ATRAPOS: Real-time Evaluation of Metapath Query Workloads

Serafeim Chatzopoulos
U. of Peloponnese & IMSI, Athena RC
Athens, Greece
schatzop@uop.gr

Thanasis Vergoulis
IMSI, Athena RC
Athens, Greece
vergoulis@athenarc.gr

Dimitrios Skoutas
IMSI, Athena RC
Athens, Greece
dskoutas@athenarc.gr

Theodore Dalamagas
IMSI, Athena RC
Athens, Greece
dalamag@athenarc.gr

Christos Tryfonopoulos
University of Peloponnese
Tripolis, Greece
trifon@uop.gr

Panagiotis Karras
Aarhus University
Aarhus, Denmark
piekarras@gmail.com

ABSTRACT

Heterogeneous information networks (HINs) represent different types of entities and relationships between them. Exploring and mining HINs relies on *metapath queries* that identify pairs of entities connected by relationships of diverse semantics. While the real-time evaluation of metapath query workloads on large, web-scale HINs is highly demanding in computational cost, current approaches do not exploit interrelationships among the queries. In this paper, we present ATRAPOS, a new approach for the real-time evaluation of metapath query workloads that leverages a combination of efficient sparse matrix multiplication and intermediate result caching. ATRAPOS selects intermediate results to cache and reuse by detecting frequent sub-metapaths among workload queries in real time, using a tailor-made data structure, the *Overlap Tree*, and an associated caching policy. Our experimental study on real data shows that ATRAPOS accelerates exploratory data analysis and mining on HINs, outperforming off-the-shelf caching approaches and state-of-the-art research prototypes in all examined scenarios.

CCS CONCEPTS

• **Information systems** → **Query optimization**; • **Theory of computation** → **Caching and paging algorithms**.

KEYWORDS

HINs, metapath queries, overlap tree

ACM Reference Format:
Serafeim Chatzopoulos, Thanasis Vergoulis, Dimitrios Skoutas, Theodore Dalamagas, Christos Tryfonopoulos, and Panagiotis Karras. 2023. ATRAPOS: Real-time Evaluation of Metapath Query Workloads. In *Proceedings of the ACM Web Conference 2023 (WWW '23), April 30–May 04, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3543507.3583322

1 INTRODUCTION

A *heterogeneous information network (HIN)* is a Knowledge Graph where each node and edge is assigned one type [22]. HINs offer an intuitive and generic model to encapsulate complex semantic information via different types of nodes and edges [52], which both have internal structure including a set of *properties*. HINs are increasingly supported by modern data systems [68] or research prototypes [10, 11] and investigated in research [13, 16, 17, 23, 32, 46, 58–60, 67, 69, 71]. Figure 1 illustrates an example HIN with its *schema* (that can be explicitly or implicitly defined). It consists of nodes representing papers (P), authors (A), venues (V), and topics (T) and (bidirectional) edges of three types: authors – papers (AP / PA), papers – topics (PT / TP), and papers – venues (PV / VP).

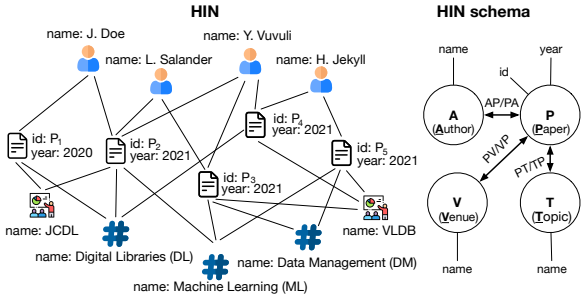


Figure 1: An example HIN and its schema.

Indirect relationships between entities are implicitly encoded by paths in the HIN. All paths that correspond to the same sequence of node and edge types (i.e., the same *'metapath'* [60]) encode latent relationships of the same interpretation between the starting and ending nodes. In the above example, the metapath ⟨*APTPA*⟩ relates authors that have published papers on the same topic (e.g., both `'J. Doe'` and `'H. Jekyll'` have papers about `'DL'`). Notice that each metapath corresponds to a path on the schema of the HIN.

Metapaths are instrumental for HIN analysis: the metapath-based connectivity can be used to define node similarity measures [49, 50, 60, 70] or to rank nodes based on their centrality in a metapath-defined network [30, 33, 37, 51]. In our example, using ⟨*TPT*⟩ to perform a similarity join could reveal that topics `'ML'` and `'DL'` are very related since they are involved in two common papers. To further elaborate the analysis, it is often useful to apply property-based *constraints* to metapaths (e.g., to consider only metapath instances involving papers published later than `'2020'`).

The first step in any metapath-based analysis is to generate a transformed network that contains edges between all pairs of nodes connected with one or more paths of a specified type. This is computationally expensive. While various methods aim to speed up the computation of a *single* metapath query, which typically involves a series of matrix multiplications [51], the problem becomes even

WWW '23, April 30–May 04, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9416-1/23/04.
https://doi.org/10.1145/3543507.3583322

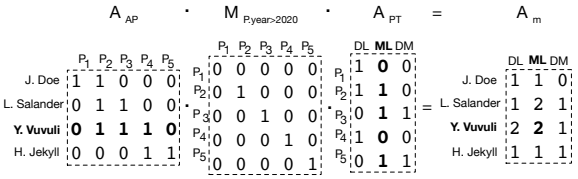


Figure 2: Evaluation of the $m = (\langle APT \rangle, \{P.year > 2020\})$ query on the HIN of Figure 1.

more severe in scenarios calling for the real-time evaluation of multiple metapath queries. In metapath-based feature selection [36, 47], different metapaths reveal different associations among entities, informing tasks like recommendation and link prediction [7, 12, 48]. Such a task requires a large number of metapath queries (i.e., a *query workload*), leading to a significant bottleneck in the data analysis process. Yet, in such a multi-query evaluation scenario, significant overlaps typically occur among metapath queries [18, 26, 42, 44, 45]; such overlaps are translated into repetitive matrix multiplications. We may avoid a large portion of these heavy computations using query materialisation; yet, no existing approach does so.

In this paper, we introduce Atrapos,[1] a metapath query evaluation approach that detects, in real time, frequent metapath overlaps within a sequence of queries. To do so, it uses: (i) sparse matrix representations, which leverage the fact that matrices involved in metapath computations are largely sparse, especially for constrained metapaths; and (ii) a tailored data structure with a customised caching policy to materialise reoccurring intermediate results. Our experimental evaluation demonstrates that Atrapos outperforms traditional, single-query approaches in the evaluation of metapath query workloads, while it is considerably faster than baseline approaches. To the best of our knowledge, this work is the first to study the problem of optimising the real-time evaluation of metapath query workloads. Our contributions are summarised as follows:

- We show how metapath computations can be accelerated using matrix multiplication algorithms tailored for sparse matrices along with an appropriate cost model.
- We introduce a new data structure, the *Overlap Tree*, that reveals overlaps among metapaths and a cache insertion policy using this structure to cache and reuse intermediate results in real-time while running a metapath query workload.
- We propose a tailored, effective cache replacement policy that exploits cache item interdependence.
- We conduct a thorough evaluation on real data showcasing the efficiency of our approach.

2 PRELIMINARIES & PROBLEM DEFINITION

A *heterogeneous information network (HIN)* [52, 60], is a directed multigraph that contains multiple types of nodes and edges:

Definition 1 (HIN). *A* HIN *is a tuple $\mathcal{H} = \langle V, E, O, R, \phi, \psi \rangle$, where V and E are the nodes and edges of a directed multigraph, respectively; O and R ($|O| > 1$, $|R| > 1$) are the sets of distinct node and edge types, respectively, while $\phi : V \rightarrow O$ and $\psi : E \rightarrow R$ are the mapping functions that determine the type of each node $v \in V$ and each edge $e \in E$, respectively.*

It is often useful to consider that each node type comes with a set of *properties*, which capture useful information for the nodes of that type.[2] Given a node $v \in V$, we use the notation $v.prop$ to refer to the value of the property $prop$ on v. All distinct node and edge types, along with the respective properties, can be easily derived by the HIN data and we often refer to them as the HIN's *schema*. Finally, it is often convenient to represent the edges of each type $r = \dot{o}\ddot{o}$, where $\dot{o}, \ddot{o} \in O$ and $r \in R$ as the *adjacency matrix* A_r, i.e., a $|\dot{o}| \times |\ddot{o}|$ matrix for which $A_r[i, j] = 1$ if the i^{th} node of type $\dot{o}$ connects to the j^{th} node of type $\ddot{o}$, and $A_r[i, j] = 0$ otherwise.

A *metapath* is a sequence of node types connected by appropriate edge types, hence corresponds to a path on a HIN schema. Given a metapath m, every path in the HIN that complies to the sequence of node and edge types specified by m is an *instance* of m. Formally:

Definition 2 (Metapath & Metapath instance). *Given a HIN $\mathcal{H} = \langle V, E, O, R, \phi, \psi \rangle$, a* metapath *$m$ on $\mathcal{H}$ is a sequence $m = \langle o_1 \xrightarrow{r_1} o_2 \ldots \xrightarrow{r_{n-1}} o_n \rangle$, where $o_i \in O$, $r_j \in R$ $\forall i, j$, and n is the metapath* length. *An* instance *of m is any path $\langle v_1 \xrightarrow{e_1} v_2 \ldots \xrightarrow{e_{n-1}} v_n \rangle$ for which $v_i \in V$, $e_j \in E$, $\phi(v_i) = o_i$, and $\psi(e_j) = r_j$ $\forall i, j$.*

By a common simplification [51, 60], we denote a metapath as $m = \langle o_1 o_2 \ldots o_n \rangle$, without edge types, when there is only a single edge type between any pair of node types. It should be highlighted that this convention is followed here solely for the sake of simplicity; all approaches can easily accommodate edges of different types among the same pairs of nodes (e.g., since they employ separate adjacency matrices for distinct edge types). Figure 1 shows an example HIN comprising 14 nodes of 4 types (A, P, V and T) and 3 edge types (AP/PA, PV/VP, and PT/TP). A simplified metapath of length 3 is $m = \langle APV \rangle$, and an instance of m is the path $\langle$'$Y.Vuvuli$' '$P3$' '$VLDB$'$\rangle$.

We also consider metapaths enhanced with constraints on node type properties, which restrict the nodes that may participate in a metapath instance. We refer to such metapaths as *constrained metapaths*. The set of instances of a constrained metapath is a subset of those of the corresponding unconstrained metapath. Formally:

Definition 3 (Constrained metapath). *A* constrained metapath *is a pair $m' = (m, C)$ where $m = \langle o_1 o_2 \ldots o_n \rangle$ is a metapath and $C = \{c_1, \ldots, c_n\}$ is a set of constraints on the nodes in m.*

Recall the metapath $\langle APTPA \rangle$ of the HIN in Figure 1, which connects two authors that have published a paper on the same topic. A data scientist working with this HIN may need to examine only recent papers, e.g., papers published after 2000, may use the constrained metapath $m' = (\langle APTPA \rangle, \{P.year > 2000\})$.

We now define the *metapath query evaluation* (MQE) problem.

Problem 1 (MQE). *Given a HIN $\mathcal{H} = \langle V, E, O, R, \phi, \psi \rangle$ and a (constrained) metapath $m = (\langle o_1 o_2 \ldots o_n \rangle, C)$, MQE retrieves all node pairs $\langle v_o, v_d \rangle$, such that $v_o \in o_1$, $v_d \in o_n$, and v_o is connected to v_d based on at least one instance of m. We refer to each such pair, along with the respective number of instances, as an* MQE result *for m.*

A conventional approach to evaluate a metapath query m is to calculate the matrix product $A_m = A_{o_1 o_2} A_{o_2 o_3} \ldots A_{o_{n-1} o_n}$, where $A_{o_i o_j}$ is the adjacency matrix for the relation $r = o_i o_j$. Each non-zero

[1] From Greek ἀτραπός, 'path', what is frequently trodden.

[2] It may be useful to assign properties to edges, as well, however for notation simplicity in this work we do not consider edge properties.

element in A_m is a response to the corresponding metapath query. Given a constraint $c_i \in C$ on object o_i in relation $r = o_i o_j$, we apply c_i in the matrix product via the *constrained* adjacency matrix $A^{c_i}_{o_i o_j} = M_{c_i} \cdot A_{o_i,o_j}$, where M_{c_i} is a $|o_i| \times |o_i|$ diagonal matrix that has element 1 in the diagonal only for nodes o_i satisfying c_i. Figure 2 shows an example for the metapath query $m = (\langle APT \rangle, \{P.year > 2020\})$. Based on the MQE problem, we define the *metapath query workload evaluation (MQWE)* problem:

Problem 2 (MQWE). *Given a HIN $\mathcal{H} = \langle V, E, O, R, \phi, \psi \rangle$ and a metapath query workload $W = \{m_1, \ldots, m_q\}$, MQWE is the task of efficiently computing all MQEs for the metapaths in W.*

We focus on MQWE and elaborate our approach in the following.

3 THE ATRAPOS APPROACH

Atrapos, our approach for efficiently evaluating metapath query workloads, (a) translates metapath query evaluation into a sequence of multiplications between the adjacency matrices of the edge types contributing to the respective metapaths, and (b) applies dynamic programming to identify an efficient order of multiplications (Section 3.1). Atrapos takes into account the inherent sparsity of the adjacency matrices to exploit fast sparse matrix multiplication algorithms, and identifies improved execution plans for each metapath using matrix multiplication cost models tailored for sparse matrices (Section 3.2). Lastly, Atrapos identifies sub-metapaths that reoccur due to metapath overlaps in a workload and caches the corresponding matrix multiplication results to avoid repeating them. To do so, it uses a specialised data structure, the *Overlap Tree*, which keeps track of metapath overlaps, occurrence frequencies, and cache entries (Section 3.3); Atrapos's cache management policy exploits the Overlap Tree to optimise cache replacements (Section 3.4).

3.1 Efficient Multiplication Plan Selection

Given a metapath query, Atrapos generates a set of alternative execution plans, each comprising a different sequence of multiplications between the adjacency matrices of the edge types involved, as in [51]. Due to associativity, there are many such orderings that produce the same result, and their computational costs may differ. For example, consider the query $m = \{\langle APT \rangle, P.year > 2020\}$ in Figure 2. The result $A_m = A_{AP} \cdot M_{P.year>2020} \cdot A_{PT}$ can be calculated as $(A_{AP} \cdot M_{P.year>2020}) \cdot A_{PT}$ or $A_{AP} \cdot (M_{P.year>2020} \cdot A_{PT})$. Since A_{AP}, $M_{P.year>2020}$, and A_{PT} are of size $[4 \times 5]$, $[5 \times 5]$, and $[5 \times 3]$, respectively, and a standard multiplication of two matrices of size $m \times n$ and $n \times l$ requires $m \cdot n \cdot l$ operations, the first option yields 160 operations, while the second 135, which is preferable.

By this standard implementation of matrix multiplication, the cost $C_{A_{i\ldots j}}$ of the optimal plan for a multiplication series $A_{i\ldots j} = A_i \cdot \ldots \cdot A_j$, $1 \le i < j$ via dynamic programming (DP) [15], is:

$$C_{A_{i\ldots j}} = \min_{i \le k < j} \{C_{A_{i\ldots k}} + C_{A_{k+1\ldots j}} + c_{A_{i\ldots k} \cdot A_{k+1\ldots j}}\} \quad (1)$$

where C is a $p \times p$ matrix of optimal costs and $c_{A_{i\ldots k} \cdot A_{k+1\ldots j}}$ the cost of multiplying the results of sub-series $A_{i\ldots k}$ and $A_{k+1\ldots j}$, estimated by matrix dimensions in the case of standard matrix multiplication.

3.2 Sparse Matrix Representation

The technique of Section 3.1 has been used to identify the optimal matrix multiplication plan for single metapath query evaluation [51]. However, most adjacency matrices in HINs are sparse (i.e., have few non-zero elements). Using a general-purpose matrix multiplication method on sparse matrices is inefficient. This sparsity is amplified if there are constraints in the metapath queries, which are represented by extra diagonal matrices with a small number of non-zero elements in the diagonal, whose multiplication with other matrices yields an even sparser result. Unfortunately, previous work opted for dense matrix representations [51].

By contrast, we use sparse matrix structures to represent adjacency matrices and multiply such matrices taking advantage of these structures. Still, the cost to multiply sparse matrices cannot be adequately estimated from their dimensions, as it also depends on the number of non-zero elements in input and result matrices, implementation details, and memory accesses. An approximation model [27] estimates the cost of multiplying sparse matrices X $[m \times n]$ and Y $[n \times l]$ to yield $Z = X \cdot Y$ as:

$$c_{\hat{X \cdot Y}} \approx \alpha \cdot \underbrace{(m \cdot n \cdot \rho_X)}_{nonZero(X)} + \beta \cdot \underbrace{(m \cdot n \cdot \rho_X \cdot l \cdot \rho_Y)}_{\hat{N_{op}}} + \gamma \cdot \underbrace{(m \cdot l \cdot \hat{\rho_Z})}_{\hat{nonZero}(Z)} \quad (2)$$

where ρ_X and ρ_Y are the densities of X and Y, $nonZero(X)$ the number of non-zero elements in X, $\hat{N_{op}}$ the estimated number of operations, $\hat{\rho_Z}$ the estimated density of Z, and $\hat{nonZero}(Z)$ the estimated number of non-zero elements in Z. Coefficients α, β, γ are determined by multilinear regression using least squares fit.

Assuming that the non-zero elements in X and Y are uniformly distributed, we compute $\hat{\rho_Z} = 1 - (1 - \rho_X \cdot \rho_Y)^n$ as indicated by the average-case density estimator E_{ac} [27, 56]. There exist more accurate estimation algorithms [14, 27, 56, 73], taking into consideration the pattern of non-zero elements in X and Y, yet they incur a higher computational cost. Appendix A reports experimental evidence that the computational savings of more accurate estimation does not pay off for the extra computational cost in our scenario.

In a nutshell, our sparsity-aware method produces a multiplication plan for a metapath query by the dynamic programming approach of Equation 1, while estimating sparse-matrix multiplication costs by Equation 2. We use an additional $p \times p$ matrix D to hold the densities $\hat{\rho_Z}$ of intermediate results needed to compute this cost in each step. Besides, since Atrapos caches intermediate matrix multiplication results (Section 3.4), we substitute these estimates with the (negligible) cost of retrieving a result matrix from the cache, or the cost of recomputing a result matrix if that had been previously cached and then evicted from the cache (Section 3.3).

3.3 The Overlap Tree

The hitherto discussed techniques focus on a single metapath query. We further enhance the evaluation of a metapath query workload by avoiding redundant computations that arise due to query overlaps. To do so, we introduce the *Overlap Tree*, a dynamic data structure that organizes information about frequent metapath overlaps in the workload and their dependencies. We use this data structure, along with a cache memory that stores matrix multiplication results for the indexed overlaps, to further reduce the workload evaluation time. We provide details on cache management in Section 3.4.

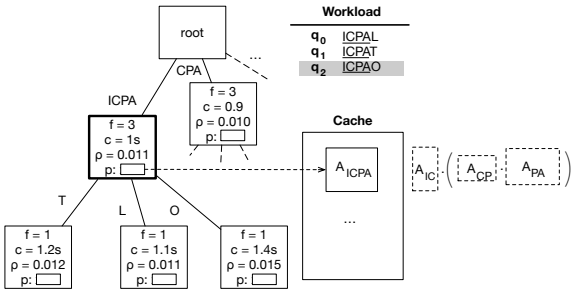


Figure 3: Example workload and Overlap Tree.

3.3.1 The structure. The Overlap Tree is a dynamic data structure that keeps track of overlaps and dependencies between the already evaluated and new queries during the evaluation of a metapath query workload. It is progressively constructed by parsing query metapaths, accompanied by a cache memory that stores matrix multiplication results. At any time, it encodes the overlaps of hitherto evaluated queries, their constraints, their occurrence frequencies, and pointers to any corresponding cache entries. We first describe the structure of the Overlap Tree ignoring query constraints; we explain how the tree considers constraints in Appendix C.

Each internal node of the tree represents an observed metapath overlap and contains the following information:

- *Frequency (f)*: The number of occurrences of the respective overlap in the query workload until the given time point.
- *Cache entry pointer (p)*: A pointer to the cache entry containing the result of the multiplication represented by the overlap. If this result is not in the cache, the pointer is null.
- *Multiplication cost (c)*: The time required to perform the matrix multiplication that corresponds to the overlap.
- *Matrix density (ρ)*: The density of the multiplication result matrix for that overlap.

In addition, the Overlap Tree contains one leaf node for each suffix of the metapaths of already evaluated queries (considering the metapath as a string); the internal structure of each leaf is identical to that of internal nodes. Each edge of the tree is labelled with a non-empty string, such that the concatenation of edge labels on a tree traversal from the root to a node gives the string that encodes the respective metapath overlap (or suffix in the case of leaves).

The Overlap Tree is inspired from the *generalised suffix tree* [6, 64], which captures the substrings of a given collection of strings; it adds pointers to cached items and information on overlap frequency, query constraints, and matrix multiplication costs. We emphasize that, as its name suggests, the Overlap tree produces a cache entry, and, for that matter, a node, only if an overlap is detected.

Figure 3 depicts an Overlap Tree for a workload of three metapath queries (q_0, q_1, and q_2), defined on the *News Articles* HIN (used in Section 4). The highlighted node represents the overlap sub-metapath *ICPA*, occurring in all three queries, points to a cache entry for the multiplication result $A_{IC} \cdot (A_{CP} \cdot A_{PA})$, and stores the density ρ of this cached result and the cost c to calculate it.

3.3.2 Update. We construct an Overlap Tree from a metapath query workload $W = \{m_1, m_2, ..., m_q\}$ by sequentially inserting into the tree all metapath queries $m_i \in W$. For each m_i of length $l_i = |m_i|$, we insert the entire m_i and all its suffixes $s_i = m_i[k : l_i]$ with $0 < k < l_i$. Starting from the root, we find the longest path that matches a prefix of s_i; the match ends either at an existing node or in the middle of an edge. In both cases, we let an internal node represent the detected overlap after the last matching character at position c, corresponding to sub-metapath $m_i[k : c]$; if the match ends on a leaf node, that node becomes internal, assigned dummy leaves if necessary. In case that occurs on an edge, we break that edge in two edges labelled with $m_i[k : c]$ and $m_i[c + 1 : l_i]$, and create a new internal node. Lastly, we create a new leaf node that corresponds to s_i and is connected to the last traversed internal node with a new edge. In this procedure, node frequencies are updated so as to reflect sub-metapath occurrences.

After this traversal, the cache pointers of the involved nodes may be updated. In particular, for the node representing the whole query m_i, the update algorithm will attempt to cache the respective multiplication result, if the node's cache item pointer is currently null; cache insertion depends on the *cache replacement policy* (Section 3.4.2). Similarly, a cache insertion will be attempted for any traversed internal node in case the respective intermediate matrix multiplication result is calculated according to the selected multiplication plan (see Sections 3.1 & 3.2). Yet even if the corresponding intermediate result is produced, it is not guaranteed to be cached, as that depends on the *cache insertion policy* (Section 3.4.1).

This process is described due to its simplicity and it is not the most efficient approach to build an Overlap Tree. Approaches based on Ukkonen's algorithm [66] are significantly faster, yet their description is beyond the scope of this work. Also note that, the worst case space complexity of the Overlap Tree is $O(\lambda)$, where λ is the total number of suffixes of all metapaths in the query workload. Due to lack of space, we elaborate more on this in Appendix B.

3.4 Atrapos-OTree Cache Management

Evaluating a metapath query workload can benefit from caching and reusing any reappearing intermediate matrix multiplication results. However, such results are unlikely to all fit in a typical machine's main memory. We thus need a cache management policy.

We exploit the Overlap Tree to obtain a two-fold benefit: first, we use the Overlap Tree to inform a *cache insertion policy* favoring the insertion of frequent multiplication results; thereby we avoid caching matrix multiplications that appear rarely, hence reduce unnecessary cache evictions (Section 3.4.1); secondly, we use the Overlap Tree to obtain information on cached items (such as computational cost to produce, size, and dependencies) for a tailored *cache replacement policy* (Section 3.4.2). In the following, we elaborate on these two uses of the Overlap Tree by Atrapos's cache management policy, the Atrapos-OTree policy.

3.4.1 Atrapos-OTree Cache Insertion Policy. In any domain of knowledge, certain informative sub-metapaths are bound to occur frequently in a query workload. Such metapath repetitions may lead to redundant matrix multiplications during the evaluation of a workload. Materialising intermediate multiplication results in cache can reduce the number of redundant computations, hence improve performance. However, as there are numerous intermediate results, we need a cache insertion policy to select which ones to insert.

The Overlap Tree is helpful to that end, as each *internal* node thereof encodes a sub-metapath occurring at least twice in the workload (Section 3.3.2). Our cache insertion policy attempts to insert a

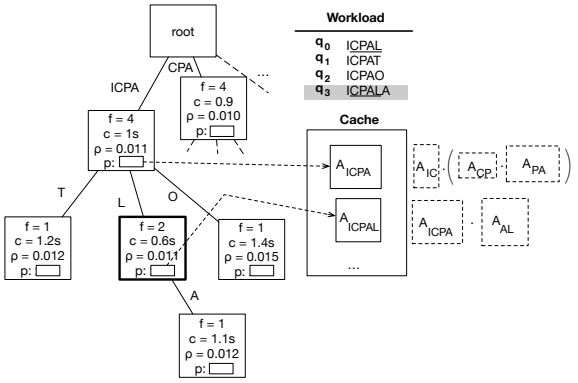


Figure 4: Overlap Tree with cache entry dependencies.

cache entry for (i) the whole of m and (ii) the longest part of m that matches an internal tree node (i.e., has occurred at least twice) *and* an intermediate result calculated by the matrix multiplication plan. Our cache replacement policy (Section 3.4.2) determines whether this attempt succeeds. Thereby, Atrapos reduces cache insertion attempts, focusing on frequent overlaps, which are likely to reoccur. Moreover, it selects to cache longest qualifying sub-metapath, which usually has the largest computational cost. For instance, in Figure 4 although *CPA* and *ICPA* are equally frequent and their intermediate results are available, only the latter is cached.

3.4.2 Atrapos-OTree Cache Replacement Policy. A straightforward way to cache matrix multiplication results is to use a generic policy, like LRU [3], which discards the *least recently used* cached item first. LRU works well when all items have similar (a) cost to fetch and (b) size. However, neither of these conditions holds for metapath query workloads, as (i) fetching an item in the cache involves matrix multiplications, whose cost differs among items, and (ii) item size varies depending on the result matrix density. In effect, one may adopt a size- and cost-aware cache replacement policy, as in web server caching [4, 24, 55]. Popularity-aware Greedy Dual-Size (PGDS) [55] measures the utility of a cache entry e as $u_e = f_e \cdot \frac{c_e}{s_e}$, where f_e is the frequency of e, c_e its cost, and s_e its size. We may obtain f_e of each cache entry from the corresponding tree counter, c_e as the estimated cost of the corresponding matrix multiplication, and s_e as its size. Let E be all cache entries. Initially, each entry $e_i \in E$ has utility value $h_{e_i} = u_{e_i}$. In case of cache saturation, we evict the entry with $\min\{h_{e_i}\}$, denoted as $H_{\min}$ and reduce the utility values of all other cache items by $H_{\min}$. Thus, recently cached items retain a higher fraction of their initial utility and are hence less likely to be evicted. To avoid subtracting $H_{\min}$ from items in the cache, PGDS uses an inflation variable L: it adds $H_{\min}$ to L at each eviction, and increases the utility of any cache hit by L.

We employ an enhanced cache policy that builds on PGDS by considering cache entry interdependence: the cost c_e to re-calculate and re-fetch an entry e in the cache may be smaller than what PGDS estimates, because part of the calculation may be obtained by another cache entry, e', as cache entries follow a hierarchy: given an internal Overlap Tree node n_p and the set of nodes in its sub-tree, $S(n_p)$, the cache entry of each $n_i \in S(n_p)$ can exploit that of n_p. Thus, each time we insert a cache entry p in an internal node, we *subtract* its cost c_p from that of cached items in its sub-tree, as these are cheaper to recompute given p. In reverse, when we evict an entry q from the cache, we *reinstate* its cost c_q to that of items in its sub-tree. Algorithm 1 in Appendix D outlines this approach.

Figure 4 shows the Overlap Tree of Figure 3 after inserting a new metapath, *ICPALA*. Assuming that the pointer p in the highlighted node for *ICPAL* was previously null, we insert a new cache item, containing the respective multiplication result, and update p to point to it. Still, since the traversal passed through the node for *ICPA*, we can use its cached result to calculate the result for *ICPAL*. In case the cached item for *ICPA* is evicted, we reinstate its cost to that of cached items in its sub-tree, i.e., to *ICPAL*, to reflect the cost of its calculation without using the entry for *ICPA*.

4 EXPERIMENTAL EVALUATION

4.1 Setup

4.1.1 Datasets. For our experiments, we use two datasets: the *Scholarly HIN* and the *News articles HIN*. The former is based on AMiner's DBLP Citation Dataset [65], enriched with topics using the CSO classifier [43] on paper abstracts, and projects funded by the EU under the H2020 programme, taken from Cordis.[3] Figure 11a in Appendix E shows its schema. The latter is based on news articles published on April 2016 and their associated entities from the GDELT Project [31], enriched with information for people involved in the Panama Papers scandal and their related companies as well as intermediaries from the Offshore Leaks database.[4] Its schema is shown in Figure 11b in Appendix E.

To assess scalability, we consider two additional splits of different sizes for the aforementioned HINs, containing 60% and 80% of their core entities, i.e. papers and articles, while preserving connected nodes and edges. Tables 2 and 3 in Appendix E show detailed statistics on the experimental HINs; these are two orders of magnitude larger than those considered in previous studies (e.g. [51]).

4.1.2 Query Workloads. Due to the lack of available real-world workloads, we create synthetic query workloads simulating the query patterns of multiple data scientists posing queries, each exploring different aspects of a certain entity via consecutive metapath queries related to it. The entity of interest is determined by an equality constraint. We refer to such a set of constrained metapath queries as a *query session*. We generate such queries by randomly choosing a metapath and a constraint applied on it. In each step, we either (a) randomly pick a new constraint and start a new session, with *session restart probability* p, or (b) continue using the same constraint and pick a different metapath, with probability $1 - p$. Indicatively, $p = 0.1$ results in sessions each containing on average 10 metapath queries. Eventually, we shuffle all queries in a workload to simulate a real-world environment. For all random selections we used a uniform or a zipfian probability distribution. Each query workload contains 500 metapath queries with length ranging from 3 to 5. We repeat each experiment with 10 different query workloads,[5] and report the average evaluation time.

4.1.3 Methods. We juxtapose the following methods:

- *HRank*: the approach of [51] in its original configuration.

[3] https://data.europa.eu/euodp/en/data/dataset/cordisH2020projects
[4] https://offshoreleaks.icij.org/pages/database
[5] https://github.com/atrapos-hin/artifacts/tree/main/workloads

Table 1: Experimental parameters.

Parameter	Range of values
Cache size (MB)	1024, 2048, **4096**, 8128, 16256
Dataset size (core entity nodes)	**60%**, 80%, 100%
Session restart probability	0.04, 0.06, **0.08**, 0.1, 0.12
Workload query distribution	**Uniform**, Zipfian

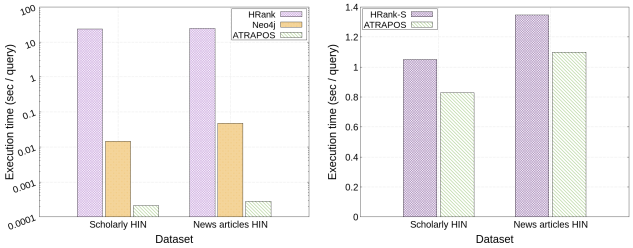
- *Neo4j* [68]: a well-established graph database. Each metapath query is expressed with Cypher,[6] using the parameters' syntax,[7] enabling effective caching of execution plans.
- *HRank Sparse (HRank-S)*: a version of HRank adjusted to exploit sparse matrix representations and the cost estimation for sparse matrix multiplications of Section 3.2.
- *Cache-based Baseline-1 (CBS1)*: an adaptation of HRank-S using an LRU cache to store *final* metapath query results.
- *Cache-based Baseline-2 (CBS2)*: an extension of *CBS1* that caches all *intermediate* matrix multiplication results.
- *Atrapos*: our approach, described in Section 3.

HRank and *Neo4j* are state-of-the-art approaches for single metapath query evaluation which we apply on query workloads. *Neo4j* has its own internal caching mechanism. As discussed in Section 4.2, these approaches perform poorly even for small datasets and do not scale to large HINs. Therefore, we also use the three baselines (*HRank-S*, *CBS1*, and *CBS2*), which take advantage of sparse matrix representations and simple caching mechanisms. Note that, all items evicted from the cache could, in theory, be stored on disk, acting like a second level cache; however, this applies to all cache-based methods and, hence, does not impact their relative comparison.

4.1.4 Evaluation Setting. We implemented[8] all methods in C++, using the Eigen Library [21] for matrix multiplications; we opted for the compressed sparse column layout (CSC), which stores non-zero values and their positions in a column-major order. Source code was compiled with the GNU Compiler (G++ v9.3.0) using O3 level optimisations and the NDEBUG preprocessor directive. All cache-based implementations are set to never store items that exceed a size threshold equal to 80% of the available cache size. This rule safeguards against evicting many useful results to store just a single large one (indicatively, this rule applied in less than 0.1% of the queries in both datasets considering the default setup).

All experiments ran on an AMD 3960X CPU machine @ 3.50GHz with 256GB RAM on Ubuntu 20.04 LTS. All approaches were able to utilise all available resources. Specifically, since Neo4j (4.2.5 CE) stores data on disk, it was configured to allocate 31GB of JVM heap size and 205GB of page size,[9] allowing it to load most of the data into main memory. The remaining memory can be allocated to Neo4j from the OS, hence all 256GB of memory could be utilised in total (alternative configurations had no significant impact on performance). In addition, Atrapos and its competitors load data from CSV files into the main memory; this ingestion time is not included in the reported query execution time. Table 1 lists the parameters we vary in our experiments, with default values in bold.

[6]Neo4j's native query language: https://neo4j.com/developer/cypher-query-language/
[7]https://neo4j.com/docs/cypher-manual/current/syntax/parameters/
[8]Code available at https://github.com/atrapos-hin/artifacts
[9]As indicated by Neo4j's memory recommendations.



(a) Vs. HRank & Neo4j (dataset subsets, log scale). **(b) Vs. HRank-S (full dataset sizes, linear scale)**

Figure 5: Evaluation against single-query methods.

4.2 Evaluation against single-query methods

First, we assess Atrapos against the dense-matrix HRank and Neo4j, using subsets of the Scholarly HIN (37k nodes, 120k edges) and the News articles HIN (37k nodes, 210k edges), as HRank and Neo4j cannot handle the full datasets due to memory requirements.[10] Figure 5a presents our results on execution time per query in logarithmic scale. Atrapos surpasses contestants on both datasets. Neo4j is over one order of magnitude slower than Atrapos, while HRank is over two orders of magnitude slower than Neo4j.

Then, we examine Atrapos against HRank-S (HRank's variant with sparse matrices), using full size HINs. Both approaches use the same sparse matrix representations and multiplication cost estimation (Section 3.2). However, Atrapos leverages metapath query overlaps using a 4GB cache. Figure 5b shows the execution time per query. Atrapos outperforms HRank-S achieving a speedup of 27% in the Scholarly HIN and 23% in the News articles HIN.

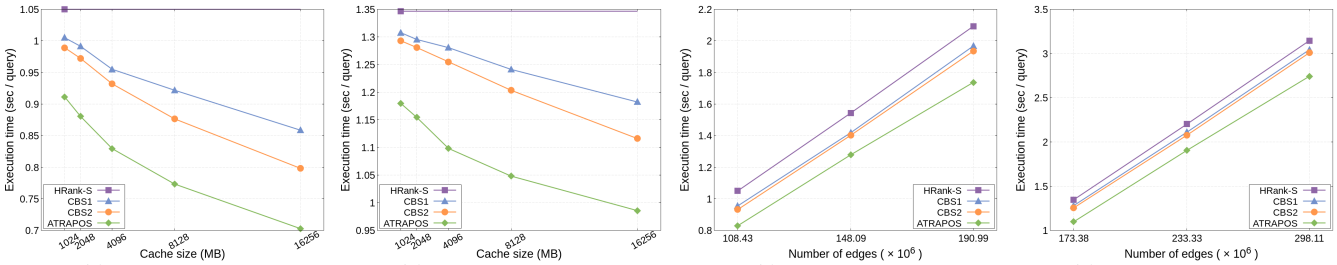
4.3 Evaluation against baseline caching

While no existing approach exploits a caching mechanism to accelerate the evaluation of a metapath query workload, evaluating Atrapos solely against non-caching approaches cannot be fair. Thus, we introduce CBS1 and CBS2, two HRank-S variants that employ basic caching mechanisms (Section 4.1) and assess Atrapos against them. We also include HRank-S for reference.

4.3.1 Varying cache size. Figures 6a-b present the average execution time per query vs. cache size. As HRank-S is oblivious to caching, its execution time is independent of cache size. All cache-based approaches outperform HRank-S, while their advantage grows with cache size. CBS1 is faster than HRank-S by virtue of exploiting repetitive metapath queries; as it caches only query results, it is slower than CBS2 and Atrapos. Atrapos outperforms other contestants in both datasets, as it leverages the Overlap Tree to identify frequent metapath overlaps and select matrix multiplication results for caching. Its cache management policy refrains from inserting all intermediate results into the cache, therefore its advantage over CBS2 is accentuated in small cache sizes ($2-4$ GB).

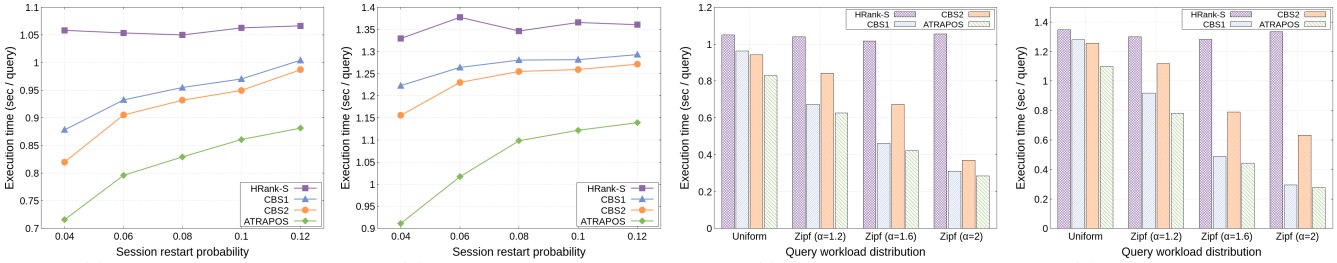
4.3.2 Varying dataset size. Figures 6c-d illustrate the average execution time per query vs. network size. The x axis contains the number of edges (in millions) for each dataset split (Section 4.1.1). All methods scale linearly with the number of edges. All three cache-based approaches outperform HRank-S, with the difference being

[10]Neo4j is a fully fledged system (e.g, supporting ACID transactions) so additional overheads both in terms of query execution times and memory usage are expected.



(a) Scholarly HIN (b) News articles HIN (c) Scholarly HIN (d) News articles HIN

Figure 6: Evaluation against baseline caching approaches with varying cache and dataset size.



(a) Scholarly HIN (b) News articles HIN (c) Scholarly HIN (d) News articles HIN

Figure 7: Evaluation against baseline caching approaches varying session restart probability and Zipf distribution of queries.

more prominent in the Scholarly HIN. ATRAPOS clearly outperforms CBS1 and CBS2 in all dataset sizes.

4.3.3 Varying session restart probability p. Figures 7a-b show the improvement in execution time over HRank-S for CBS1, CBS2 and ATRAPOS as the session restart probability p falls. As p grows, the performance of all cache-based methods degrades, as there are fewer queries per session on average, hence fewer overlaps to exploit. Overall, ATRAPOS outperforms its competitors. Among the rival methods, CBS2 surpasses CBS1, as it exploits intermediate results.

4.3.4 Zipfian query workloads. Figures 7c-d show execution times per query as we vary the distribution from which we select metapaths and constraints. Apart from the uniform distribution, used in other experiments, we consider Zipfian distributions varying the scaling parameter α. HRank-S achieves comparable execution times in all settings as it does not take advantage of repetitions, while cache-based approaches improve as α grows. As α rises, the probability of queries being repeated also increases, while smaller values indicate a heavier tail in the probability distribution. ATRAPOS outperforms competitors in all scenarios: it outperforms CBS1 by exploiting query overlaps on top of the result that CBS1 caches. The difference is more notable with $\alpha = 1.2$ and $\alpha = 1.6$, yet ATRAPOS surpasses CBS1 even with $\alpha = 2$, whereby a few queries are repeated in the workload. CBS2 exhibits significantly larger execution times with Zipfian distributions, as caching intermediate results bars it from exploiting all query repetitions. Appendix F shows times per query against baseline caching as the workload evolves.

4.4 Effect of cache replacement policies

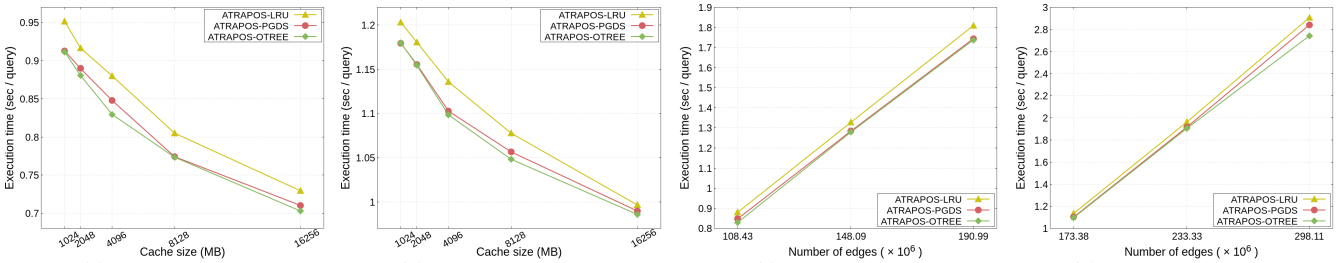
We now benchmark our cache replacement policy, OTREE (Section 3.4), against the Least Recently Used (LRU) [3] and Popularity-aware Greedy Dual-Size (PGDS) [55]. We examine performance varying cache size (Section 4.4.1), dataset size (Section 4.4.2), session restart probability p (Section 4.4.3) and distributions for selecting queries (Section 4.4.4). All methods utilise the Overlap Tree, hence they differ solely in the underlying cache replacement policy.

4.4.1 Varying cache size. Figures 8a-b shows our findings with varying cache size. Standard ATRAPOS outperforms PGDS, the best performer among the rest on most scenarios; its advantage is more prominent on Scholarly HIN. Yet, ATRAPOS is marginally faster than the PGDS policy on News articles HIN; the most notable difference appears with cache size equal to 4GB; at the same time, both cache policies incorporating frequency and item size outperform LRU.

4.4.2 Varying dataset size. Figures 8c-d present our results for all policies as we vary the dataset size. We observe a linear increase in execution time per query for all approaches as dataset size grows. LRU performs poorly compared to the other approaches in both datasets. ATRAPOS is faster than PGDS in most examined settings; it is marginally faster in the Scholarly HIN, with performance gains being more apparent when using the full News articles HIN.

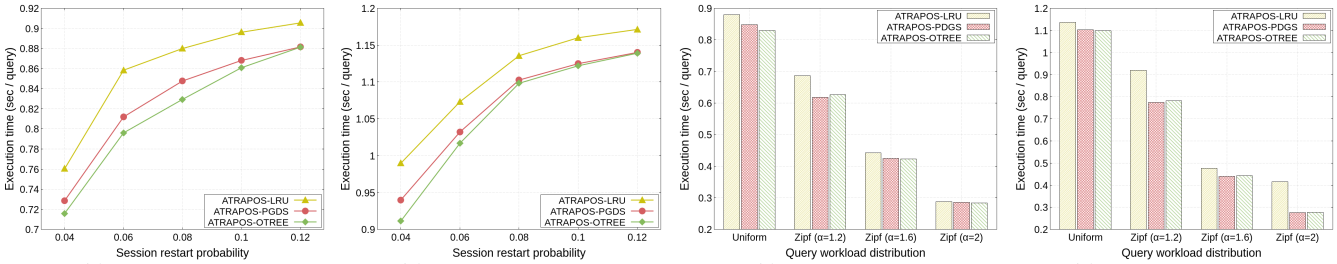
4.4.3 Varying session restart probability p. Figures 9a-b present performance when varying the session restart probability p. The speedup falls for all methods as p grows, which is reasonable, as a larger p results in sessions containing fewer queries and thus fewer overlaps to exploit. ATRAPOS attains larger gains than other contestants in most examined configurations. More noteworthy differences appear on the Scholarly HIN, especially for p equal to 0.04, 0.06 and 0.08. In the News articles HIN, ATRAPOS is faster for small values of p i.e. 0.04 and 0.06, while PGDS achieves comparable results with ATRAPOS for larger p. Both of them outperform LRU, with significant differences in both datasets.

4.4.4 Zipfian query workloads. Figures 9c-d illustrate the performance of the examined cache replacement policies using a Zipfian distribution to generate workloads. all approaches achieve performance gains with workloads generated with the Zipfian distribution compared to the uniform. Moreover, as parameter α of the Zipfian



(a) Scholarly HIN (b) News articles HIN (c) Scholarly HIN (d) News articles HIN

Figure 8: Evaluation against cache replacement policies with varying cache and dataset size.



(a) Scholarly HIN (b) News articles HIN (c) Scholarly HIN (d) News articles HIN

Figure 9: Evaluation against cache replacement policies varying session restart probability and Zipf distribution of queries.

distribution grows, gains become more prominent, as larger α indicates a higher probability for query repetitions. PGDS and ATRAPOS achieve comparable results, with ATRAPOS being marginally faster with the uniform distribution and the Zipfian with α equal to 1.6 and 2 on the Scholarly HIN. Still, both approaches outperform LRU on both datasets. Since all hereby considered approaches utilise the Overlap Tree, differing only in the cache replacement policy, their difference in performance is small, as expected. Appendix F reports results on execution time for individual queries.

5 RELATED WORK

HIN analysis. Several works rank HIN nodes based on their centrality in a metapath-defined network [30, 33, 37]. HRank [51] co-ranks HIN nodes and relations based on the intuition that important nodes are connected to other nodes through important relations. Metapath-based similarity measures among HIN nodes are also used for similarity search and join [49, 50, 60, 70]. Recent works incorporate metapath-based connectivity in link prediction and recommendation [7, 12, 48]. Furthermore, several community detection algorithms have been proposed for HINs; some approaches detect clusters that contain nodes of different types [53, 58, 63], while others generate homogeneous clusters [61, 62, 74]. [17] detects the community of a given node in a HIN extending traditional metrics for community cohesiveness to take metapaths into acccount.

Metapath query evaluation. A recent line of work focuses on metapath query evaluation. Shi et al. [51] apply dynamic programming [15] to reorganize the order of matrix multiplications required for metapath query evaluation, thereby reducing computational cost. When small errors can be tolerated, truncation strategies [9, 29] can be applied, zeroing values below a certain threshold. Another approach follows a Monte Carlo strategy [34] to approximate the query result via random walkers; the number of which strikes a balance in the tradeoff between accuracy and computation cost. Unlike ATRAPOS, these techniques focus on the performance of single-query evaluation, incognizant of data exploration workloads with naturally emerging overlapping queries. Sun et al. [60] propose the offline materialisation of short metapaths in a preprocessing stage and concatenate them at query time to make the online evaluation of a single query more efficient. Lastly, similar techniques have also been proposed for *graph pattern queries* in RDF-triple stores [20, 35, 39].

Cache replacement policies. There is a vast literature on cache replacement, especially in web server caching [5, 40]. Much of it follows the Least Recently Used (LRU) policy [3]; LRU-MIN [1] favors retaining smaller items in cache, while LRU-S [57] incorporates the size of cached items into a randomised caching mechanism. Size-adjusted LRU (SLRU)[2], considers both size and cost. Greedy Dual-based approaches [8, 72] form another class of cache replacement policies: Popularity-aware GreedyDual-Size (PGDS) [55] utilises the frequency, size, and cost of cached items. GreedyDual* (GD*) [24] adaptively adjusts the significance of *recency* and *frequency* information over time. Other policies take into account cached item size and frequency [4, 19, 41, 54]. Yet none of the aforementioned policies considers the *interdependence* among cached items that inevitably appears in a metapath query workload, as ATRAPOS does.

6 CONCLUSIONS

We introduced ATRAPOS, a method to evaluate metapath query workloads in HINs exploiting query overlaps. ATRAPOS accelerates computations using sparse-matrix multiplication along with an appropriate cost model. Furthermore, it introduces an overlap-aware cache replacement policy leveraging a tailor-made data structure, the *Overlap Tree*, that identifies overlapping query metapaths. Our thorough experimental evaluation on two large real-world HINs shows that ATRAPOS outperforms existing approaches in efficiency across examined scenarios. In the future, we aim to investigate metapath query evaluation over compressed network representations conforming to space [25, 28] and privacy [38] constraints.

ACKNOWLEDGMENTS

This work was partially funded by the EU H2020 project SmartDataLake (GA: 825041) and the EU Horizon Europe projects SciLake (GA: 101058573) and STELAR (GA: 101070122).

REFERENCES

[1] Marc Abrams, Charles R Standridge, Ghaleb Abdulla, Stephen Williams, and Edward A Fox. 1995. *Caching proxies: Limitations and potentials.* Technical Report. Department of Computer Science, Virginia Polytechnic Institute & State.
[2] Charu Aggarwal, Joel L Wolf, and Philip S. Yu. 1999. Caching on the world wide web. *TKDE* 11, 1 (1999), 94–107.
[3] Alfred V Aho, Peter J Denning, and Jeffrey D Ullman. 1971. Principles of optimal page replacement. *JACM* 18, 1 (1971), 80–93.
[4] Hyokyung Bahn, Kern Koh, Sam H Noh, and SM Lyul. 2002. Efficient replacement of nonuniform objects in web caches. *Computer* 35, 6 (2002), 65–73.
[5] Abdullah Balamash and Marwan Krunz. 2004. An overview of web caching replacement algorithms. *IEEE Communications Surveys & Tutorials* 6, 2 (2004), 44–56.
[6] Paul Bieganski, John Riedl, John V Carlis, and Ernest F Retzel. 1994. Generalized suffix trees for biological sequence data: applications and implementation. In *HICSS*. 35–44.
[7] Bokai Cao, Xiangnan Kong, and Philip S. Yu. 2014. Collective Prediction of Multiple Types of Links in Heterogeneous Information Networks. In *ICDM*. 50–59.
[8] Pei Cao and Sandy Irani. 1997. Cost-aware www proxy caching algorithms.. In *Usenix*, Vol. 12. 193–206.
[9] Soumen Chakrabarti. 2007. Dynamic personalized pagerank in entity-relation graphs. In *WWW*. 571–580.
[10] Serafeim Chatzopoulos, Kostas Patroumpas, Alexandros Zeakis, Thanasis Vergoulis, and Dimitrios Skoutas. 2020. SPHINX: a system for metapath-based entity exploration in heterogeneous information networks. *VLDB* 13, 12 (2020), 2913–2916.
[11] Serafeim Chatzopoulos, Thanasis Vergoulis, Panagiotis Deligiannis, Dimitrios Skoutas, Theodore Dalamagas, and Christos Tryfonopoulos. 2021. SciNeM: A Scalable Data Science Tool for Heterogeneous Network Mining.. In *EDBT*. 654–657.
[12] Hongxu Chen, Hongzhi Yin, Weiqing Wang, Hao Wang, Quoc Viet Hung Nguyen, and Xue Li. 2018. PME: Projected Metric Embedding on Heterogeneous Networks for Link Prediction. In *SIGKDD*. 1177–1186.
[13] Hsi-Wen Chen, Hong-Han Shuai, De-Nian Yang, Wang-Chien Lee, Chuan Shi, S Yu Philip, and Ming-Syan Chen. 2021. Structure-Aware Parameter-Free Group Query via Heterogeneous Information Network Transformer. In *ICDE*. 2075–2080.
[14] Edith Cohen. 1998. Structure Prediction and Computation of Sparse Matrix Products. *Journal of Combinatorial Optimization* 2, 4 (1998), 307–332.
[15] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, 3rd Edition.* MIT Press. http://mitpress.mit.edu/books/introduction-algorithms
[16] Yixiang Fang, Kai Wang, Xuemin Lin, and Wenjie Zhang. 2021. Cohesive Subgraph Search over Big Heterogeneous Information Networks: Applications, Challenges, and Solutions. In *SIGMOD*. 2829–2838.
[17] Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, and Xin Cao. 2020. Effective and efficient community search over large heterogeneous information networks. *VLDB* 13, 6 (2020), 854–867.
[18] Sheldon Finkelstein. 1982. Common expression analysis in database applications. In *SIGMOD*. 235–245.
[19] Annie P Foong, Yu-Hen Hu, and Dennis M Heisey. 1999. Adaptive web caching using logistic regression. In *SSP*. 515–524.
[20] Sidan Gao and Kemafor Anyanwu. 2013. PrefixSolve: efficiently solving multi-source multi-destination path queries on RDF graphs by sharing suffix computations. In *WWW*. 423–434.
[21] Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.
[22] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al. 2021. Knowledge graphs. *ACM Computing Surveys (CSUR)* 54, 4 (2021), 1–37.
[23] Xun Jian, Yue Wang, and Lei Chen. 2020. Effective and efficient relational community detection and search in large dynamic heterogeneous information networks. *VLDB* 13, 10 (2020), 1723–1736.
[24] Shudong Jin and Azer Bestavros. 2001. GreedyDual* Web caching algorithm: exploiting the two sources of temporal locality in Web request streams. *Computer Communications* 24, 2 (2001), 174–183.
[25] Panagiotis Karras and Nikos Mamoulis. 2008. Hierarchical synopses with optimal error guarantees. *ACM Trans. Database Syst.* 33, 3 (2008), 18:1–18:53.
[26] Tarun Kathuria and S Sudarshan. 2017. Efficient and provable multi-query optimization. In *Symposium on Principles of Database Systems (ACM SIGMOD-SIGACT-SIGAI)*. 53–67.
[27] David Kernert, Frank Köhler, and Wolfgang Lehner. 2015. SpMacho - Optimizing Sparse Linear Algebra Expressions with Probabilistic Density Estimation. In *EDBT*.
[28] K. Ashwin Kumar and Petros Efstathopoulos. 2018. Utility-Driven Graph Summarization. *Proc. VLDB Endow.* 12, 4 (2018), 335–347.
[29] Ni Lao and William W Cohen. 2010. Fast query execution for retrieval models based on path-constrained random walks. In *SIGKDD*. 881–888.
[30] Sangkeun Lee, Sungchan Park, Minsuk Kahng, and Sang Goo Lee. 2012. PathRank: a novel node ranking measure on a heterogeneous graph for recommender systems. *CIKM*.
[31] Kalev Leetaru and Philip A Schrodt. 2013. Gdelt: Global data on events, location, and tone, 1979–2012. In *ISA annual convention*, Vol. 2. Citeseer, 1–49.
[32] Xiang Li, Danhao Ding, Ben Kao, Yizhou Sun, and Nikos Mamoulis. 2021. Leveraging Meta-path Contexts for Classification in Heterogeneous Information Networks. In *ICDE*. 912–923.
[33] Yitong Li, Chuan Shi, S Yu Philip, and Qing Chen. 2014. Hrank: a path based ranking method in heterogeneous information network. In *WAIM*. 553–565.
[34] Jun S Liu. 2008. *Monte Carlo strategies in scientific computing.* Springer Science & Business Media.
[35] Amgad Madkour, Ahmed M Aly, and Walid G Aref. 2018. Worq: Workload-driven rdf query processing. In *International Semantic Web Conference (ISWC)*. 583–599.
[36] Changping Meng, Reynold Cheng, Silviu Maniu, Pierre Senellart, and Wangda Zhang. 2015. Discovering Meta-Paths in Large Heterogeneous Information Networks. In *WWW*. 754–764.
[37] Michaek Kwok-Po Ng, Xutao Li, and Yunming Ye. 2011. Multirank: co-ranking for objects and relations in multi-relational data. In *SIGKDD*. 1217–1225.
[38] Sadegh Nobari, Panagiotis Karras, HweeHwa Pang, and Stéphane Bressan. 2014. L-opacity: Linkage-Aware Graph Anonymization. In *EDBT*. 583–594.
[39] Nikolaos Papailiou, Dimitrios Tsoumakos, Panagiotis Karras, and Nectarios Koziris. 2015. Graph-aware, workload-adaptive SPARQL query caching. In *SIGMOD*. 1777–1792.
[40] Stefan Podlipnig and Laszlo Böszörmenyi. 2003. A survey of web cache replacement strategies. *CSUR* 35, 4 (2003), 374–398.
[41] Luigi Rizzo and Lorenzo Vicisano. 2000. Replacement policies for a proxy cache. *IEEE/ACM Transactions on Networking* 8, 2 (2000), 158–170.
[42] Prasan Roy, Srinivasan Seshadri, S Sudarshan, and Siddhesh Bhobe. 2000. Efficient and extensible algorithms for multi query optimization. In *SIGMOD*. 249–260.
[43] Angelo Salatino, Francesco Osborne, and Enrico Motta. 2022. CSO Classifier 3.0: a scalable unsupervised method for classifying documents in terms of research topics. *International Journal on Digital Libraries* 23, 1 (2022), 91–110.
[44] Timos Sellis and Subrata Ghosh. 1990. On the multiple-query optimization problem. *Transactions on Knowledge and Data Engineering (TKDE)* 2, 02 (1990), 262–266.
[45] Timos K Sellis. 1988. Multiple-query optimization. *TODS* 13, 1 (1988), 23–52.
[46] Wei Shen, Jiawei Han, and Jianyong Wang. 2014. A probabilistic model for linking named entities in web text with heterogeneous information networks. In *SIGMOD*. 1199–1210.
[47] Baoxu Shi and Tim Weninger. 2014. Mining Interesting Meta-Paths from Complex Heterogeneous Information Networks. In *ICDM*. 488–495.
[48] Chuan Shi, Binbin Hu, Wayne Xin Zhao, and Philip S. Yu. 2019. Heterogeneous Information Network Embedding for Recommendation. *TKDE* 31, 2 (2019), 357–370.
[49] Chuan Shi, Xiangnan Kong, Yue Huang, S Yu Philip, and Bin Wu. 2014. Hetesim: A general framework for relevance measure in heterogeneous networks. *TKDE* 26, 10 (2014), 2479–2492.
[50] Chuan Shi, Xiangnan Kong, Philip S Yu, Sihong Xie, and Bin Wu. 2012. Relevance search in heterogeneous networks. In *EDBT*. 180–191.
[51] Chuan Shi, Yitong Li, Philip S. Yu, and Bin Wu. 2016. Constrained-meta-path-based ranking in heterogeneous information network. *Knowledge and Information Systems* 49, 2 (2016), 719–747.
[52] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and S Yu Philip. 2016. A survey of heterogeneous information network analysis. *TKDE* 29, 1 (2016), 17–37.
[53] Chuan Shi, Ran Wang, Yitong Li, Philip S Yu, and Bin Wu. 2014. Ranking-based clustering on general heterogeneous information networks by network projection. In *CIKM*. 699–708.
[54] Junho Shim, Peter Scheuermann, and Radek Vingralek. 1999. Proxy cache algorithms: Design, implementation, and performance. *TKDE* 11, 4 (1999), 549–562.
[55] Shudong Jin and A. Bestavros. 2000. Popularity-aware greedy dual-size Web proxy caching algorithms. In *ICDCS*. 254–261.
[56] Johanna Sommer, Matthias Boehm, Alexandre V. Evfimievski, Berthold Reinwald, and Peter J. Haas. 2019. MNC: Structure-Exploiting Sparsity Estimation for Matrix Expressions. In *SIGMOD*. 1607–1623.
[57] David Starobinski and David Tse. 2001. Probabilistic methods for web caching. *Performance Evaluation* 46, 2-3 (2001), 125–137.
[58] Yizhou Sun, Charu C Aggarwal, and Jiawei Han. 2012. Relation strength-aware clustering of heterogeneous information networks with incomplete attributes. *arXiv preprint arXiv:1201.6563* (2012).

[59] Yizhou Sun, Jiawei Han, Xifeng Yan, and Philip S Yu. 2012. Mining knowledge from interconnected data: a heterogeneous information network analysis approach. *VLDB* 5, 12 (2012), 2022–2023.
[60] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. 2011. PathSim: Meta Path-Based Top-k Similarity Search in Heterogeneous Information Networks. *PVLDB* 4, 11 (2011), 992–1003.
[61] Yizhou Sun, Jiawei Han, Peixiang Zhao, Zhijun Yin, Hong Cheng, and Tianyi Wu. 2009. Rankclus: integrating clustering with ranking for heterogeneous information network analysis. In *EDBT*. 565–576.
[62] Yizhou Sun, Brandon Norick, Jiawei Han, Xifeng Yan, Philip S Yu, and Xiao Yu. 2013. Pathselclus: Integrating meta-path selection with user-guided object clustering in heterogeneous information networks. *TKDD* 7, 3 (2013), 1–23.
[63] Yizhou Sun, Yintao Yu, and Jiawei Han. 2009. Ranking-based clustering of heterogeneous information networks with star network schema. In *SIGKDD*. 797–806.
[64] Wojciech Szpankowski. 1993. A generalized suffix tree and its (un) expected asymptotic behaviors. *SICOMP* 22, 6 (1993), 1176–1198.
[65] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. ArnetMiner: Extraction and Mining of Academic Social Networks. In *SIGKDD*. 990–998.
[66] Esko Ukkonen. 1995. On-line construction of suffix trees. *Algorithmica* 14, 3 (1995), 249–260.
[67] Yue Wang, Zhe Wang, Ziyuan Zhao, Zijian Li, Xun Jian, Lei Chen, and Jianchun Song. 2020. HowSim: A General and Effective Similarity Measure on Heterogeneous Information Networks. In *ICDE*. 1954–1957.
[68] Jim Webber. 2012. A programmatic introduction to neo4j. In *SPLASH*. 217–218.
[69] Tao Xie, Yangjun Xu, Liang Chen, Yang Liu, and Zibin Zheng. 2021. Sequential Recommendation on Dynamic Heterogeneous Information Network. In *ICDE*. 2105–2110.
[70] P. Yu Y. Xiong, Y. Zhu. 2014. Top-k Similarity Join in Heterogeneous Information Networks. *TKDE* 27, 6 (2014), 1710–1723.
[71] Yixing Yang, Yixiang Fang, Xuemin Lin, and Wenjie Zhang. 2020. Effective and efficient truss computation over large heterogeneous information networks. In *ICDE*. 901–912.
[72] Neal Young. 1994. Thek-server dual and loose competitiveness for paging. *Algorithmica* 11, 6 (1994), 525–541.
[73] Yongyang Yu, Mingjie Tang, Walid G. Aref, Qutaibah M. Malluhi, Mostafa Abbas, and Mourad Ouzzani. 2017. In-memory distributed matrix computation processing & optimization. In *ICDE*. 1047–1058.
[74] Yang Zhou and Ling Liu. 2013. Social influence based clustering of heterogeneous information networks. In *SIGKDD*. 338–346.

A BENEFIT OF COST ESTIMATION

We found experimentally that cost estimation techniques do not provide significant benefits in our application scenario. Figure 10 illustrates the execution time breakdown of E_{ac} against the MNC estimator [56], which builds sketches to estimate the sparsity of the result. This experiment considers 2 000 randomly selected metapath queries with their length varying from 3 to 5 for the Scholarly and News article HINs (see Sections 4.1.1 and 4.1.2). Both estimators achieve very comparable results for the actual matrix multiplication execution, with E_{ac} requiring a negligible amount of time for planning compared to MNC. Interestingly, both estimators produce the same multiplication plans for over 70% of the examined queries. As we consider constrained metapath queries, very sparse matrices arise in the multiplication, and the plans produced by E_{ac} are on par with those of more sophisticated solutions without the added cost of planning. Since the added cost of more accurate estimators does not pay off in terms of cost savings for the query types we examine, we settled for the former best-effort estimation.

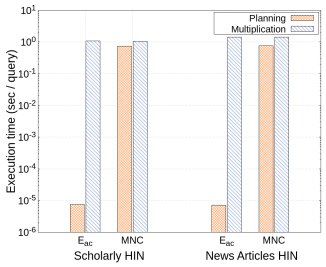


Figure 10: MNC vs the average-case density estimator E_{ac} (log scale on y-axis).

B SPACE COMPLEXITY

Given a metapath query workload $W = \{m_1, m_2, ..., m_q\}$ and the respective Overlap Tree T_W. The exact number of leaves in T_W is $\lambda = |m_1| + |m_2| + ... + |m_q|$, since each $m_i \in W$ has exactly $|m_i|$ suffixes. As an internal tree node is created when and only when a query overlap is detected, each internal node has at least two children; in the worst case, all internal nodes have exactly two children, hence T_W is a full binary tree having $2\lambda - 1 = O(\lambda)$ nodes.

Based on the internal structure of the Overlap Tree nodes, all leaves have the same memory footprint: the aggregated size of integer f, two floating point numbers, c and ρ, and one pointer p. All internal nodes, on the other hand, have the same memory footprint plus the size of the pointers to their children. Since in the worst case each internal node has two such pointers, the respective worst-case size can be easily calculated. Finally, the root has a memory footprint equal to that of two pointers, since none of the four node variables (f, c, ρ, p) are included in this node.

It is worth to highlight that for most practical scenarios, like those examined in our experiments, the size of the Overlap Tree is negligible compared to the size of the available main memory of a contemporary server. This means that the structure allows for a large size of cache to be determined, benefiting the performance of the workload evaluation process.

C HANDLING CONSTRAINED QUERIES

In Section 3.3, we have described Overlap Tree operations without considering query constraints (see Definition 3 for details on constrained metapaths). To support constraints, we modify the internal structure of the tree nodes enriching them with a key-value data structure, called the *constraints index*, that holds a given constraint in the form of a string (e.g., $P.year > 2000$) as key and the respective variables (f, p, c, and ρ) as a composite value. The constraint index of each node can be implemented as a suitable data structure, such as a hash table index. We adapt the respective operations accordingly to accommodate this change.

Specifically, upon probing the Overlap Tree for a constrained pattern, we traverse the tree following the exact same process described in Section 3.3.2 until we reach the ending node. Then, we probe the constraints index of the node to retrieve the respective entry (or to create it, if there is not one) and we update the corresponding constraints index counters accordingly; if we select to cache the result, the cache entry pointer should also be updated.

Regarding space complexity, the calculations in Section B remain valid, yet we should also take into consideration the number of constraints each node accommodates in the constraint index. In the worst case, each overlapping metapath query has a different constraint, hence the number of key-value pairs in a node equals the node's frequency f.

D CACHE REPLACEMENT POLICY

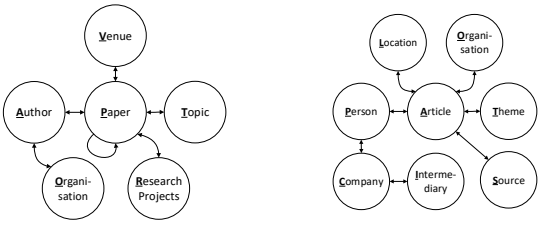
Algorithm 1 summarises Atrapos' tailored cache replacement policy, Atrapos-OTree policy, which we elaborated on Section 3.4.2, that exploits cache item interdependence.

Algorithm 1: Atrapos-OTree cache replacement policy

```
1  L = 0
2  foreach request of cache item p do
3      f_p = f_p + 1
4      if p is in cache then
5          p_l = L
6          h_p = f_p · c_p/s_p + p_l
7      else
8          while not enough space for p in cache do
9              L = min{h_q | q ∈ cache}
10             evict q that satisfies h_q = L
11             foreach cache entry e in the subtree of q do
12                 c_e = c_e + c_q
13                 h_e = f_e · c_e/s_e + e_l
14         insert p in cache
15         p_l = L
16         h_p = f_p · c_p/s_p + p_l
17         foreach cache entry e in the subtree of p do
18             c_e = c_e − c_p
19             h_e = f_e · c_e/s_e + e_l
```

E DETAILS OF EXPERIMENTAL DATASETS

In our experiments, we used two experimental HINs: Scholarly HIN and News articles HIN. Figures 11a and 11b show their schema definitions, respectively. In addition, Table 2 shows some basic statistics on our experimental datasets, i.e., number of nodes/edges and average degree. while Table 3 collects the number of nodes and edges per type for each dataset split considered. Note that, we consider bi-directional relationships between entities.



(a) Scholarly HIN (b) News articles HIN

Figure 11: Schema definitions of experimental HINs.

Table 2: Details of experimental HINs.

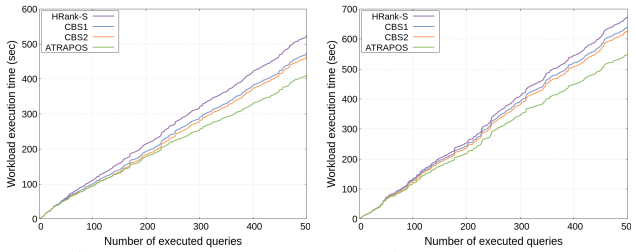
Dataset	Contained Core Entity Nodes	Nodes (in millions)	Edges (in millions)	Avg. Degree
Scholarly HIN	60%	8.65	108.43	25.06
	80%	10.44	148.09	28.36
	100%	12.15	190.99	31.43
News articles HIN	60%	7.71	173.38	44.92
	80%	10.06	233.33	46.37
	100%	12.43	298.11	47.95

Table 3: Number of nodes/edges per type.

	Node Type	Million nodes per split (60% / 80% / 100%)	Edge Type	Million edges per split (60% / 80% / 100%)
Scholarly HIN	P	2.936/3.915/4.894	PP	16.367/29.236/455.641
	A	3.100/3.781/4.398	AP/PA	17.916/23.898/29.869
	O	2.488/2.608/2.706	OA/AO	14.921/15.976/16.832
	V	0.009/0.009/0.010	VP/PV	5.247/6.994/8.743
	T	0.117/0.125/0.132	TP/PT	53.977/71.974/89.974
	R	0.001/0.001/0.001	RP/PR	0.008/0.010/0.013
News articles HIN	A	4.394/5.859/7.324	IC/CI	0.010/0.010/0.010
	O	1.153/1.486/1.829	OA/AO	25.786/35.186/45.584
	P	1.953/2.465/2.995	PA/AP	32.902/44.334/57.125
	L	0.172/0.202/0.229	LA/AL	31.767/42.985/55.319
	T	0.014/0.016/0.17	TA/AT	73.184/97.618/122.750
	S	0.022/0.025/0.030	SA/AS	9.714/13.185/17.306
	C	0.005/0.005/0.005	CP/PC	0.014/0.014/0.014
	I	0.001/0.001/0.001		

F EXECUTION TIME PER QUERY

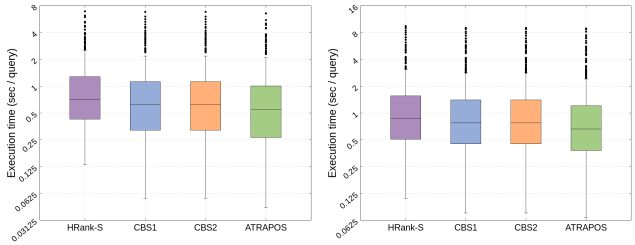
In our experiments, we presented the average execution time per query across all query workloads. Here, we investigate the performance of individual queries, averaging the reported execution times by query position across 10 workflows. Figure 12 shows the cumulative time during workload execution; cache-based approaches are noticeably faster than HRank-S, especially for the Scholarly HIN. ATRAPOS is also considerably faster than CBS1 and CBS2 with the difference increasing with the number of queries.



(a) Scholarly HIN (b) News articles HIN

Figure 12: Evaluation against baseline caching approaches considering cumulative time per query.

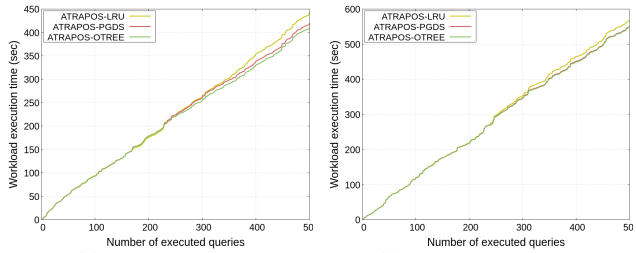
Then, we investigate the skewness of the query execution times for the same workloads. Specifically, Figure 13 illustrates the execution time per query; it reconfirms previous findings as each quartile of ATRAPOS starts lower than the respective ones of the competitor approaches in both box plots.



(a) Scholarly HIN (b) News articles HIN

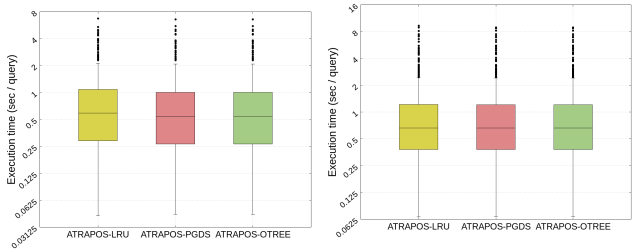
Figure 13: Evaluation against baseline caching approaches considering query execution time skewness.

Figure 14 shows the cumulative times for the different cache replacement policies during workload execution. ATRAPOS and PGDS achieve comparable times, yet ATRAPOS is slightly faster on the Scholarly HIN, as evident in Figure 14a. LRU is slower than its competitors in both datasets; the difference is visible in the second half of both query workload executions. Figure 15 plots the execution time of each metapath query for the same workloads. It reconfirms that PGDS and ATRAPOS are faster than the LRU as each quartile in their box plots starts lower than the respective quartile for LRU for the Scholarly HIN.



(a) Scholarly HIN (b) News articles HIN

Figure 14: Evaluation against cache replacement policies considering cumulative time per query.



(a) Scholarly HIN (b) News articles HIN

Figure 15: Evaluation against cache replacement policies considering query execution time skewness.