyset)$ and $w(n_i|\emptyset)$. For instance, we provided them, with the amount of times the co-author C. Faloutsos appears in the M. Faloutsos OS, his $li(.)$ etc. We also provided summarized ranked tables (similar to Tables 2 and 3 at the end of each OS) with the top-10 most frequent and top-10 most important nodes and their respective $w(n_i|.)$ scores.

8.1.1. *Precision and recall*

We provided evaluators with OSs and asked them to DSize-l and PSize-l them using both types of relevance (i.e., equality and similarity) for $l = 10, 15, 30$. Figure 8 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 8(a) and 8(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 Fig. 8 are obtained using the LASP algorithm (as the BF-l algorithm was prohibitively expensive). We omit results obtained by our other approximate algorithms as they do not vary from these results. For instance, the 2-LASPe 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 later, they have very minor impact on the quality of the computed snippets.



(a) DBLP Author (b) Google+ User

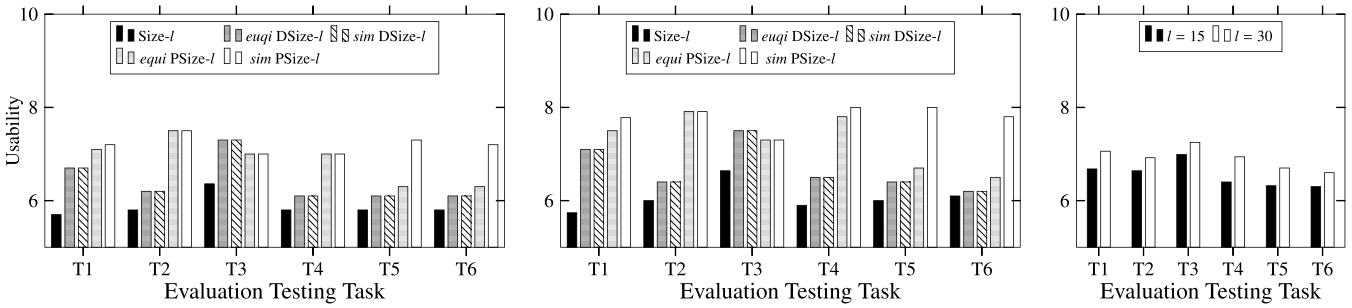
Fig. 8. Effectiveness (i.e., Recall=Precision).

8.1.2. *Usability test*

We conducted a comparative study of the usability of the five types that verifies users' preference for *sim* over *equi* relevance and DSize-l and PSize-l OSs over size-l OSs. In summary, the evaluation reveals the usability superiority of *sim* PSize-l OSs over all other types. 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.[b] More precisely, for a given OS, we measured the ease of use of all types through a usability test. We presented to users the various versions of size-l OSs in a random order to avoid any bias and we also gave them six 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 in their answers, where possible, the usability of the five approaches when completing these tasks. Namely, to score them considering (1) the ease of accomplishing each task, (2) how easy and (3) satisfying are to learn and use.

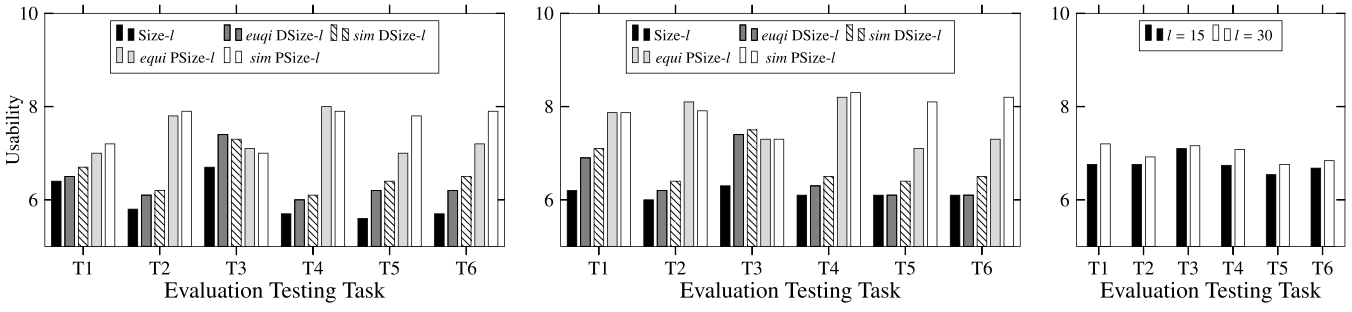
More precisely, the first task (T1) was to score the general use of all types; namely which one they prefer as a representative and informative snippet. For this purpose, we emphasized again that a snippet should be short, stand-alone and a meaningful synopsis of the most important and representative 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 the DBLP Author, Task 2 (T2) was to determine 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) was to determine the most important co-authors (e.g., whether C. Faloutsos and S. Madden are among the most important co-authors of M. Faloutsos). Task 4 (T4) was to determine the most frequent journal/conference the DS has published. Task 5 (T5) was to determine the most frequent topic of an author's papers (i.e., repeated set of keywords appearing in author's papers). Finally, Task 6 (T6) was to determine the most frequent topic appearing in papers citing an author's papers. Analogous tasks were used for the Google+ User. Namely, T2 was to determine a couple of the most frequent users in the DS's circles; T3 was to determine a couple of the most important users in DS's circles; T4 was to determine the most frequent user making comments on DS's activities; T5 was to determine the most frequent topic of the DS's comments and T6 was to determine the most frequent topics of DS's activities. Note that for comparison purposes, we maintain an analogy between the respective tasks of the two databases, e.g., T2 of both databases aim to determine the most

[b]www.wikipedia.org/wiki/Usability.



(a) $l = 15$ (b) $l = 15$ (c) *Average per l*

Fig. 9. Usability on DBLP author using *equi* and *sim* relevance.



(a) $l = 15$ (b) $l = 15$ (c) *Average per l*

Fig. 10. Usability on Google+ user using *equi* and *sim* relevance.

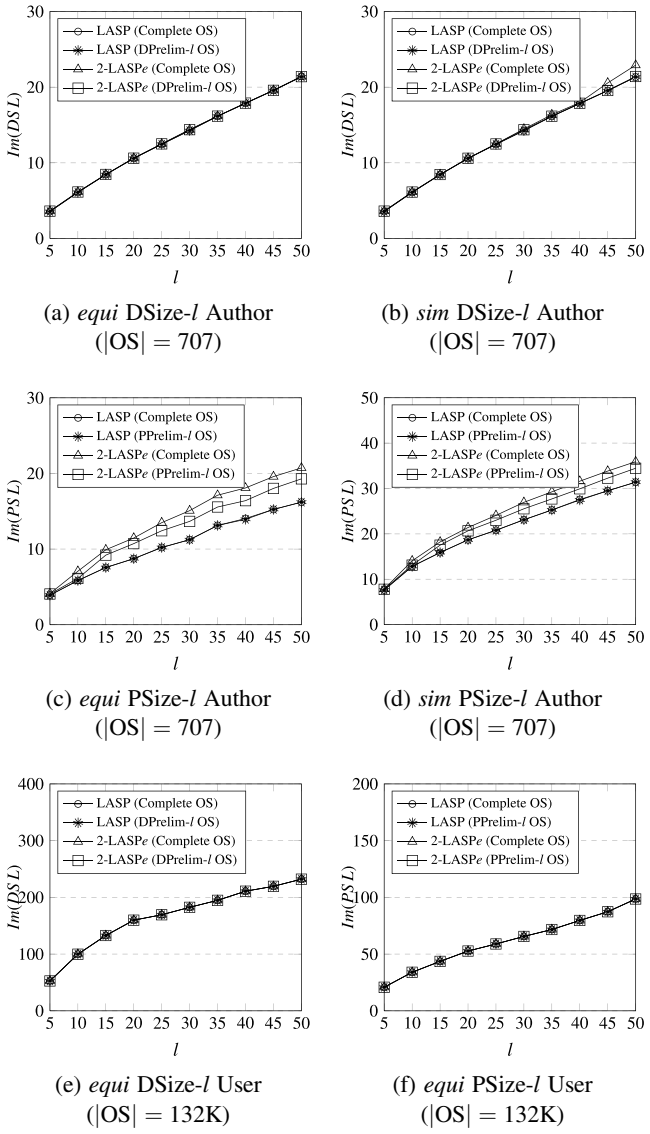
frequent co-authors/users associated with the DS, whereas T3 to determine the most important co-authors/users, etc.

Figures 9 and 10 average the evaluators' usability scores of all methods per G^{DS}, per task and per l. More precisely, respective subfigures (a) represent scores for $l = 15$, (b) for $l = 30$, and (c) the average of all tasks per l. The results show that evaluators preferred firstly *sim* PSize-l OSs, secondly *equi* PSize-l OSs, then *sim* and *equi* DSize-l OSs and lastly size-l OSs for both datasets. They also preferred size $l = 30$ over $l = 15$. For instance for the Author G^{DS}, the average scores of all tasks and both values of l, for *sim* PSize-l OSs is 7.5, for *equi* PSize-l OSs is 7.1, for *sim* DSize-l OSs is 6.7, for *equi* DSize-l OSs is 6.6 and finally for size-l OSs is 6.0. Evaluators expressed very similar preference for *equi* and *sim* DSize-l OSs because the snippets of these two types are almost identical (i.e., their constituent nodes are almost the same). The reason is that for the specific DBLP Author G^{DS}, both relevance types *equi* and *sim* result to the same DSize-l OSs; as the *sim* relevance will only impact towards the avoidance of including papers or conferences with frequent textual similarity to already added nodes (which was not very often in these cases).

The evaluators also provided justifications for their scores. We summarize them for each type and l and we also analyze their reflection on the given tasks. The evaluators explained that in general they prefer the concept of PSize-l OS as it also considers frequent nodes and topics; this is a property other types do not consider. This is evidenced by the superiority of the usability of PSize-l for tasks T2, T4–T6, since these tasks consider the frequency of nodes and topics. In addition, the evaluators explained that they found useful results considering the frequency of keywords (i.e., frequent topics); this is evidenced by high scores of *sim* PSize-l for tasks T5 and T6 which address the frequency of topics in the results. However, as they pointed out, although the inclusion of repeated frequent items or topics is informative, it comes at the cost of excluding other important nodes. They found that a DSize-l OS is very useful in covering the most important elements of an OS (i.e., evidenced by high scores of DSize-l for Task 3); however, they pointed out that rare but important elements may appear which again can be misleading to some extent. They found the nondiversified size-l summaries[7] more misleading as very important nodes are too dominant in them. The evaluators stated that l values of around 30 are the most appropriate, since the corresponding snippets include sufficient descriptive information about the corresponding OSs, giving a better representation of frequent and important information, and without being overwhelmingly large. This is also evidenced by Figs. 9(c) and 10(c).

8.2. *Quality of snippets*

We now compare the holistic importance $Im(.)$ scores of DSize-l and PSize-l OSs produced by the greedy methods.



(a) *equi* DSize-l Author ($|\text{OS}| = 707$)

(b) *sim* DSize-l Author ($|\text{OS}| = 707$)

(c) *equi* PSize-l Author ($|\text{OS}| = 707$)

(d) *sim* PSize-l Author ($|\text{OS}| = 707$)

(e) *equi* DSize-l User ($|\text{OS}| = 132\text{K}$)

(f) *equi* PSize-l User ($|\text{OS}| = 132\text{K}$)

Fig. 11. Quality on DBLP and Google+.

More precisely, the results of Fig. 11 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 ($\overline{|\text{OS}|}$)). The results show that in most cases, the results of LASP and 2-LASP*e* are of very similar (or even identical) quality, i.e., they have similar (or equal) holistic $Im(.)$ scores. The evaluation also reveals that using the DPrelim-l and PPrelim-l OSs results to very minor (even to zero) quality loss compared to using the complete respective OSs; e.g., by using LASP on either the complete OS or on the corresponding DPrelim-l OS, we obtain a DSize-l OS of the same $Im(\text{DS}l)$. More precisely, for the case of the DBLP Author *equi* PSize-l OSs, we get the maximum score loss by our algorithms; i.e., 2-LASP*e* Complete and LASP PPrelim-l algorithms return scores 20 and 15.5, respectively for $l = 50$. In the Google+ User case, respective quality remains the same for all (combinations of) algorithms (thus we omit *sim* DSize-l and *sim* PSize-l results). We did not compare with the optimal results, as the BF-l algorithm is too expensive.

8.3. *Efficiency*

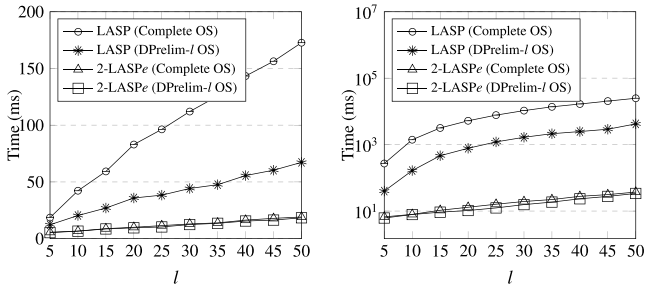
We compare the run-time performance of our greedy algorithms in Figs. 12–14. We used the same OSs as in Sec. 8.2 (i.e., the same 10 OSs per G^{DS}). Figures 12 and 13 show the costs of our algorithms for computing DSize-l (respectively PSize-l) for both types of relevance (*equi* and *sim*), excluding the time required to generate and pre-process OSs (i.e., the generation of $w(.), ap(.)$ scores, etc.), where each algorithm operates on.

More precisely, Fig. 12 show the costs of our algorithms for computing size-ls from OSs of the two G^{DS} 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^{DS}. Figures 13(a) and 13(b) show the scalability for Author PSize-l of different sizes, after fixing $l = 10$ (analogous results were obtained from User G^{DS} 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 1,309 has a PPrelim-10 size 157 (i.e., 11% of the size of the complete OS). From Figs. 12 and 13, we can see that the use of 2-LASP*e* on a preliminary OS is the fastest approach. For instance, *equi* DPrelim-l 2-LASP*e* for $l = 50$ requires only 18.3 ms. The results also verify that the use of *sim* relevance is more expensive than the use of *equi* relevance as it dictates the comparison of each node against all other OS nodes. For instance, *sim* DPrelim-l 2-LASP*e* for $l = 50$ requires up to 33.9 ms (which remains a practical time).

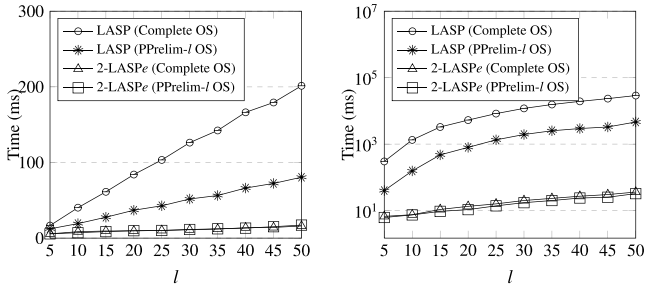
Finally, Figs. 14(a) and 14(b) break down the cost to OS generation and pre-processing time (bottom of the bar) and size-l computation (top of the bar) for each method for PSize-l. The figures also show (on the x-axis) the average sizes of the complete OSs and the PPrelim-l OSs for $l = 10$ and $l = 50$, respectively. For instance, the average size of the complete OS is 707; whereas the average sizes of the corresponding *equi* PPrelim-10 and PPrelim-50 OSs are 119 and 272.

Evidently, 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*e* 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*e* is the node with its parent only). The comparison of Figs. 14(a) and 14(b) verifies again that *sim* relevance is more demanding than *equi* relevance. In general as expected, the OS size, l and *sim* relevance negatively affect the cost.



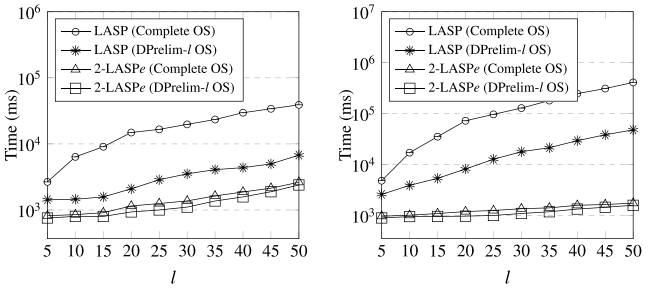
(a) *equi* DSize-l Author ($|\text{OS}| = 707$)

(b) *sim* DSize-l Author ($|\text{OS}| = 707$)



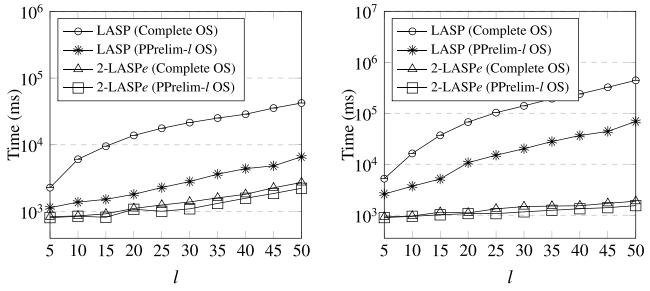
(c) *equi* PSize-l Author ($|\text{OS}| = 707$)

(d) *sim* PSize-l Author ($|\text{OS}| = 707$)



(e) *equi* DSize-l User ($|\text{OS}| = 132\text{K}$)

(f) *sim* DSize-l User ($|\text{OS}| = 132\text{K}$)

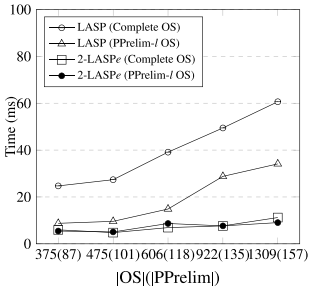


(g) *equi* PSize-l User ($|\text{OS}| = 132\text{K}$)

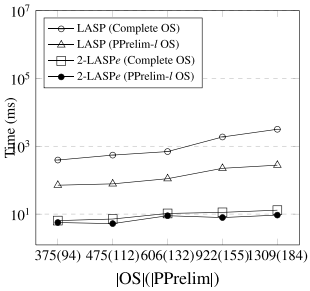
(h) *sim* PSize-l User ($|\text{OS}| = 132\text{K}$)

Fig. 12. Efficiency on DBLP and Google+.

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., 16 ms for the Author R^{DS}), we had to terminate the algorithm for $l \geq 10$ as it exceeded 30 min 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 Sec. 8.2, their results are snippets of high quality. In addition, the use of preliminary OSs and 2-LASPe is constantly a better choice over the complete OSs and LASP respectively since they are always faster with a negligible quality loss.

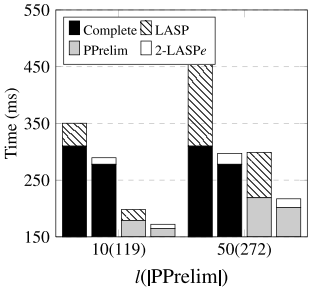


(a) *equi* PSize-l Author (Size-l = 10)

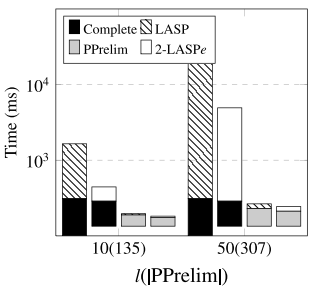


(b) *sim* PSize-l Author (Size-l = 10)

Fig. 13. Efficiency (verying OS size).



(a) *equi* PSize-l Author ($|\text{OS}| = 707$)



(b) *sim* PSize-l Author ($|\text{OS}| = 707$)

Fig. 14. Efficiency (cost breakdown).

9. Conclusion

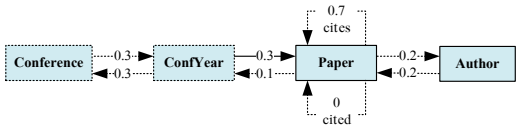
In this paper, we introduced the concept of object summary and size-l OSs, we also investigated the effectiveness and efficiency of two novel types of size-l OSs, namely DSize-l OSs and PSize-l OSs. For this purpose, we employed two types of nodes pairwise relevance, i.e., similarity and equality. Meanwhile, we proposed 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 nondiversified size-l OSs.[7] The evaluation also verified preference for results produced using *sim* relevance over

results produced by using *equi* relevance that was proposed in Refs. 19 and 20.

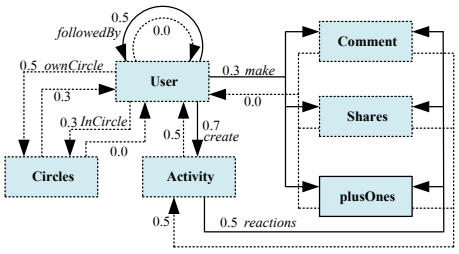
Acknowledgement

The work of Zhi Cai was partially supported by the Beijing Natural Science Foundation under grant number 4172004, and Beijing Municipal Education Commission Science and Technology Program under grant number KM201610005022.

Appendix



(a) The DBLP G^A



(b) The Google + G^A

Fig. A.1. The G^As for the DBLP and Google + Datasets.

References

[1]A. Turpin, Y. Tsegay, D. Hawking and H. E. Williams, Fast generation of result snippets in web search, *SIGIR* (2007), pp. 127–134.

[2]V. Hristidis and Y. Papakonstantinou, Discover: Keyword search in relational databases, (VLDB) (2002), pp. 670–681.

[3]Georgios John Fakas, Automated generation of object summaries from relational databases: A novel keyword searching paradigm, *DBRank, ICDE* (2008), pp. 564–567.

[4]Georgios John Fakas, A novel keyword search paradigm in relational databases: Object summaries, *DKE* **70**(2), 208 (2011).

[5]G. Cheng, T. Tran and Y. Qu, Relin: Relatedness and informativeness-based centrality for entity summarization, *The Semantic Web-ISWC* (2011), pp. 114–129.

[6]M. Sydow, M. Pikula and R. Schenkel, The notion of diversity in graphical entity summarisation on semantic knowledge graphs, *J. Intel. Infor. Syst.* **10**(2), 1 (2013).

[7]G. J. Fakas, Z. Cai and N. Mamoulis, Size-*l* object summaries for relational keyword search, *PVLDB* **5**(3), 229 (2011).

[8]G. J. Fakas, Z. Cai and N. Mamoulis, Versatile size-*l* object summaries for relational keyword search, *TKDE* **26**(4), 1026 (2014).

[9]G. J. Fakas, Z. Cai and N. Mamoulis, Diverse and proportional size-*l* object summaries using pairwise relevance, *VLDB J.* **25**(6), 791 (2016).

[10]A. Balmin, V. Hristidis and Y. Papakonstantinou, Objectrank: Authority-based keyword search in databases, *VLDB* (2004), pp. 564–575.

[11]G. J. Fakas and Z. Cai, Ranking of object summaries, *DBRank '08, ICDE* (2009), pp. 1580–1583.

[12]R. Fagin, A. Lotem and M. Naor, Optimal aggregation algorithms for middleware, *PODS* (2001), pp. 102–113.

[13]V. Hristidis, L. Gravano and Y. Papakonstantinou, Efficient ir-style keyword search over relational databases, *VLDB* (2003), pp. 850–861.

[14]Y. Luo, X. Lin, W. Wang and X. Zhou, Spark: Top-*k* keyword query in relational databases, *SIGMOD* (2007), pp. 115–126.

[15]S. Gollapudi and A. Sharma, An axiomatic approach for result diversification, *WWW* (2009), pp. 381–390.

[16]V. Dang and W. B. Croft, Diversity by proportionality: An election-based approach to search result diversification, *SIGIR* (2012), pp. 65–74.

[17]S. Cheng, A. Arvanitis, M. Chrobak and V. Hristidis, Multi-query diversification in microblogging posts, *EDBT* (2014), pp. 133–144.

[18]L. Wu, Y. Wang, J. Shepherd and X. Zhao, An optimization method for proportionally diversifying search results, *Adv. Knowl. Discov. Data Min.* **70**(2), 390 (2013).

[19]L. Wu, Y. Wang, J. Shepherd and X. Zhao, Diverse and proportional size-*l* object summaries for keyword search., *SIGMOD* (2015), pp. 363–375.

[20]L. Wu, Y. Wang, J. Shepherd and X. Zhao, Diverse and proportional size-*l* object summaries using pairwise relevance, *VLDB J.* **25**(6), 791 (2016).