Short paper

Personalized Page Rank on Knowledge Graphs: Particle Filtering is all you need!

Denis Gallo
University of Trento
denis.gallo@alumni.unitn.it

Matteo Lissandrini
Aalborg University
matteo@cs.aau.dk

Yannis Velegrakis
Utrecht University
i.velegrakis@uu.nl

ABSTRACT

Graphs are everywhere. Personalized Page Rank (PPR) is a particularly important task to support search and exploration within such datasets. PPR computes the proximity between query nodes and other nodes in the graph. This is used, among others, for entity exploration, query expansion, and product recommendation. Graph databases are used for storing knowledge graphs. Unfortunately, the exact computation of PPR is computationally expensive. While different solutions have been proposed to compute PPR values with high precision, these are extremely complex to implement, and in some cases require heavy preprocessing. In this work, we sustain that a better approach exists: *particle filtering*. Particle filtering methods produce ranks with sufficient precision while exploiting what graph databases architectures are already optimized for: navigating local connections. We present the implementation of such an approach in a popular commercial database and show how this outperforms the already implemented functionality. With this, we aim to motivate future research to optimize and improve upon this research direction.

1 INTRODUCTION

Graphs are everywhere [20], in particular, Knowledge Graphs (KG) [17] gained increasing attention thanks to their ability to represent entities and their relationships in many domains. Knowledge graphs model entities as nodes and the relationships among them as labelled edges. They are used to store the relationships about products, customers, events, locations, and more.

Personalized Page Rank [4, 5, 9, 12] (PPR) is a particularly important task to support search and exploration within graphs. At a high level, PPR extends the well known Page Rank [18] by computing a *local* popularity (or *proximity*) instead of a global importance score for nodes. In practice, given a small set of query entities, PPR returns a ranked list of other relevant entities based on a computed *random-walk proximity* to the query nodes. Famous examples of the application of PPR are the Twitter *Who To Follow* [8] that suggest to users other users to follow and the Pinterest related pins suggestions [14]. In other contexts, it can suggest related scientific articles, related entities, or suggest products to buy.

When it comes to PPR applications to KGs, a number of aspects become important, namely: the possibility to use a small set of nodes as query (e.g., for product recommendation or query expansion), the ability to include edge weights in the computation (given that in a KG different edges have different semantics), and fast response times for top-k queries (since in practical cases only a small set of high ranked nodes are required in contrast to computing such values for all nodes in the graph).

Currently, graph data management systems (GDBMS) [13] are the de-facto solution for managing knowledge graphs in transactional settings, thus, they need to support PPR queries. Unfortunately, the exact calculation of PPR is computationally expensive. In the literature many solutions have been proposed to provide fast computation of PPR values [2, 6, 7, 10, 11, 15, 19, 22–25]. Yet, they have been designed with the needs of social networks in mind, i.e., they focus on single source queries, unlabeled edges, and high-precision PPR value computations (required for community detection [26]). In many cases, they require pre-computation of indexes and other data-structures. In short, most existing methods are not designed to focus on the real needs for KG search, neither they take into account the requirements for being implemented in real-world Graph DBMS.

In this work, we study PPR computation solutions designed around the needs of Knowledge Graph search and the strengths of graph databases. We proposed a much simpler approach for computing Personalized Page Rank queries with multiple sources and heterogeneous edge weights. To achieve a significant performance in computing the top-k PPR, we extend the *damping function* template [4] with the *particle filtering procedure* [12], extended to correctly take into account the *teleportation probability* and account for the non-uniform edge importance typical of KGs. Currently, the only implementation of a similar approach has been proposed for an in-memory research prototype [16]. In our work, instead, we show how real word commercial graph databases can support this functionality.

Our experiments, on real large graphs, demonstrate the superiority of this approach (which we have made available as open-source[1]) against the currently implemented version in a major commercial GDBMS (Neo4j[2]). Furthermore, this direction is open to interesting challenges and can foster the development of new techniques to improve real-world graph databases.

2 RELATED WORKS

Personalized Page Rank [9] has been initially proposed as an alternative to global Page Rank since it computes local proximity to query nodes based on random-walks. Since that seminal work, many different approaches and implementations have followed. In general, they follow three alternative strategies, namely (1) matrix computation, (2) Monte-Carlo simulations, or (3) local search. For matrix computation, the graph is represented as its adjacency matrix, as in the original formulation, and different matrix multiplications are performed to compute the final values. These approaches [7, 10, 11, 22] are typically computationally expensive, hence they tend to be optimized through heavy pre-computations and large scale indexing. Matrix computation approaches are impractical for real-world graph databases, since GDBMS do not usually represent a graph as an adjacency matrix, and would require to maintain the pre-computed results.

[1] https://github.com/DenisGallo/Neo4j-ParticleFiltering
[2] https://neo4j.com/docs/graph-algorithms/

Monte-Carlo approaches, on the other hand, simulate a number of random traversals of the graph in order to compute the final PPR values. Employing this strategy offers few guarantees on the final output and on the actual response time. In general, a large number of random traversals are required to obtain reliable results. Hence, also in this case, precomputed values and indexes are employed [23, 24], suffering though from the same shortcomings discussed earlier.

Local search approaches, start, instead, from the query nodes and spread a rank value following the local neighborhoods until some stopping condition verifies. Usually, once this condition is encountered, they employ some additional Monte-Carlo traversals to refine the obtained approximate results and ensure higher precision [2, 5, 6, 15, 19, 24, 25]. Yet, most of these approaches have focused on either source-target queries, i.e., compute the PPR value of a target node given a source node, or single source queries with focus on computing high precision PPR values for very large portions of nodes in the graph. Both these directions, while useful in cases like social networks, impose an unnecessary burden on the system (since they focus on the actual PPR value instead of just computing a ranking of nodes) and neither address other important required features, like the necessity to differentiate edge types, the possibility to have multiple source nodes in the query, or the requirement of returning a ranked list of nodes of usually small size (i.e., top-k queries).

Contrary to most recent literature, following promising preliminary results from knowledge graph exploration [16], we study a method to enable multi-source, edge-weighted, top-k Personalized Page Rank queries within a Graph DBMS. We explicitly focus on the now prevalent need of Knowledge Graph search and on fully exploiting the ability of graph databases to efficiently query the local neighborhood of nodes [13]. Our method extends the *damping function template* [4], yet it implements an approach similar to *particle filtering procedure* [12] with two main differences: it has been extended to correctly take into account the *teleportation probability* that was not accounted for there, and it provides the ability to include edge relevance in the calculation, a feature that has been largely neglected in the literature. This solution has the advantage that it does not require any internal ad-hoc data-structure (our solution is a stored procedure of about two hundreds lines of code) and obtains both fast response time and high-quality results in practice.

3 APPROXIMATE PERSONALIZED PAGE RANK COMPUTATION

Given a graph $G : \langle V, E\rangle$ with V nodes and E edges, the result of a Personalized Page Rank (PPR) [9] given a set of query nodes $Q \subset V$ computes a proximity value of every node in V to the nodes in Q. Formally, the result of the computation is represented as a vector $\mathbf{v}$, with size $|V|$ representing the stationary distribution of the Markov chain [9] with state transition given by the equation

$$(1 - c)A\mathbf{v} + c\mathbf{p} \tag{1}$$

Given the column normalized transition probability matrix A, the teleportation probability c, and the preference vector $\mathbf{p}$. The matrix A (of size $|V| \times |V|$) contains values between 0 and 1 according to the probability that an edge is traversed (hence the column normalization), where a value of 0 corresponds to non-existing edges. In general, a KG is a graph with edges of different types (an edge-labelled graph) and in many cases, different edge types are assigned different relevance scores (i.e., a weight in [0,1]). Furthermore, $\mathbf{p}$ is an $|V| \times 1$ column vector, which serves as the normalized preference vector, for which $\mathbf{p}[n] \neq 0$ and in particular $0 < \mathbf{p}[n] \leq 1$ *iff* $n \in Q$. Finally, the *teleportation* probability $c \in (0, 1)$ is typically ≈ 0.15 in the literature [18].

Algorithm 1 PPR by Particle Filtering

Require: Graph G; Query nodes $\mathbf{Q}$
Require: Restart probability $c \in [0, 1]$; Threshold $\tau \in [0, 1]$
Require: Query value k
Ensure: Ranked Top-K nodes

```
1:  p ← {}
2:  for each q_i ∈ Q do
3:      p[q_i] ← 1/τ                                  ▷ Initialize Particles
4:  while ∃ n_i ∈ p | p[n_i] ≠ 0 do
5:      temp ← {}
6:      for each n_i ∈ p | p[n_i] ≠ 0 do
7:          particles ← p[n_i] × (1 − c)
8:          for each e : (n_i → n_j) ∈ G do            ▷ Sorted by Weight
9:              if particles ≤ τ then
10:                 break
11:             passing ←MAX(particles × e.weight(),τ)
12:             temp[n_j] ← temp[n_j] + passing
13:             particles ← particles − passing
14:     p ← temp
15:     for each n_i ∈ p do
16:         v[n_i] ← v[n_i] + p[n_i] × c               ▷ Update score
17: return top-k(v)
```

In practice, the goal of the PPR value is to rank nodes, hence *the exact value of the PPR is not necessary as far as the ranking is preserved.* We propose an approximation of this process [4] and apply an approach similar to the *weighted particle filtering procedure* [12] to consider the non-uniform edge weights.

The approach simulates a set of $1/\tau$ floating particles (lines 2-3, Algorithm 1) starting from each node in the query set Q. At each iteration (lines 5-16), the particles distribute among the neighbors of the current node (minus the number of particles that *restart*, line 7). An important optimization is to prevent particles to split to arbitrarily small sizes, limiting them to a minimum of τ (lines 9-11). When distributing the particles among the neighbors, the algorithm gives preference to the edges with higher weights (line 8). Since the weight is normalized on the edges of each node, this operation matches the damping function framework [4]. The restart probability c will dissipate part of the particles at every iteration (line 8), and the algorithm will stop when no more particles can be distributed. During the process, a vector $\mathbf{v}$ accumulates the total amount of particles visiting each node (lines 15-16). Hence, the final list of top-k nodes is based on $\mathbf{v}$.

Here, we argue that for the case of Knowledge Graphs, the Particle Filtering approach for PPR computation is the best-suited approach to extend GDBMS functionalities for retrieving Top-K nodes.

The benefits of this approach are that (1) it does not require any complex preprocessing nor any additional persistent data structure, (2) it is directly implementable within any graph databases by direct use of core operations that are already optimized (namely local node neighbor traversal [13]), (3) it returns a ranking that strictly correlates with the actual ranking, (4) it can account for heterogeneous edge types and importance, (5) it does not require to traverse the entire graph and its exploration rate (and hence running time) can be fine-tuned through the τ and c parameters.

4 EXPERIMENTS

We investigated the performance and the quality of the ranking of Particle Filtering (PF) compared to the current exact implementation of Personalized Page Rank (PPR) in a commercial database. In particular, we implemented particle filtering (**PF** – Algorithm 1) as a Java stored procedure in Neo4j[3] v3.5.5, and compared it against the current implemented exact solution (**PPR** - Graph Algorithms v3.5.2)[4]. Experiments have been executed on a server with 12 cores and 128GB RAM.

Datasets: We compared the two alternatives on 4 different KGs from different domains and different sizes, widely used in the literature (Table 1). These KGs have been obtained from a dataset of movies, including also information about actors, directors, and genres[5] (*Movies*); from an established benchmark for triplestores [1] representing an heterogenous product catalog (*WatDiv*); an open domain knowledge graph (*DBpedia* [3]); and finally a knowledge base about drugs, their composition, and their interactions (*DrugBank* [21]). Of these, Movies, DBpedia, and DrugBank are real-world datasets, while WatDiv is synthetic.

Queries, Parameters, and Evaluation Metrics: For each dataset, we extracted 5 sets of 20 queries. Each set contains 20 queries of the same size (i.e., number of source nodes). We generated through sampling queries of size 1, 5, 10, 20, and 100 nodes, for a total of 100 queries. For each query, we recorded the execution time (average of 3 runs), and for the queries executed with PF, we recorded NDCG score at top-5, 50, 100, and 500. We executed queries both with the weighted (i.e., assigning weights based on label informativeness [16]) and unweighted (i.e., uniform weights) version of the algorithm. We tested the PF with three values of τ: 0.1, 0.05, and 0.01.

Results: Due to space constraints, we report here only results for the two largest graphs, namely WatDiv and DBpedia, for the labelled case. The full set of experimental results can be found in the extended version of this document[6]. Nonetheless, we also comment on some of the findings of the excluded experiments.

Quality of Ranking: Over all the datasets, the NDCG score for PF with $\tau = 0.01$ has the best quality, and often close to the perfect ranking (i.e., between 0.8 and 1.0) with queries containing up to 10 nodes. In most cases, also PF with $\tau = 0.05$ obtains a good quality ranking (NDCG>0.65). With more than 10 query nodes, the quality of ranking is subject to high variability, especially depending on the dataset. On DBpedia and WatDiv, still, we obtain NDCG scores above 0.65 for both τ=0.05 and 0.01 with 20 nodes in input at top-500, while for 100 nodes, we need τ=0.01 on DBpedia. This confirms the suitability for KG exploration cases.

Running Time: Compared to the running time of exact PPR, the PF algorithm provides a speedup between 1 and 4 orders of magnitudes, i.e., returning on average in 1-30 seconds while the exact solution requires 3-6 minutes (on DBpedia). In general, the speedup is proportional to the value of τ, i.e., τ=0.01 is between 10 and 100 times slower than τ=0.1. Yet, for 100 query nodes on DBpedia, we report that τ=0.01 rarely achieves sensible improvements due to the high number of particles generated.

Effect of Weights: when considering weights particle transitions are skewed towards more relevant nodes (through more informative edges). This has a noticeable effect on the running time because for high degree edges only the most informative edges are traversed (given the fact that they are prioritized), and many less relevant edges are skipped. For a similar reason, we notice that the NDCG score when weighted edges are considered is higher because for nodes with very high degrees, there are not enough particles to visit all the neighbors, hence in the weighted case the PF prioritization is consistent with the importance of the node, while in the unweighted case all of them should be visited.

	Movies	WatDiv	DBpedia	DrugBank
#Nodes	63K	5.2M	11.6M	391K
#Edges	106K	95.8M	216.7M	1M
#Edge types	4	31	13k	68
Density	3.93E-05	3.48E-06	1.61E-06	6.82E-06
#Con. Components	433	1	2	1
Min size CC	3	5.2M	59.3k	391K
Max size CC	58.2k	5.2M	11.5M	391K
Avg size CC	142	5.2M	579K	391K
Median size CC	6	5.2M	579K	391K
Max Out-degree	71	345	7.2k	423
Avg Out-degree	2.13	18.4	22.6	3.3
Median Out-degree	1	1	9	2
Max In-degree	92	585K	3.3M	316K
Avg In-degree	9.18	20.3	20.1	2.8
Median In-degree	8	1	2	1

Table 1: Size and characteristics of the datasets

Open Challenges: In our experiments, we noticed that when starting nodes are hubs (i.e., nodes with high degree) or are near hubs, the weighted traversal is the bottleneck. The reason is that current GDBMS can very quickly retrieve all the neighbors of a node, but then we require to sort them by weight. Hence, we identify the opportunity in GDBMS for implementing *sorted edge iterators for weighted edges* to speed up this step and other similar cases. Moreover, automatic tuning of the τ threshold depending on the query is an open research challenge.

5 CONCLUSIONS

In this paper, we argue that, when computing the Personalized Page Rank value in a knowledge graph, the approximate computation framework offered by the Particle Filtering approach, provides substantial advantages in terms of running time and ease of implementation, while ensuring good ranking quality. Our implementation can provide an efficient solution for extending existing graph databases since it exploits the strengths of these systems. While this approach is simple and effective for queries with few input nodes and limited to the first few hundreds nodes (typical of on-line exploration settings), we believe that the algorithm can be further expanded to assure high-quality ranking also for nodes in the long tail and larger queries.

[3]https://neo4j.com/download-center/#community
[4]https://github.com/neo4j-contrib/neo4j-graph-algorithms
[5]https://neo4j.com/developer/example-data/
[6]http://people.cs.aau.dk/~matteo/pdf/EDBT2020-pf-long.pdf

REFERENCES

[1] Güneş Aluç, Olaf Hartig, M. Tamer Özsu, and Khuzaima Daudjee. 2014. Diversified Stress Testing of RDF Data Management Systems. In *ISWC'14*. 197–212.
[2] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local Graph Partitioning Using PageRank Vectors. In *FOCS '06*. 475–486.
[3] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. DBpedia: A Nucleus for a Web of Open Data. In *ISWC'07*. 722–735.
[4] Ricardo Baeza-Yates, Paolo Boldi, and Carlos Castillo. 2006. Generalizing PageRank: Damping Functions for Link-based Ranking Algorithms. In *SIGIR '06*. ACM, 308–315.
[5] Soumen Chakrabarti. 2007. Dynamic Personalized Pagerank in Entity-relation Graphs. In *WWW '07*. ACM, 571–580.
[6] Dániel Fogaras and Balázs Rácz. 2004. Towards Scaling Fully Personalized PageRank. In *Algorithms and Models for the Web-Graph*. Springer, 105–117.
[7] Yasuhiro Fujiwara, Makoto Nakatsuji, Takeshi Yamamuro, Hiroaki Shiokawa, and Makoto Onizuka. 2012. Efficient Personalized Pagerank with Accuracy Assurance. In *KDD '12*. ACM, 15–23.
[8] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. 2013. WTF: The Who to Follow Service at Twitter. In *WWW '13*.

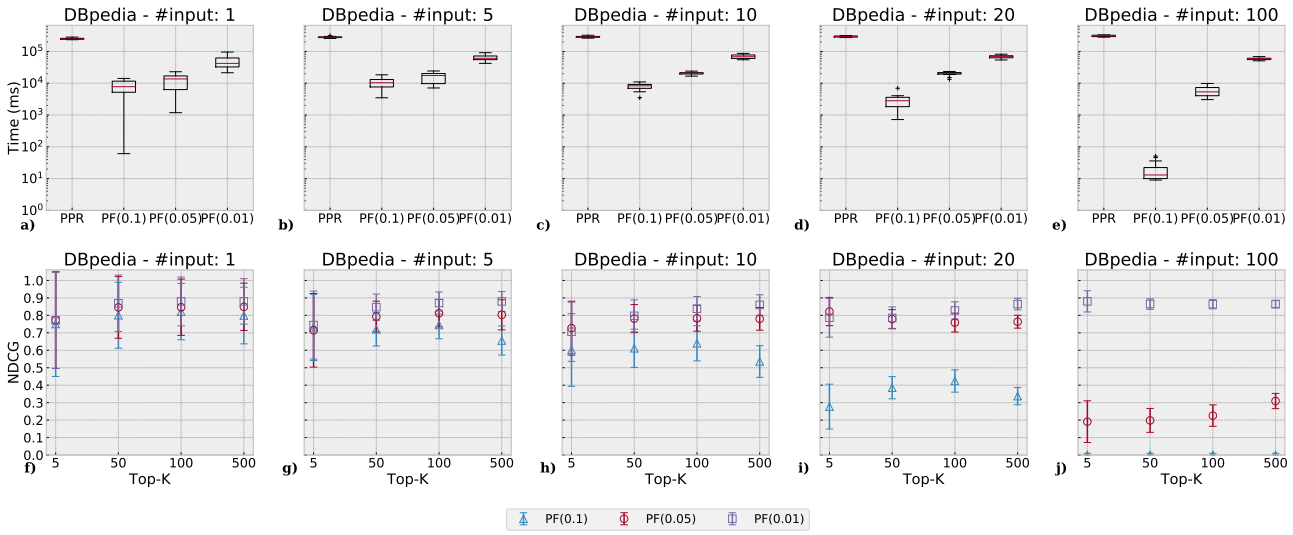

Figure 1: Running time (top) and NDCG (bottom) vs. weighted exact PPR on DBpedia, varying τ and # of input nodes.

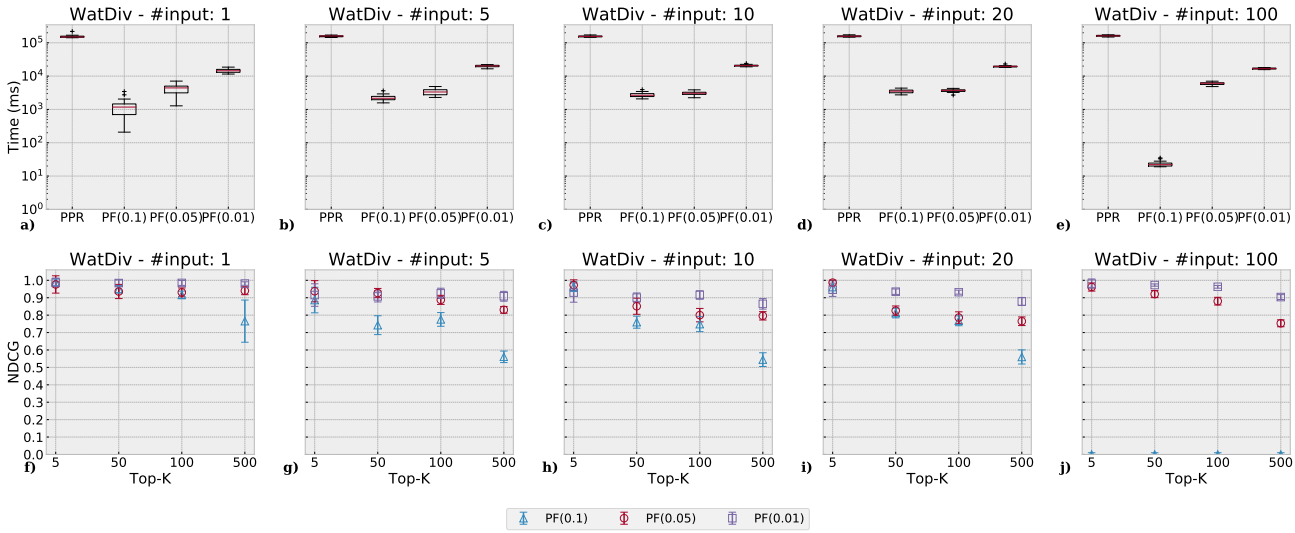

Figure 2: Running time (top) and NDCG (bottom) vs. weighted exact PPR on WatDiv, varying τ and # of input nodes.

[9] Glen Jeh and Jennifer Widom. 2003. Scaling Personalized Web Search. In *WWW '03*. ACM, 271–279.

[10] Jinhong Jung, Namyong Park, Sael Lee, and U Kang. 2017. BePI: Fast and Memory-Efficient Method for Billion-Scale Random Walk with Restart. In *SIGMOD '17*. ACM, 789–804.

[11] Jinhong Jung, Kijung Shin, Lee Sael, and U Kang. 2016. Random Walk with Restart on Large Graphs Using Block Elimination. *ACM Trans. Database Syst.* 41, 2 (2016), 12:1–12:43.

[12] Ni Lao and William W. Cohen. 2010. Fast Query Execution for Retrieval Models Based on Path-constrained Random Walks. In *KDD*. ACM, 881–888.

[13] Matteo Lissandrini, Martin Brugnara, and Yannis Velegrakis. 2018. Beyond Macrobenchmarks: Microbenchmark-based Graph Database Evaluation. *PVLDB* 12, 4 (2018), 390–403.

[14] David C. Liu, Stephanie Rogers, Raymond Shiau, Dmitry Kislyuk, Kevin C. Ma, Zhigang Zhong, Jenny Liu, and Yushi Jing. 2017. Related Pins at Pinterest: The Evolution of a Real-World Recommender System. In *WWW '17*. 583–592.

[15] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. 2016. Personalized PageRank Estimation and Search: A Bidirectional Approach. In *WSDM '16*. ACM, 163–172.

[16] Davide Mottin, Matteo Lissandrini, Yannis Velegrakis, and Themis Palpanas. 2016. Exemplar Queries: A New Way of Searching. *The VLDB Journal* 25, 6 (2016), 741–765.

[17] Natasha Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. 2019. Industry-scale Knowledge Graphs: Lessons and Challenges. *Queue* 17, 2, Article 20 (2019), 48–75 pages.

[18] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[19] Jia-Yu Pan, Hyung-Jeong Yang, Christos Faloutsos, and Pinar Duygulu. 2004. Automatic Multimedia Cross-modal Correlation Discovery. In *KDD '04*. ACM, 653–658.

[20] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M. Tamer Özsu. 2017. The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing. *PVLDB* 11, 4 (2017), 420–431.

[21] Muhammad Saleem, Gábor Szárnyas, Felix Conrads, Syed Ahmad Chan Bukhari, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. 2019. How Representative Is a SPARQL Benchmark? An Analysis of RDF Triplestore Benchmarks. In *WWW '19*. ACM, 1623–1633.

[22] Hanghang Tong, Christos Faloutsos, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast Random Walk with Restart and Its Applications. In *ICDM '06*. 613–622.

[23] Sibo Wang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. 2016. HubPPR: Effective Indexing for Approximate Personalized Pagerank. *PVLDB* 10, 3 (2016), 205–216.

[24] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: Simple and Effective Approximate Single-Source Personalized PageRank. In *KDD '17*. ACM, 505–514.

[25] Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Shuo Shang, and Ji-Rong Wen. 2018. TopPPR: Top-k Personalized PageRank Queries with Precision Guarantees on Large Graphs. In *SIGMOD '18*. ACM, 441–456.

[26] J. J. Whang, D. F. Gleich, and I. S. Dhillon. 2016. Overlapping Community Detection Using Neighborhood-Inflated Seed Expansion. *TKDE* 28, 5 (2016).