iSummary: Workload-based Selective Summaries for Knowledge Graph Exploration

Giannis Vassiliou [a], Nikolaos Papadakis [a] and Haridimos Kondylakis [b,*]
[a] *Electrical and Computer Engineering, Hellenic Mediterranean University, Heraklion, Greece*
E-mails: giannisvas@ics.forth.gr, npapadak@cs.hmu.gr
[b] *Computer Science Department, University of Crete & FORTH-ICS, Heraklion, Greece*
E-mail: kondylak@ics.forth.gr

Abstract. The rapid growth in size and complexity of knowledge graphs (KGs) available on the web has created a pressing need for efficient and adaptive methods to facilitate their understanding and exploration. Recently, semantic summaries have emerged as a means to quickly comprehend and explore large KGs. However, most existing approaches are static, failing to adapt to user needs and often struggling to scale. In this paper, we introduce iSummary, a workload-based and scalable approach for constructing selective summaries tailored to specific user requests. Unlike prior methods that process the entire KG, iSummary leverages query logs, exploiting the collective knowledge embedded in past user queries to identify relevant resources and relationships. The summarization process operates linearly with respect to the number of queries, enabling incremental and scalable summary generation even for large workloads. We formally define the (λ, κ)-Selective Summary problem, provide an approximate and efficient algorithm with theoretical guarantees, and evaluate it on two real-world datasets. Experimental results demonstrate that iSummary consistently outperforms existing techniques in both coverage and efficiency, producing high-coverage summaries up to 40× faster than state-of-the-art approaches.

Keywords: Semantic Summaries, Knowledge Graph, RDF

1. Introduction

Every day, a great deal of new information becomes available online. RDF knowledge graphs (KGs) are rapidly expanding to encompass millions or even billions of triples accessible through the web. For example, the Linked Open Data Cloud currently contains over 62 billion triples, organized into large and complex RDF data graphs [1].

The sheer size and complexity of these data sources limit their potential for effective use, which makes it essential to develop efficient methods to explore and understand their content [2]. In response to this challenge, semantic summarization methods have emerged trying to extract useful, concise information from large semantic graphs. Such summaries enable applications to perform tasks more efficiently than using the full data graphs, including visualization [3], exploration [4, 5], and query answering [6]. Structural semantic summaries primarily focus on the graph's structure to extract the necessary information and are classified into quotient and non-quotient ones. Quotient summaries devise supernodes and classify the graph nodes into those supernodes, while non-quotient summaries aim to identify the most important elements of the graph to produce the final summaries. In this paper, we focus on the latter category.

The problem. Most existing works on structural, non-quotient semantic summarization produce generic, static summaries [6]. However, as individuals have different data exploration needs, the generated summaries should be

*Corresponding author. E-mail: kondylak@ics.forth.gr.

tailored to each user's specific interests. Although this challenge has been acknowledged by the research community, current methods that offer use-based summaries generally rely on user-specified node weights. These are followed by algorithms that make assumptions about the relevant subsets of the semantic graph to complement the initial user input [7, 8]. More recent approaches, such as [9], leverage user queries to infer preferences, but they still compute the summary directly on the knowledge graph itself, requiring traversal or scoring over the entire graph, which is computationally expensive for large-scale KGs. Moreover, capturing a complete set of user queries is often impractical.

The solution. Instead of relying on node weights or a set of queries provided by users, we exploit generic logs already available through the SPARQL endpoints of the various KGs available online. Then, in order to generate a *selective* summary, i.e., a summary focusing on the user input, we only require one or a few nodes the user is most interested in. As previous users have already identified through their queries the most common connections to the specific user-selected nodes, we exploit this information in order to formulate the generated summaries. More specifically, to guide exploration, iSummary constructs what we call a (λ, κ)-Selective Summary, where λ denotes the number of nodes initially selected by the user as focal points (the "seed nodes"), and κ denotes the desired total number of nodes in the resulting summary. The goal is to generate a compact subgraph of size κ that best captures the information most relevant to the λ selected entities. Our contributions are summarized as follows:

- **Problem formulation**: We introduce and formally define the (λ, κ)-Selective Summary problem, proving that its optimal solution is NP-complete, as it generalizes the well-known Steiner tree problem.
- **Workload-based approach**: We propose iSummary, the first approximate, workload-driven method for constructing selective, structural, non-quotient semantic summaries. Instead of traversing the entire knowledge graph, iSummary leverages query logs—a compact behavioral representation of past user interactions—to infer relevant nodes and relations.
- **Scalable algorithm**: We design an efficient algorithm with theoretical quality guarantees, whose total cost scales linearly with the number of queries in the log. The algorithm exploits frequency statistics derived from query workloads and thus remains practical for large-scale query logs.
- **Hybrid use of the knowledge graph**: While the computation primarily relies on the query log, iSummary optionally accesses the underlying knowledge graph through its SPARQL endpoint only to instantiate unresolved variables in the generated summary. In this step, a single first result is retrieved per variable to ensure completeness.
- **Comprehensive evaluation**: We conduct extensive experiments on two real-world knowledge graphs (DBpedia and WikiData) and their corresponding workloads, showing that iSummary consistently outperforms all baselines in both summary quality (in terms of coverage) and execution time (up to 40× faster).
- **Open-source implementation**: We publicly release the full implementation and replication resources for iSummary, ensuring transparency and reproducibility.

To the best of our knowledge, this is the first approach to constructing selective, structural, non-quotient semantic summaries exploiting generic query workloads. iSummary primarily serves two complementary user profiles. First, non-expert or occasional users benefit from concise, interpretable overviews that highlight the most relevant entities and relations around a small set of seeds, enabling exploration without writing complex SPARQL queries. Second, expert users and data curators can exploit the summaries to identify frequently queried subgraphs, monitor evolving interests, and optimize endpoints. In both cases, iSummary provides an advantage over direct querying by surfacing workload-based, high-relevance connections that reflect collective user behavior and accelerate orientation within large and heterogeneous knowledge graphs. Further, it is important to emphasize that iSummary does not aim to summarize the entire knowledge graph directly. Instead, it constructs workload-based selective summaries, i.e., structural subgraphs inferred from the query workload that reflect the regions of the KG most frequently accessed by users. Hence, while the approach is grounded in the KG through its queries, it avoids traversing or scoring all triples. This enables scalability and highlights the parts of the KG that are practically most relevant to exploration, producing summaries that are faithful to the query-induced subgraphs rather than to the full KG topology.

A preliminary version of iSummary was presented at the ESWC 2023 [10] and demonstrated at ISWC 2023 [11]. In this journal version, we extend the original algorithm in several important ways. First, whereas the previous version handled only a single seed node ($\lambda = 1$), we now generalize the approach to multiple user-selected nodes

($\lambda gt1$), which non-trivially modifies node selection and linking. Second, we introduce two complementary procedures for variable resolution—one relying solely on the query log and one that optionally accesses the SPARQL endpoint—providing more flexibility and completeness. Finally, we expand our experimental evaluation to include both DBpedia and WikiData workloads, offering a broader and more rigorous assessment of quality and scalability.

The rest of this paper is organized as follows: Section 2 provides preliminaries and problem definition. Then, Section 3 presents our solution, iSummary, detailing the various steps for generating a selective summary. Section 4 presents the experimental evaluation of our work, whereas Section 5 presents related work. Finally, Section 6 concludes this paper and presents directions for future work.

2. Preliminaries & Problem Definition

2.1. Preliminaries

In this paper, we focus on RDF knowledge graphs, as RDF is among the most widely used standards for publishing and representing data on the Web, promoted by the W3C for semantic web applications. An *RDF KG* $\mathcal{G}$ is a set of *triples* of the form (s, p, o). A triple states that a *subject* s has the *property* p, and the value of that property is the *object* o. We consider only well-formed triples, according to the RDF specification [12]. These belong to $(\mathcal{U}\cup\mathcal{B})\times\mathcal{U}\times(\mathcal{U}\cup\mathcal{B}\cup\mathcal{L})$, where $\mathcal{U}$ is a set of Uniform Resource Identifiers (URIs), $\mathcal{L}$ a set of typed or untyped literals (constants), and $\mathcal{B}$ a set of blank nodes (unknown URIs or literals); $\mathcal{U}, \mathcal{B}, \mathcal{L}$ are pairwise disjoint. Additionally, we assume an infinite set $\mathcal{X}$ of variables that is disjoint from the previous sets. Blank nodes are essential features of RDF, allowing the support *unknown URI/literal tokens*. The RDF standard includes the `rdf:type` property, which allows specifying the type(s) of a resource. Each resource can have zero, one, or several types.

Complementary to this view, an RDF knowledge graph G—originally defined as a set of triples, can also be seen as directed graph $G = (V, E)$, where each node $v \in V$ corresponds to a resource or literal appearing in at least one triple of the form (s, p, o), and an edge $(s, o) \in E$ exists whenever a triple (s, p, o) is present for some predicate p. Thus, E abstracts the connectivity induced by the RDF triples while preserving the underlying structure of the KG. In the rest of this paper, we use this simplified graph notation for algorithmic clarity, since iSummary operates primarily over the node and edge structure rather than the full RDF labeling. In this representation, all predicates—including the special `rdf:type` relation—are treated uniformly as edges connecting resources. Thus, both entity–entity and entity–type connections contribute to the same structural graph used for summarization. This abstraction simplifies processing and reflects the way users typically query knowledge graphs, where `rdf:type` patterns often appear alongside other properties in the same query.

For querying, we use SPARQL [13], the W3C standard for querying RDF datasets. The basic building units of SPARQL queries are triple patterns and Basic Graph Patterns (BGP). A triple pattern is a triple from $(\mathcal{U}\cup\mathcal{B}\cup\mathcal{X})\times(\mathcal{U}\cup\mathcal{X})\times(\mathcal{U}\cup\mathcal{B}\cup\mathcal{L}\cup\mathcal{X})$. A set of triple patterns constitutes a BGP. The BGPs of a query can be seen as a graph structure where nodes represent the entities, values, or variables in the query and edges represent the properties (predicates) that connect the nodes, corresponding to the relationships in the data. Our implementation exploits all SPARQL queries in a given query log. However, it extracts the BGPs, ignoring FILTER, UNION, or OPTIONAL clauses, as our algorithm relies on co-occurrence and path frequency derived from such structural query patterns.

2.2. Informal problem statement

Informally, the problem we address may be described as follows: Given a knowledge graph G, a limited set of λ resources that the user wants his/her summary to be focused on, and a number κ denoting number of nodes to be included in the summary, efficiently construct a summary graph $G' \in G$ that best captures the user's preferred information in G. In practice, λ corresponds to the number of entities the user selects as starting points for exploration — for example, specific people, organizations, or concepts of interest. This aligns with typical KG exploration interfaces, where users begin by choosing one or more known entities to investigate. iSummary then builds the summary around these λ nodes to surface the most relevant surrounding information.

Resolving this problem is really important, as users usually visit a KG with a specific information request in mind, and are used to providing a starting point to begin the KG exploration that will lead to the information they are looking for. Usually, they are not interested in generic summaries of the overall graph, but would like to identify information pertinent to a specific part of the graph [14].

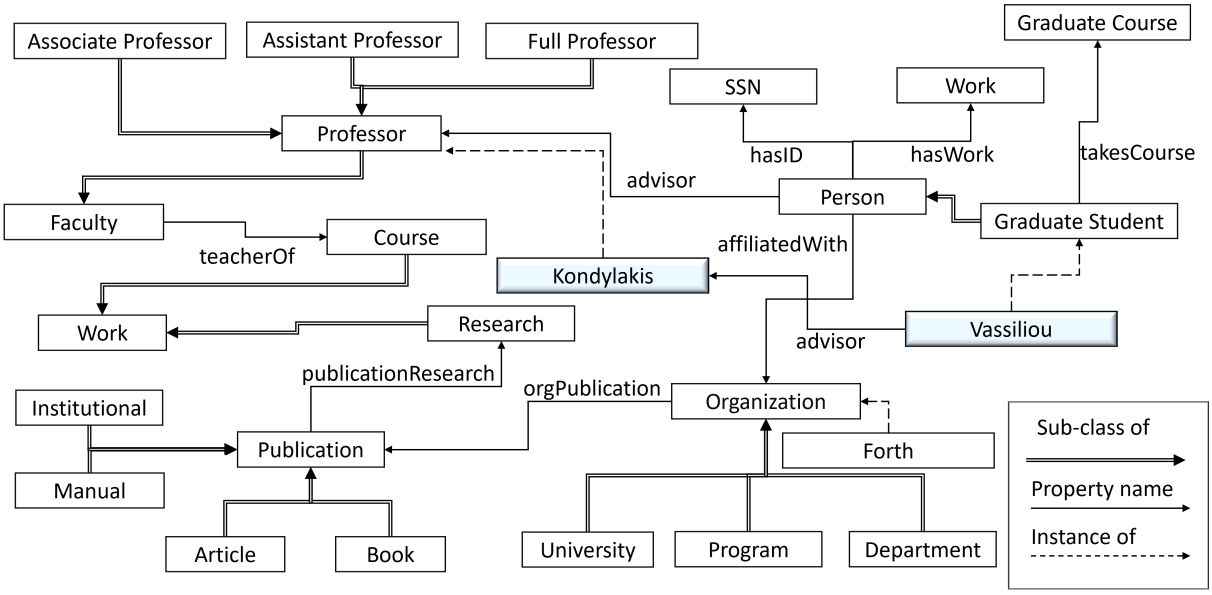

Figure 1. An example RDF KG.

Example 2.1. *The KG shown in Figure 1 includes information on the university domain. The figure visualizes people and organizations and also presents some indicative instances. Note that prefixes are omitted from the figure for the sake of clarity. Now assume that the user selects two nodes (λ=2), i.e., $S = \{Kondylakis, Vassiliou\}$ (the highlighted ones), and would like to get a selective summary of size five (κ=5). As such, three more nodes should be selected from the graph and linked to the two nodes provided by the user.*

A way to select the three additional nodes and the edges for the result summary, used by previous approaches (e.g., [15]) is to have weights available on all (or some of) the nodes and select the nodes maximizing the weight of the chosen sub-graph. However, those weights should be specific to user requests. For example, Publication might not be of interest when requesting a summary for Kondylakis and Vassiliou whereas it might be of interest when requesting a summary for Research.

In practice, users do not explicitly provide the numeric parameters λ and κ. Instead, they interact with the system by selecting one or more entities of interest (defining the set S of seed nodes, with $|S| = \lambda$) and optionally adjusting the desired summary size through an interface control (e.g., a slider or preset level). Hence, λ and κ should be understood as conceptual parameters that model the user's information need, rather than as values the user specifies directly.

2.3. Formal problem statement

The previous example makes it obvious that requesting the user to provide weights each time for all (or a least some) of the nodes is impractical. Next, we formally present the problem of (λ, κ)-Selective Summary and show that, although useful, it is also computationally expensive, in addition to being impractical.

Definition 1 ((λ,κ)-Selective Summary)**.** *Let $G=(V,E)$ be a knowledge graph with a non-negative node-weight function $w: V \to \mathbb{R}_{\geqslant 0}$. Given a set of user-selected* seed nodes *$S \subseteq V$ with $|S| = \lambda$ and a desired summary size κ (where $\lambda \leqslant \kappa$), a* (λ,κ)-Selective Summary *is a subgraph $G'=(V',E') \subseteq G$ such that*

$$S \subseteq V', \quad |V'| \leqslant \kappa, \quad \textit{and} \quad W(G') = \sum_{v \in V'} w(v) \textit{ is maximized.}$$

G' may contain one or more connected components, each summarizing the local context of one or several seed nodes; when all seeds are mutually reachable, G' becomes a single connected component (tree).

In our approach, these weights are not provided by the user but are automatically computed from query-log frequencies (Definition 2). Further, a solution to the (λ,κ)-Selective Summary problem is not necessarily unique, as there might be many maximum-weight trees with the smallest size that are equally useful for the user. Next, we prove that the aforementioned problem is NP-complete.

Theorem 1. *The (λ,κ)-Selective Summary problem is NP-complete.*

Proof. We consider the decision version: given a graph $G=(V,E)$, a seed set $S \subseteq V$ with $|S| = \lambda$, an edge-cost function $c: E \to \mathbb{Z}_{\geqslant 0}$, a node limit κ, and a threshold C_0, decide whether there exists a connected subgraph $H=(V_H,E_H)$ such that $S \subseteq V_H$, $|V_H| \leqslant \kappa$, and $\sum_{e \in E_H} c(e) \leqslant C_0$.

Membership in NP is immediate: given H, we can verify connectivity, the node bound, and the total cost in polynomial time.

NP-hardness follows by reduction from the classical STEINER TREE problem, which is NP-complete. Given an instance (G,T,c,C) of Steiner tree, construct the corresponding (λ,κ)-Selective Summary instance by setting $S=T$, $\lambda=|T|$, $\kappa=|V|$, and $C_0=C$. Any feasible Steiner tree connecting T within cost C is a feasible summary satisfying the same bound, and vice versa. The reduction is clearly polynomial, hence the problem is NP-hard. Since the problem is both in NP and NP-hard, it is NP-complete.

A nice property of the (λ,κ)-Selective Summaries is that their quality is *monotonically increasing* as the κ increases. This means that as the summary size increases, a greater amount of relevant information is added to the summary for the same seed nodes selected by the user.

Lemma 2. *Let S_κ be a (λ,κ)-Selective Summary and $S_{\kappa+1}$ be a $(\lambda,(\kappa+1))$-Selective Summary for G. Then $W(S_{\kappa+1}) \geqslant W(S_\kappa)$, where $W(S)$ the sum of all node weights in S.*

Proof. As S_k is a maximum-weight tree for λ including κ nodes, adding one more node in the summary and looking for the maximum-weight tree including that node as well guarantees that the total weight of S_{k+1} will be equal or greater than the total weight of S_k.

Over the years, many approximations have been proposed for resolving the Steiner tree problem [16, 17] that could be exploited for resolving the (λ,κ)-Selective Summary problem as well. CHeapeast INSertion (CHINS), one of the fastest approximation algorithms, has a worst time complexity of $O(2\kappa|V|\log|V|)$. CHINS starts with a partial solution consisting of a single selected node, and it incrementally adds the nearest one of the selected not yet in the solution. However, computing a Steiner tree approximate solution over commodity hardware for a large KG such as WikiData is not feasible. For example, assuming 1μs for each operation, running CHINS for WikiData that includes 1.4 billion statements would require more than a year to calculate a (5, 10)-Selective Summary.

3. iSummary

As we have shown in the previous section, computing a (λ,κ)-Selective Summary is both *impractical*, as different weights should be assigned to the graph nodes for each distinct user selection, and computationally *expensive*, as it requires computing a Steiner tree solution. In this section, we are going to provide an approximate solution based on *query workloads*.

3.1. Resolving the problem of multiple weight assignments.

Assume now that for the KG G we have a query log $Q = \{Q_1, \ldots, Q_n\}$ available. This assumption is reasonable, as all big KGs offer a SPARQL endpoint that logs user queries for various purposes. Multiple studies already confirm this (see, e.g., [18]), and we were also able to easily get access to such logs for DBpedia and WikiData (more about this in Section 4).

Having such a query log available, our first idea is that we can *use it to mine user preferences* for the specific seed nodes in which the user is interested. The idea here is that if a user is interested in a (λ, κ)-Selective Summary for a set S of λ specific seed nodes, we can use Q to identify *relevant queries* to that set, i.e., queries that include *at least one* of the λ nodes. In those queries, other nodes relevant to the user input will be available. In fact, since those queries have been issued by thousands of users, we assume that *the most useful related nodes will be those that appear more frequently* there.

Based on this assumption, we can have multiple weight assignments, one per user input, as they occur from thousands of user queries that involve the provided user input and that are based on past users' preferences, as expressed in their queries. Note here that we do not need weights for the whole graph, as by default, we can set the weight of the nodes that do not appear in the filtered user queries to zero.

Definition 2. *(Weight definition). For a given set of seed nodes S, let $Q_u(S)$ denote the subset of queries in which at least one element of S appears. The weight of a node v is then computed as*

$$w_u(v, S) = \frac{1}{|Q_u(S)|} \sum_{q_i \in Q_u(S)} \mathbb{I}[v \in q_i],$$

where $\mathbb{I}[v \in q_i] = 1$ if v occurs in q_i and 0 otherwise.

Intuitively, $w_u(v, S)$ captures how frequently node v co-occurs with the user's selected entities in past queries. Further, the mined weights and co-occurrence statistics for nodes could be stored in a dedicated index, maintained either globally (aggregating all query logs) or per user for personalized summarization, enabling fast reuse of computed frequencies, avoiding repeated log scans for each new summary request and supporting incremental updates as new queries appear.

Example 3.1. *Assume that for our example KG, shown in Figure 1, we have available a query log consisting of the following SPARQL queries:*

```
Q1. SELECT ?x ?y WHERE
 {x? a Person. z? a Professor. ?x advisor ?z.}
Q2. SELECT ?x ?y WHERE
 {x? a Person. y? a Organization. ?x affiliatedWith ?y.}
Q3. SELECT ?x WHERE
 {x? a Person. y? a Organization. ?x affiliatedWith ?y.
 ?y orgName "FORTH".}
Q4. SELECT ?y WHERE
 {y? a Organization.}
Q5. SELECT ?y WHERE
 {y? a Publication. ?x orgPublication ?y. ?x a University.}
Q6. SELECT ?o WHERE
 {Vassiliou a Person. Vassiliou advisor ?o.}
Q7. SELECT ?m WHERE
 {?m advisor Kondylakis. Kondylakis a Professor.}
Q8. SELECT ?p WHERE
 {Vassiliou affiliatedWith FORTH. Vassiliou lives ?p}
```

Suppose that a user selects the seed node set $S = \{Person\}$ ($\lambda = 1$) and we wish to compute weights for other nodes appearing with this seed. The subset of queries that mention Person is $Q(S) = \{Q1, Q2, Q3, Q6\}$, hence $|Q(S)| = 4$. Counting the occurrences yields:

$$w(Professor) = \tfrac{1}{4}(1) = 0.25, \quad w(Organization) = \tfrac{1}{4}(2) = 0.5, \quad w(Vassiliou) = \tfrac{1}{4}(1) = 0.25.$$

All other nodes have weight 0. Thus, Organization obtains the highest weight and will be added first when expanding the summary for the seed Person. Analogously, if $S = \{Publication\}$, only query Q5 contributes, giving $w(University) = 1.0$.

Finally, for a (2, 3)-Selective Summary for $S = \{Person, Professor\}$, $Q(S) = \{Q1, Q2, Q3, Q6, Q7\}$, hence $|Q(S)| = 5$. Frequencies now become:

$$w(Organization) = \tfrac{3}{5} = 0.6, \quad w(Vassiliou) = \tfrac{2}{5} = 0.4, \quad w(Kondylakis) = \tfrac{1}{5} = 0.2.$$

For $\kappa = 3$, we select one additional node beyond the two seeds, namely the most frequent one: Organization.

This simple computation illustrates how iSummary derives weights from query co-occurrence frequencies rather than from the KG itself, enabling efficient and user-dependent summary construction.

3.2. Resolving the computational problem

Now that we have a way to assign weights to the nodes, we will provide a computationally efficient procedure to link the selected nodes over a big graph. We use ideas proposed by the CHINS approximation algorithm. We begin with a solution including one arbitrary node by the ones selected by the user, adding one node each time, first from the ones selected by the user, and then from the ones with the maximum weight, till all k nodes are included in the summary. However, for doing so we will not use the original data graph but again *relevant user queries*. The main idea here is the following: link the k nodes using *the most frequent shortest paths* from the user queries, i.e., the most frequent paths with the minimal set of triple patterns that link them in user queries. Note that shortest paths are computed inside each query's internal graph (variables/terms and their joins), not on the KG; hence, no Steiner-tree computation is performed per query.

Example 3.2. *We now continue our example for constructing a (1, 2)-Selective Summary for the node Person. As we have already explained, the node Organization has the highest frequency in queries involving Person, and as such, it will be selected to be included in the summary. Now, instead of searching the graph shown in Figure 1 for linking Person with Organization we will additionally filter queries including Person, keeping only the ones including Organization as well. Those are Q2 and Q3. For each one of those queries, we calculate the minimal set of triple patterns for linking Person and Organization, and we eventually select the most frequent shortest path to include in the summary. As such, the (1, 2)-Selective Summary for the node Person includes the triples found in the shortest path of Q2:*

```
(?x, a, Person), (?y, a, Organization), (?x, affiliatedWith, ?y)
```

Note that the resulting subgraph satisfies the criteria of Definition 1: it includes the selected seed node ($\lambda = 1$), contains at most two nodes ($\kappa = 2$), and forms a connected component linking them through the most frequent minimal path pattern. Hence, it qualifies as a valid $(1, 2)$-Selective Summary. Note also that in this stage, the summary includes variables, but in the next subsection, we will explain how to remove them.

In the case where we are interested in a (1, 3)-Selective Summary for the node Person, the summary would have to include the Professor node as well, besides Organization. To link Person with Professor we would filter the queries to keep only those where both Professor and Person appear, i.e., Q1. Then, for linking those nodes, we would keep the most frequent minimal sets of triple patterns, i.e., $(?x, a, Person), (?z, a, Professor), (?x, advisor, ?z)$. Now, the (1, 3)-Selective Summary would become:

```
(?x, a, Person), (?y, a, Organization), (?x, affiliatedWith, ?y)
(?x, a, Person), (?z, a, Professor), (?x, advisor, ?z)
```

Using a weight based on the frequency as the selection criterion offers several advantages: it provides a simple, scalable, and interpretable measure of relevance that naturally highlights entities frequently co-occurring with the user's seed nodes in past queries. However, this heuristic also has inherent drawbacks. It may favor generally popular nodes, potentially overlooking less frequent but semantically important connections. Consequently, while frequency captures the collective importance of nodes across users, it may underrepresent rare or context-specific relations.

3.3. Resolving the variables in the summary

Note that the summary up to this point might contain variables, representing structural abstraction. Retaining them preserves generality and highlights recurring query patterns without committing to specific resources—useful when users wish to understand typical relationships and occurring patterns. Conversely, replacing variables with concrete resources yields more interpretable and example-driven summaries, suitable for presenting concrete instances or supporting user inspection. iSummary thus supports both views, balancing abstraction and specificity according to the intended use.

There are two options for replacing those variables. Using the first one, iSummary tries and *search again the query logs* in order to find patterns that can instantiate the missing variables:

Example 3.3. *As already shown in Example 3.2 the summary of the (1, 3)-Selective Summary includes the most frequent minimal set of triple patterns for linking a) Person with Organization and b) Person with Professor, i.e.:*

```
a:(?x, a, Person), (?x, affiliatedWith, ?y), (?y, a, Organization)
b:(?x, a, Person), (?x, advisor, ?z), (?z, a, Professor)
```

Now, for replacing the variables of the first path with concrete resources, we could search again the query log in order to find queries that map a triple pattern of the path to a concrete triple. In our case here $(?x, a, Person)$ *can be mapped to* $(Vassiliou, a, Person)$ *from Q6, and then the resource* $Vassiliou$ *is mapped to all* $?x$ *in that path. As such, the path now becomes*

```
a':(Vassiliou, a, Person), (Vassiliou, affiliatedWith, ?y),
 (?y, a, Organization)
```

Next, the same process is followed, and Q8 is used in order to replace $?y$ *with* $FORTH$*. As such, the path now becomes:*

```
a'':(Vassiliou, a, Person), (Vassiliou, affiliatedWith, FORTH),
 (FORTH, a, Organization)
```

The same process continues with the second path, which similarly considers Q6 and Q7 and becomes:

```
b:(Vassiliou, a, Person), (Vassiliou, advisor, Kondylakis),
 (Kondylakis, a, Professor)
```

As such, the complete summary is now:

```
(Vassiliou, a, Person), (Vassiliou, affiliatedWith, FORTH),
(FORTH, a, Organization), (Vassiliou, a, Person),
(Vassiliou, advisor, Kondylakis), (Kondylakis, a, Professor)
```

However, matching variables only to explicitly named entities appearing in the query log might provide limited coverage and may not resolve all variable instances. This restriction was a deliberate design decision to maintain scalability and determinism, as a) it ensures that replacements are grounded in observed user behavior and existing queries, and b) it avoids speculative inference or expensive graph traversals. Nevertheless, we acknowledge that this approach may miss entities not yet observed in the query log. To alleviate this, iSummary includes a second, hybrid procedure that queries the knowledge graph's SPARQL endpoint to instantiate unresolved variables. This already allows the system to discover and include entities that have not appeared in the log.

Example 3.4. *In the second option, the following query would be sent to the endpoint for completing the first path*

```
SELECT ?o WHERE {?o a Person.} LIMIT 1
```

Assuming that the endpoint returns Vassiliou as an answer to the query this would again make the path

```
a':(Vassiliou, a, Person), (Vassiliou, affiliatedWith, ?y),
 (?y, a, Organization)
```

Then a second query would be sent to the endpoint

```
SELECT ?y WHERE {Vassiliou affiliatedWith ?y.} LIMIT 1
```

Assuming that FORTH would be returned from the endpoint, that would lead to the following path.

```
a'':(Vassiliou, a, Person), (Vassiliou, affiliatedWith, FORTH),
 (FORTH, a, Organization)
```

Similarly after two more queries to the endpoint for the second path the overall summary would have been:

```
(Vassiliou, a, Person), (Vassiliou, affiliatedWith, FORTH),
(FORTH, a, Organization),
(Vassiliou, advisor, Kondylakis), (Kondylakis, a, Professor)
```

Although in the previous example, the summaries produced by the two approaches are the same, in the generic case, they might not be the same, as the endpoint might return different triples from the ones we find in the query log. Further, the first approach only relies on the query log, whereas the second one requires access to the SPARQL endpoint of the KG. Further, the simplified running example is designed purely for clarity of exposition. In practice, the same process scales directly to complex query patterns found in large query logs. For instance, in our evaluation on DBpedia and WikiData (Section 4), iSummary processed more than 250,000 real SPARQL queries containing multi-hop joins and dozens of triple patterns. Since the algorithm depends linearly on the number of queries and extracts frequent shortest paths from the workload rather than exploring the full knowledge graph, its behavior on large-scale data remains consistent with the principles illustrated in the example.

3.4. The complete algorithm

Now we are ready to present the corresponding algorithm for constructing an approximate (λ, κ)-Selective Summary for the λ input nodes S. The input set S corresponds to the entities or classes that define the user's current information needs. In practice, S is derived from the user's query or explicit selection of nodes of interest. For example, if a user issues a SPARQL query mentioning *Person* and *Organization*, then $S = \{Person, Organization\}$, and $\lambda = |S| = 2$. These input nodes serve as anchors around which *iSummary* constructs the selective summary.

The algorithm is presented in Algorithm 1 and receives as input κ, S, a query log Q, the SPARQL endpoint E, and a *MODE* according to which the variables in the last step of the summary should be resolved. Each query $q \in Q$ is represented as a query graph of basic graph patterns, as already explained in Section 2.1. The algorithm starts by visiting sequentially each seed node s (line 3). For each one of those nodes, it filters all queries to keep only the ones including s (line 4), and then the algorithm selects a share of the highest-frequency nodes from the corresponding subset of queries Q_s (lines 5-6) to keep in the Top set. Specifically, we allocate $\lfloor(\kappa - \lambda)/\lambda\rfloor$ additional nodes per seed, ensuring that the final summary includes approximately κ nodes in total (including the λ seeds)[1]. The next step is to visit one by one the *Nodes*, each time identifying an optimal way to link them to the ones already visited (lines 10-22). More specifically, for each node in $Nodes \setminus Visited$, we examine how to link them with the already visited nodes (lines 13-19). To do so, we filter the queries in Q, retrieving $Q_{x,y}$ that contains both x and y (line 14). Then, for each query in $Q_{x,y}$, we find the minimal set of triple patterns linking x with y (line 16) and we keep the most frequent one (line 18) to use for the result summary. However, as we identify paths in the queries, those might include variables that we should replace with actual resources. This is achieved through the *resolveVariables*

Algorithm 1 iSummary

Input: workload Q, seed set S with $|S| = \lambda$, summary size κ ($\lambda \leqslant \kappa$), resolution mode MODE $\in \{\mathsf{log}, \mathsf{KG}\}$, SPARQL endpoint E

Output: $Summary$ an approximate (λ, κ)-Selective summary

1: $Summary \leftarrow \emptyset$
2: $Nodes \leftarrow S$
3: **for all** $s \in S$ **do**
4: $\quad Q_s \leftarrow filter(Q, \{s\})$
5: $\quad n_s \leftarrow \lfloor \frac{\kappa - \lambda}{\lambda} \rfloor$
6: $\quad Top \leftarrow selectTopNodes(Q_s, n_s)$
7: $\quad Nodes \leftarrow Nodes \cup Top$
8: **end for**
9: $Visited \leftarrow \{$the first $s_0 \in S\}$ `// start from one seed (CHINS-style growth)`
10: **for all** $x \in (Nodes \setminus Visited)$ **do**
11: $\quad shortestPaths \leftarrow \emptyset$
12: $\quad selectedPath \leftarrow \emptyset$
13: $\quad$ **for all** $y \in Visited$ **do**
14: $\quad\quad Q_{x,y} \leftarrow filter(Q, \{x, y\})$
15: $\quad\quad$ **for all** $q \in Q_{x,y}$ **do**
16: $\quad\quad\quad shortestPaths[q] \leftarrow getMinimalConnectingPattern(q, \{x, y\})$
17: $\quad\quad$ **end for**
18: $\quad\quad selectedPath \leftarrow findMostFrequent(shortestPaths)$
19: $\quad$ **end for**
20: $\quad Visited \leftarrow Visited \cup \{x\}$
21: $\quad Summary \leftarrow Summary \cup resolveVariables(selectedPath, Q, MODE, E)$
22: **end for**
23: **return** $Summary$

function (line 21). As already explained, we provide two implementations for this function, which we will present in the sequel. Finally, we return to the user the constructed set of triples $Summary$ as a summary (line 23).

By design, iSummary produces tree-structured summaries conn ecting the selected seed nodes through the most frequent and informative paths. This restriction enables a polynomial-time computation and guarantees a concise, acyclic representation that users can easily interpret. In practice, the majority of query-induced connections already exhibit a tree-like structure around focal entities, making this assumption well-suited for summarization tasks. Nevertheless, in certain domains—such as scientific collaboration networks, organizational structures, or social graphs—important semantics may arise from cross-connections or cycles. We leave the corresponding exploration for future work.

Variable resolution. As already mentioned, the selected paths come from queries, which can contain variables that should be resolved and replaced by concrete resources. To achieve this, we have devised two algorithms with the following main ideas: a) replace the variables with resources mined from instantiated triple patterns in other queries, and b) formulate queries including the paths with the missing variables and answer them by the corresponding endpoint for replacing the variables with their instantiations. The two approaches are analytically presented in Algorithm 2 and Algorithm 3.

More specifically, Algorithm 2 begins by splitting the selected path into triple patterns (line 1) and trying to resolve the variables within each triple pattern individually. As such, for each triple pattern (line 5), we search all

[1] Note that if the selected seed nodes are not connected in the underlying KG or in the available query log, iSummary still produces a valid summary by including each selected node and its most frequent local neighbors derived from the log. In such cases, the generated summary may contain multiple disconnected components, each summarizing the local context of one seed node.

triple patterns in the queries (line 6) in order to find a mapping (line 8) able to instantiate the variable. As soon as this is found, it is propagated to the whole path, and the process continues with the other triple patterns, till no change occurs.

Algorithm 2 resolveVariables (the first approach using query workload Q)

Input: *selectedPath* the selected path, Q the query workload, *log* the selected mode for this algorithm, E the SPARQL endpoint
Output: the selected path with the resolved variables

```
1:  tps ← TriplePatterns(selectedPath)
2:  oldtps ← ∅
3:  while tps! = oldtps do
4:      oldtps ← tps
5:      for all tp ∈ tps do
6:          for all q ∈ Q do
7:              for all qtp ∈ TriplePatterns(q) do
8:                  if maps(qtp, tp) then
9:                      updateSelectedPath(selectedPath, tp, qtp)
10:                 end if
11:             end for
12:         end for
13:     end for
14:     tps ← TriplePatterns(selectedPath)
15: end while
16: return selectedPath
```

In Algorithm 3, on the other hand, the selected path is split into triple patterns (line 1), and each of those triple patterns is forwarded to the corresponding SPARQL endpoint in order to be answered (line 3). As soon as an answer is retrieved, the corresponding variables of the entire path are replaced with the result from the endpoint (line 4). The process is repeated until there is no triple pattern with a variable left. Note that as a selected path can have many results when issued to the SPARQL endpoint, we always pick the first one to be used for replacing the variables with their instantiations, leaving a more complex selection for future work.

Algorithm 3 resolveVariables (the second approach using the Endpoint E)

Input: *selectedPath* the selected path, Q the query workload, KG the selected mode for this algorithm, E the SPARQL endpoint
Output: the selected path with the resolved variables

```
1: tps ← TriplePatterns(selectedPath)
2: for all tp ∈ tps do
3:     a ← answerFromEndpoint(tp, E)
4:     updateSelectedPath(selectedPath, tp, a)
5:     tps ← TriplePatterns(selectedPath)
6: end for
7: return selectedPath
```

Comparison of the two resolution modes. The two variable–resolution strategies, namely the *log–based* and the *endpoint–based* approaches, differ not only in efficiency but also in several qualitative aspects.

(1) Data freshness: The endpoint–based mode retrieves bindings from the live knowledge graph, thereby reflecting the most recent state of the data. In contrast, the log–based mode depends on past user queries, which may lag

behind recent updates. This makes endpoint–based mode preferable in dynamic domains where new entities and relations frequently emerge.

(2) Semantic completeness: By accessing the KG directly, the endpoint–based mode can instantiate variables that have never appeared in the workload, producing semantically richer summaries. Conversely, the log-based mode restricts variable instantiations to entities observed in queries, ensuring that every replacement corresponds to concepts users have actually requested—yielding summaries that better mirror real exploration behavior.

(3) Stability and reproducibility: Since endpoint responses may depend on internal evaluation order or backend ranking, the endpoint–based mode may introduce slight non–determinism. The log-based mode, on the other hand, is entirely deterministic: the same workload and parameters always yield identical summaries, which is valuable for reproducible experiments.

(4) Scalability and autonomy: The log–based strategy operates offline and scales linearly with the number of queries, whereas the endpoint–based mode incurs additional network latency and depends on endpoint availability.

In practice, both variants yield highly similar summaries with an empirical divergence below 10% in our evaluation. Such divergence may stem from biases in the log content (e.g., reflecting the frequency of user interests) or from implementation details of the endpoint, such as default result ordering or sampling policies. In practice, these differences were observed to affect only a small fraction of variable bindings, typically in cases where multiple candidates were equally plausible. The top-ranked entities, which dominate both query frequencies and endpoint answers, generally coincide across the two methods.

Overall, the two approaches serve complementary purposes: *log–based* for stable, workload–faithful summarization and *endpoint-based* for situations demanding maximal semantic completeness or data freshness.

3.5. Theoretical guarantees

Determinism. The result produced by Algorithm 1 is deterministic: for a given (G, Q, S, κ) and parameter setting, it always produces the same summary. All operations rely on frequency counts and predefined tie-breaking rules. Specifically, nodes are sorted by their fixed weights, the initial seed is chosen deterministically (e.g., first URI in S), and the sub-procedures select the path of highest frequency with lexicographic tie-breaking. For resolving the variables the algorithm also follows a fixed policy. Since no randomization or sampling is used, repeated runs on the same input produce identical results. We note that in some cases, slightly longer paths could convey richer semantics by introducing intermediate, informative relations. Nevertheless, our design deliberately restricts selection to the most frequent shortest minimal connecting patterns, in order to maintain concise and interpretable summaries under the fixed parameter κ. This choice is consistent with the observation that user-generated SPARQL queries in practice tend to express direct, compact relationships.

Permutation invariance. Further, while the algorithm is deterministic, it is not strictly permutation invariant. This is because ties (e.g., equal path frequencies or identical weights) are resolved using a fixed total order on URIs and predicates (lexicographic ordering). Hence, reordering the elements in the input sets S or Q does not affect the frequencies, but may affect which node or path is selected among tied candidates. In practice, such differences are minor and occur only when frequencies are exactly equal. A fully permutation-invariant variant could be obtained by aggregating all ties or randomizing tie-breaking, which we leave for future work.

Approximation nature. The (λ, κ)-Selective Summary problem is NP-complete (Theorem 1); hence, *iSummary* employs a polynomial-time greedy approximation. The approximation arises in lines 10–22 of Algorithm 1, where the procedure iteratively selects the most frequent minimal connecting pattern between an unvisited node and the current partial summary. This local choice maximizes immediate coverage (path frequency) rather than global optimality, following the principle of the Cheapest Insertion (CHINS) Heuristic for the Steiner tree problem. Consequently, *iSummary* yields an approximate but efficient solution whose quality is formally bounded (see Theorem 3) and empirically validated in Section 4.

Theorem 3. *iSummary attains a multiplicative approximation guarantee for the (λ, κ)-Selective Summary maximization problem:*

$$\frac{W_{\text{opt}}}{W} \leqslant 2\left(1 - \frac{\ell}{k}\right),$$

where W and W_{opt} are the total weights of the summary returned by iSUMMARY *and an optimal summary, respectively; κ is the target summary size, and $\ell \geqslant 1$ is the number of terminals already connected at initialization.*

Proof. We formalize the reduction that underlies the proof and then appeal to the standard bound for the *Cheapest Insertion (CHINS)* heuristic on the induced instance.

Step 1: From queries to a weighted connectivity instance. Let $S \subseteq V$ be the user's seed set with $|S| = \lambda$. Using the query log Q, iSummary constructs the working graph $G_Q = (V_Q, E_Q)$ that contains (only) nodes and edges appearing in queries that mention at least one seed in S. Each edge $e \in E_Q$ is assigned a non-negative *frequency weight* $w(e)$ equal to its empirical frequency in Q after filtering by S (ties are immaterial for the argument). For any connected subgraph $H = (V_H, E_H)$ with $S \subseteq V_H$ and $|V_H| \leqslant \kappa$, define its *coverage weight*

$$W(H) \;:=\; \sum_{e \in E_H} w(e).$$

The (λ, κ)-Selective Summary objective is to find a connected H (respecting the size bound) that *maximizes* $W(H)$.

Step 2: A cost-minimization view. To link to CHINS (which minimizes cost), define edge costs $c(e) := w_{\max} - w(e)$, where $w_{\max} = \max_{e \in E_Q} w(e)$ is a constant for a fixed instance.[2] Then for any feasible H we have

$$C(H) := \sum_{e \in E_H} c(e) \;=\; |E_H|\, w_{\max} \;-\; W(H).$$

Because $|E_H|$ is bounded by a function of k (the construction grows a connected subgraph by at most one node per iteration; hence $|E_H| \leqslant k - 1$), maximizing $W(H)$ is equivalent (up to an additive constant) to minimizing $C(H)$. Let $H^\star$ be an optimal solution for $W(\cdot)$ and observe it is also optimal for $C(\cdot)$; denote $W_{\text{opt}} := W(H^\star)$ and $C_{\text{opt}} := C(H^\star)$.

Step 3: Metric closure and reduction to Steiner-type insertion. Let $R := S \cup T$ be the set of terminals that must be connected by the summary, where T contains the additional $(\kappa - \lambda)$ nodes that iSummary may include to respect the size bound while connecting S. Consider the metric closure $\mathcal{M}$ of G_Q under costs $c(\cdot)$: for any $u, v \in V_Q$, define $d(u, v)$ as the minimum c-cost of a u–v path in G_Q. The metric closure (V_Q, d) is a metric (triangle inequality holds by shortest paths). Any connected subgraph H that spans R corresponds to a Steiner tree in G_Q and to a tree over R in $\mathcal{M}$ with the same total c-cost.

Step 4: iSummary implements CHINS on $\mathcal{M}$. The main loop of iSummary starts from an initial connected set on S (which may be a single seed or a path connecting a subset of seeds; denote the number of terminals already connected at initialization by $\ell \geqslant 1$), and at each iteration inserts one new node $v \in V_Q \setminus V_H$ by choosing the *most frequent minimal connecting pattern* to the current partial summary. In the cost view, this is exactly the *cheapest insertion* of one terminal into the current partial tree: the algorithm selects v and a pair (x, y) of currently connected nodes that minimize

$$\Delta C(v \mid H) \;=\; d(x, v) + d(v, y) - d(x, y),$$

i.e., the increase in the d-length of the best tour/tree surrogate—equivalently (by standard transformations), the increase in Steiner-tree c-cost used in CHINS. Thus, iteration-by-iteration, iSummary is a CHINS run on the instance $(R, \mathcal{M})$.

Step 5: The CHINS bound. For Steiner-type insertion heuristics, the total cost C_{CHINS} after inserting $k - \ell$ terminals into an initial connected set of size ℓ satisfies [16]:

$$C_{\text{CHINS}} \;\leqslant\; 2\Big(1 - \frac{\ell}{k}\Big)\, C_{\text{opt}}.$$

[2] Any strictly decreasing affine transform of weights suffices. We use this one to keep costs nonnegative.

Intuitively, the factor 2 arises from doubling-and-shortcutting arguments, while the term $(1-\ell/k)$ accounts for the fact that we start from ℓ already-connected terminals and perform only $k-\ell$ insertions.

Step 6: Translating the bound back to weights. Using $C(H) = |E_H|\, w_{\max} - W(H)$ and the fact that $|E_H| \leqslant k-1$ for any feasible H, we have

$$W_{\text{opt}} - W \;=\; C - C_{\text{opt}} \;+\; \big(|E_H| - |E_{H^\star}|\big) w_{\max} \;\leqslant\; C - C_{\text{opt}} \;+\; (k-1) w_{\max}.$$

Combining with the CHINS inequality from Step 5 and absorbing the additive $(k-1)w_{\max}$ into the standard analysis (it is dominated by the telescoping sum used in the insertion proof and does not affect the leading approximation factor; see the affine-transform invariance noted in Step 2), yields

$$W \;\geqslant\; \frac{1}{2\big(1-\ell/k\big)}\, W_{\text{opt}}.$$

Equivalently,

$$\frac{W_{\text{opt}}}{W} \;\leqslant\; 2\Big(1-\frac{\ell}{k}\Big),$$

which is the claimed worst-case bound. [3]

Relation to Theorem 1. Theorem 1 establishes that the (λ,κ)-Selective Summary problem is NP-complete. Consequently, computing an exact optimal summary is intractable for large graphs. Theorem 3 therefore quantifies the performance of our polynomial-time heuristic iSummary, showing that it achieves a bounded approximation ratio. In other words, Theorem 1 justifies the need for an approximate method, and Theorem 3 guarantees how close this method can get to the optimal solution in the worst case.

Complexity. To identify the complexity of the algorithm, we should first identify the complexity of its components. Assuming $|Q|$ is the number of queries in the available workload, for each of the λ selected user nodes, we need to scan the log once for filtering and retrieving the *top* nodes, i.e., $O(\lambda|Q|)$. Then we need to connect each one of the κ nodes one at a time to the nodes already connected. This will result in k iterations in each of which the Q queries should be filtered, i.e. $O(k^2|Q|)$. Then, for each query appearing in the filtering results, we should run the Dijkstra algorithm to get the path with the minimal set of triple patterns. At the worst case for each node, we need to calculate the shortest paths for all queries, i.e., $O(k^2|Q||V_Q^2|)$, where V_Q^2 is the maximum number of nodes that appear in the queries in the workload.

Finally, for each of the κ selected paths to be included in the summary, we need one more scan of the Q in order to resolve the variables, leading to an additional $O(\kappa|Q|)$ in case we use algorithm 2. In case we use algorithm 3, we need at worst one query per triple pattern in the κ paths, i.e., $O(\kappa|tps|\log(T))$ where tps is the maximum number of triple patterns in a path and T is the number of triples in the knowledge graph. Although evaluating arbitrary SPARQL queries on a KG is NP-complete in combined complexity, iSummary does not execute such evaluations. It only analyzes and links patterns within bounded-size query graphs (basic graph patterns without filters), whose processing cost is polynomial in the number of triple patterns and thus tractable in practice.

Overall, the complexity of the algorithm (when using algorithm 2) is

$$O(\lambda|Q|) + O(k^2|Q||V_Q^2|) + O(\kappa|Q|) \leqslant O(|Q||V_Q^2|)$$

increasing additively by $O(\log(T))$ when algorithm 3 is used for query answering.

[3] **Remarks on ℓ.** The constant ℓ is fixed by the algorithm's initialization (e.g., $\ell = 1$ if we begin from a single seed; if the initialization phase connects a subset of seeds with a zero-cost pattern under $c(\cdot)$, then ℓ equals the number of terminals already connected). In all cases ℓ is independent of k and satisfies $1 \leqslant \ell \leqslant \min\{\lambda, k\}$.

In practice, the query log analysis can be preprocessed once, maintaining frequency statistics (e.g., node and co-occurrence counts) that can be incrementally updated as new queries are logged. This allows reusing these cached statistics, avoiding the need to rescan the entire log for each user request and further enhancing scalability in high-query environments.

Limitations. The aforementioned algorithm provides a solution to the (λ, κ)-Selective Summary problem. However, it assumes that *adequate* queries are available in the query log. In other words, it assumes that a) there are queries available including user input, and b) that there are at least κ other nodes available in those queries. These assumptions hold for popular online KGs, which can easily log thousands of user queries, but might not hold for other, less popular KGs. As such, our approach should be considered complementary to approaches working directly on the graphs of the KGs. However, as we showed, the problem is NP-complete, and neither existing approximate solutions nor competitors (as we will show) will terminate within a reasonable time.

Further, our approach assumes that the input query log mainly contains syntactically valid and semantically meaningful SPARQL queries, which was true for the already preprocessed queries we had in our datasets. In practice, logs may include malformed or empty-result queries, which should be filtered out during preprocessing, a process that might require significant time – nevertheless is only executed once and offline. While this assumption is reasonable for public SPARQL endpoints such as DBpedia and WikiData, it may reduce coverage in datasets with low-quality query logs.

Further, if the logs are sparse, biased, or fail to reflect genuine user interests, the resulting summaries may omit relevant regions of the KG or overrepresent frequently but uninformatively queried entities. This issue is inherent to all usage-driven methods that learn from user interactions. In practice, such effects can be mitigated by combining query-log statistics with structural signals from the KG (e.g., degree or centrality measures) or by applying smoothing, outlier detection, and sampling techniques to handle low-frequency queries. Algorithm 3 is a first step towards the direction of including signals from the KG. Future work will explore hybrid models that dynamically balance user-centric and structure-centric information to ensure robustness even under incomplete or noisy logs.

Finally, we note that iSummary assumes the user provides at least one valid entity ($\lambda \geqslant 1$) as a seed node for summary generation. This reflects a common interaction model in KG exploration systems, but may not capture scenarios where user interests are implicit or unspecified. Furthermore, the current implementation focuses on structural queries (Basic Graph Patterns) and does not yet exploit the filters, unions, or aggregation parts of the queries— these remain directions for future extensions.

4. Experimental Evaluation

In this Section, we present the experiments performed for evaluating our approach using two real-world datasets along with the corresponding query workloads. The source code and guidelines on how to download the datasets and the workloads are available online[4].

4.1. Setup

Implementation. The iSummary was developed using Java. The evaluation was performed using Windows 10 on an Intel® Core™ i3-10100 CPU @ 3.60 GHz (4 cores) with 16 GB RAM, running Java OpenJDK 17.

Datasets. The first dataset we use is DBpedia v3.8, along with the corresponding query workload. DBpedia is a structured dataset extracted from Wikipedia and stored as a knowledge graph in RDF. It enables users to query Wikipedia's structured data using SPARQL. DBpedia v3.8 consists of 422 classes, 1323 properties, and more than 2.3M instances. The available query workload is 16.3 MB, including 58,610 queries.

WikiData is a free and open knowledge base that can be read and edited by both humans and machines. WikiData is a collaborative, structured knowledge base that serves as a central repository for structured data across all Wikimedia projects (such as Wikipedia, Wikimedia Commons, and more). It is designed to store and manage

[4]https://github.com/giannisvassiliou/iSummary

machine-readable, linked data. WikiData contains 100 million items, and 1.4 billion statements, and covers many general topics for knowledge exploration and data science applications. The query workload for WikiData was retrieved from [19] and includes 192,325 queries

The query logs of both datasets had already been preprocessed, in order to retain only syntactically valid queries that return at least one answer to the corresponding graph.

4.2. Metrics

We already have shown a theoretical bound of our algorithm in terms of quality when compared to an optimal solution. In addition, it is not feasible to compute the optimal solution for our big graphs for evaluating the quality of the generated algorithms. As such we use **coverage**, which has been shown rather useful in evaluating structural, non-quotient semantic summaries in the past [20], [21], [4], [14], [22]. The idea behind coverage is that, ideally, we would like to maximize the portions of the queries that can be matched by the summary. In other words, a higher-quality summary is one that can correctly answer a larger number of query patterns and cover a greater fraction of each query's structure. However, as we are generating selective summaries, we would like the generated summaries to maximize the number and fragments that include the input provided by the user. As such, we define coverage as follows:

Definition 3 (Coverage)**.** *Assuming a (λ, κ)-Selective summary S for s, a query workload Q, and two weights for nodes and edges, w_n and w_p, we define coverage as follows:*

$$Coverage(Q, S, s) = \frac{1}{n} \sum_{s \in q_i} (w_n \frac{snodes(S, q_i)}{nodes(q_i)} + w_p \frac{sedges(S, q_i)}{edges(q_i)}) \quad (1)$$

where $nodes(q_i)$ and $edges(q_i)$ denote the number of nodes and edges respectively in q_i, and $snodes(S, q_i)$ and $sedges(S, q_i)$ denote the number of nodes and edges respectively that appear in S.

In our experiments, we set w_n=0.5 and w_p = 0.5 as we perceive both nodes and edges as equally important in a summary. Different applications, however, may warrant different choices:

- **Relation-/path-centric workloads** (e.g., where predicates and multi-hop structure carry the semantics): prefer $W_p > W_n$ so that missing edges penalize coverage more than missing nodes.
- **Entity-centric workloads** (e.g., catalog/browsing scenarios emphasizing which entities are present): prefer $W_n > W_p$.
- **Noisy aliasing or many near-duplicate entities**: a larger W_p can reduce sensitivity to superficial node variation by emphasizing structural matches.

We therefore report with $W_n = W_p$ as a neutral default and recommend adjusting them to reflect the relative value of entity presence versus relational structure in a given use case.

Additionally, we evaluate the efficiency of the various methods and baselines for computing the summary.

4.3. Procedure.

The query log Q was randomly divided into training (80%) and testing (20%) subsets. In this context, "training" does not involve parameter learning, but 80% of the queries are used for the construction of the summary. Then we use the remaining 20% of the queries for evaluating the summary in terms of coverage.

4.4. Baselines & competitors

We compare our approach with a *random* baseline, which we implemented to be used as a baseline. Our idea there was to randomly select nodes and edges from the training queries that include at least one of the λ nodes selected by the user to be included in the summary and then evaluate node selection and coverage over the test queries - so it

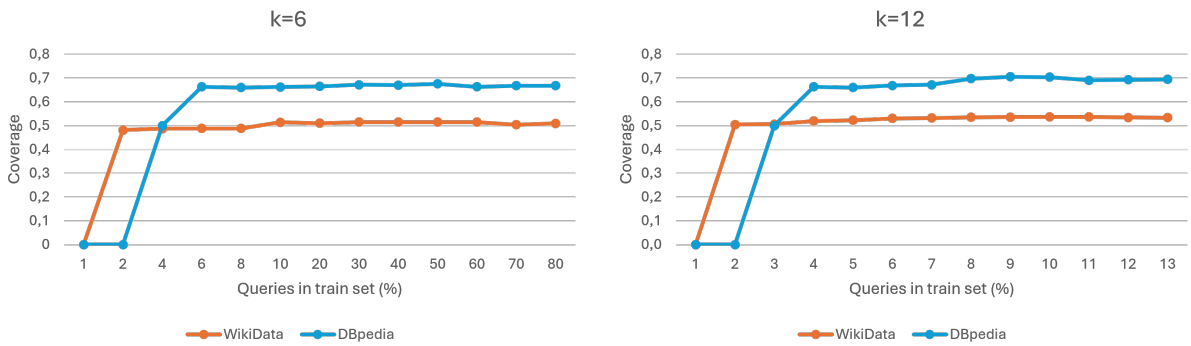

Figure 2. Coverage as the number of queries increases.

is not quite random from the whole graph, however it presents a naive approach that could be used to compare with a more sophisticated approach like iSummary.

In addition, we compare our approach with another summarization method for selective summaries *GLIMPSE* [9], which tries to maximize a user's inferred "utility" over a given KG, subject to a user- and device-specific constraint on the summary's size.

Finally, we explore an approximate version of the *personalized PageRank*[5] which works directly on the KG trying to identify the most important nodes and paths given the start nodes through random walks.

4.5. Results

4.5.1. Coverage for various query log sizes

In the first experiment, we try to understand what is the size of the query log required for getting high-quality results in terms of coverage for the iSummary only. As already explained, we keep a random 20% of the queries for testing and we use the remaining for the training. We gradually increase the percentage of queries considered and report the average coverage each time. We randomly pick a node to be used as a seed node for construction summaries for k=6, and 12. We repeat the experiment 10 times. We only present results using Algorithm 2, as we dedicate the next subsection to comparing Algorithm 2 with Algorithm 3. The results are shown in Figure 2, whereas the detailed tables including standard deviation are shown in Table 1.

As shown, the mean coverage values and standard deviations exhibit a consistent and interpretable pattern across both datasets and summary sizes. For very small training sets (1–2%), coverage values remain close to zero, as no queries containing the selected seed nodes are available in such small samples. Once a minimal number of relevant queries is introduced (from 2-4% of the total queries available for each dataset), i.e., 18K queries for DBpedia and 30K queries for WikiData, coverage rises sharply and quickly stabilizes.

For both $\kappa = 6$ and $\kappa = 12$, the coverage values for WikiData plateau around 0.53, with standard deviations below 0.015 after 4%, indicating high statistical stability and reproducibility across runs. DBpedia follows a similar trend but exhibits slightly higher variability, especially for $\kappa = 12$, where the standard deviation gradually increases as more queries are introduced. This variance can be attributed to DBpedia's greater structural and topical diversity, which results in more heterogeneous query subgraphs and consequently larger fluctuations in the resulting summaries.

Importantly, increasing the summary size from $\kappa = 6$ to $\kappa = 12$ yields a modest improvement in mean coverage (typically 0.01–0.02 points) without affecting the overall trend or stability. Linear regression on the coverage curves for both datasets confirmed that the slopes are statistically close to zero ($|\text{slope}| < 0.002$), showing that coverage saturates rapidly once approximately 10% of the query log is used for training. These results reinforce that iSummary

[5]https://github.com/asajadi/fast-pagerank

Table 1

Mean coverage and standard deviation for *iSummary* under varying percentages of training queries for summary sizes $\kappa = 6$ and $\kappa = 12$.

Percentage (%)	DBpedia ($\kappa = 6$)		WikiData ($\kappa = 6$)		DBpedia ($\kappa = 12$)		WikiData ($\kappa = 12$)	
	Cov.	Std.	Cov.	Std.	Cov.	Std.	Cov.	Std.
1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	0.000	0.000	0.481	0.000	0.000	0.000	0.504	0.012
4	0.500	0.028	0.487	0.003	0.500	0.027	0.506	0.009
6	0.663	0.004	0.488	0.000	0.663	0.004	0.520	0.008
8	0.660	0.007	0.488	0.012	0.660	0.007	0.523	0.003
10	0.662	0.004	0.514	0.012	0.668	0.007	0.530	0.002
20	0.665	0.007	0.510	0.012	0.672	0.004	0.532	0.003
30	0.672	0.007	0.515	0.001	0.697	0.011	0.535	0.001
40	0.670	0.007	0.515	0.001	0.705	0.004	0.535	0.002
50	0.676	0.004	0.515	0.001	0.704	0.006	0.536	0.002
60	0.663	0.007	0.515	0.001	0.691	0.005	0.537	0.002
70	0.668	0.007	0.504	0.015	0.692	0.007	0.534	0.002
80	0.668	0.007	0.509	0.012	0.694	0.007	0.534	0.000

Coverage values (*Cov.*) are averaged over ten runs; standard deviations (*Std.*) are rounded to three decimals for readability.

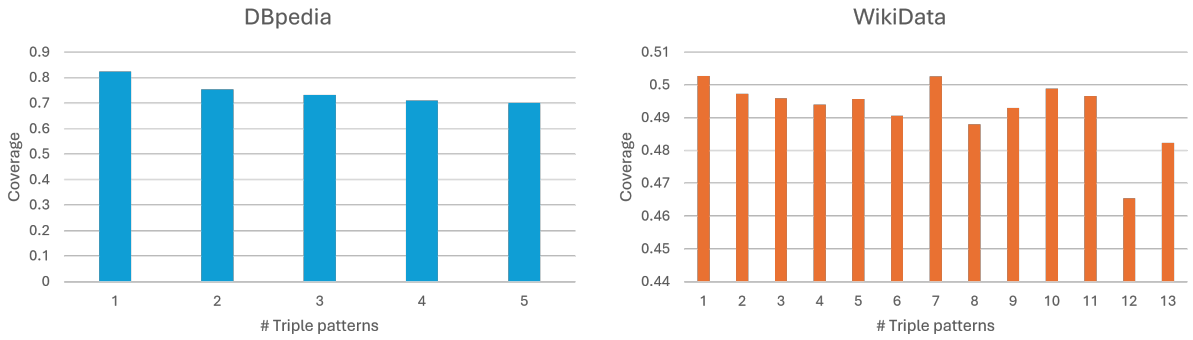

Figure 3. Coverage as the size of the triple patterns increases in the two datasets.

achieves stable, high-quality summaries with only a small fraction of the available queries, demonstrating both efficiency and robustness of the workload-based approach.

4.5.2. *Coverage vs query size*

As shown in Figure 3, the mean coverage gradually decreases as the number of triple patterns per query increases. This trend is expected: larger query patterns are structurally more complex and therefore harder for a concise summary to cover completely. For DBpedia, coverage starts at 0.825 for single-triple queries and decreases to approximately 0.70 for five-triple patterns, indicating that iSummary can already answer most simple queries and still retain reasonable coverage for moderately complex ones. WikiData exhibits lower absolute coverage values (around 0.50 on average) because its queries are typically deeper and involve more joins, increasing the denominator in the coverage computation. Nevertheless, the coverage values remain relatively stable beyond three triple patterns, showing that the generated summaries capture the most frequent structural patterns even as query complexity grows. Overall, the results confirm that iSummary maintains high coverage for simple and mid-sized queries across both datasets, and that coverage degradation with query complexity follows a smooth, predictable trend.

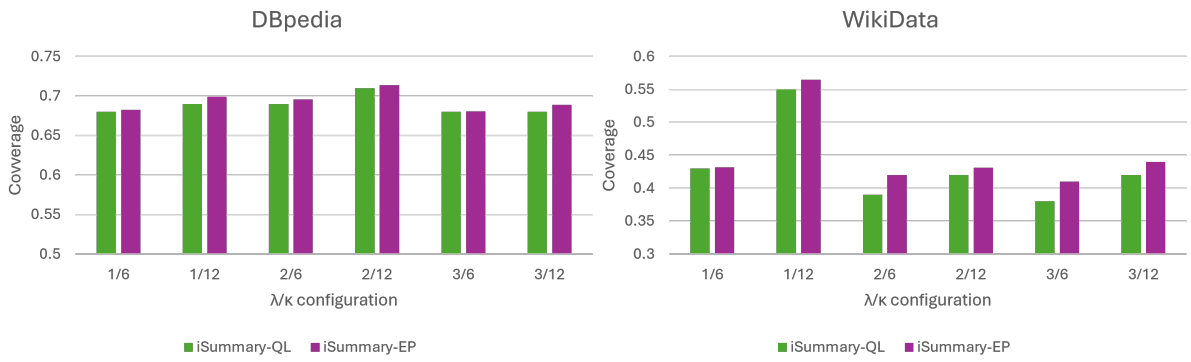

Figure 4. Coverage when using Algorithm 2 (iSummary-QL) or Algorithm 3 (iSummary-EP) for the DBpedia and WikiData datasets.

4.5.3. *Coverage using query log vs using the endpoint*

Having studied the number of queries required for effective training, we fix from here and on the percentage of queries for training to 80%, and the rest 20% we use for testing as commonly done in similar studies. Next, we compare coverage for iSummary when using Algorithm 2 vs Algorithm 3. We randomly make 10 selections for generating a (λ, κ)-Selective summary for various configurations (λ, κ)= {(1, 6), (1, 12), (2, 6), (2, 12), (3, 6), (3, 12)}. We repeat 10 times the aforementioned procedure. The results are shown in Figure 4 for both DBpedia (left) and WikiData (right).

As shown, in all cases the iSummary-EP dominates iSummary-QL as the endpoint can always resolve all variables with concrete resources, whereas this is not always possible from the query logs, leading to an increase in the total coverage. This is achieved with minimal impact in terms of execution time as we shall see in the sequel. Further as expected, the larger the summary the better the coverage in all cases. Finally, as already explained the larger queries of the WikiData lead to a smaller coverage when compared to the summaries produced for DBpedia.

4.5.4. *Coverage of iSummary vs baselines*

Next, we keep the same setting (80% of the queries for training and 20% for testing) and we compare with various baselines. More specifically we compare with Random which uses query logs, PPR and GLIMPSE which run directly on the KG, whereas we keep iSummary-EP for our case. Again all experiments run 10 times and we present the mean times. The results are shown in Figure 5 for both DBpedia (left) and WikiData (right).

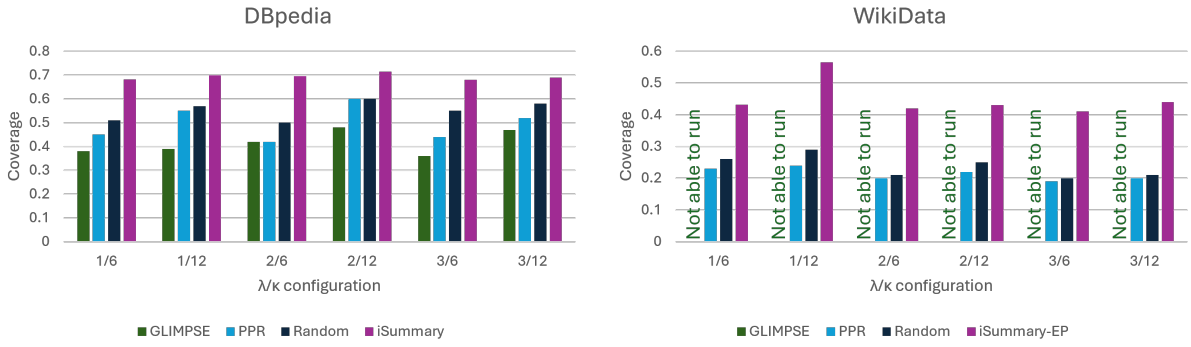

Figure 5. DBpedia coverage for various baselines.

Regarding DBpedia, as shown, approaches that work on the data graph have worse coverage than those working directly on the queries. GLIMPSE performs worst for all cases, as the provided input nodes are not enough to provide

a high-quality summary in terms of coverage. PPR has better results than GLIMPSE, but still, it is outperformed by both Random and iSummary.

The same trend appears for WikiData. Here, GLIMPSE is not able to produce output for such a big graph on our machine, as it fully loads the memory, and the application crashes after some time. When comparing with the other baselines again, iSummary has a relative improvement of 66-109% over Random and 95-135% over PPR.

We can also notice that as the size of the Selective Summary increases (κ=6,12) the coverage increases as well, as more nodes are added to the summary. Note also that as the size of the WikiData queries is larger than the other datasets, it is reasonable to be a bit more difficult to cover them, and as such, coverage is smaller. Nevertheless, the algorithm shows stability across two different datasets, always dominating other approaches.

It is worth clarifying that our Random baseline is not a uniform random selection from the entire knowledge graph, but rather a random selection restricted to the subset of queries that include at least one of the user-provided seed nodes. Thus, even though node and edge selection are random within this subset, the resulting summaries are still drawn from semantically relevant regions of the graph as defined by the workload. In contrast, GLIMPSE and PPR operate directly on the full knowledge graph without leveraging query-log information, often prioritizing globally central but workload-irrelevant entities. As a result, when evaluated using coverage over query workloads, the random baseline can occasionally outperform these global approaches. Nevertheless, iSummary consistently yields the highest mean coverage and lowest variance, demonstrating the benefit of frequency-aware, workload-driven summarization over both random and purely graph-based selection methods. Overall, iSummary outperforms all baselines with a relative improvement of 46-89% over GLIMPSE, 18-23% over PPR, and 18-39% over Random, showing the benefits of our approach.

4.5.5. Comparing execution time

Finally, we compare average execution times for the various algorithms. As in essence, only the size of the summary (κ) affects the execution time, we report the mean execution time for various κ. Further, we only present results for DBpedia as it is the only dataset that all competitors are able to run.

Figure 6 reports the average runtime of all methods. iSummary and Random operate on the query-induced subgraph G_Q, whereas PPR and GLIMPSE process the entire knowledge graph (KG). The subgraphs derived from queries are structurally limited in both size and complexity—they contain only the entities and relations actually appearing in user queries—while the KG includes all domain entities and relations. Consequently, methods restricted to G_Q complete their computations considerably faster. This efficiency reflects the smaller effective search space rather than an intrinsic algorithmic advantage. It is worth noting that in our experiments, although the query log contained nearly 100,000 queries, constructing and searching the corresponding query-induced graph still offered a significant runtime advantage, even without employing any caching or indexing mechanism. This suggests that query logs naturally concentrate on frequently accessed regions of the KG, producing a much sparser structure than the full graph. If the query log were to cover most of the KG or contain very large query patterns, the runtime gap would naturally diminish or even reverse.

Both versions of the iSummary (iSummary-QL and iSummary-EP) are just a bit slower than Random, showing that linking the k nodes using queries has a minimal impact on query execution, but improves the summaries' quality. On the other hand, sending a few questions to the endpoint to be answered (iSummary-EP) also has a minimal impact on execution time than just relying on query logs. Further, we can observe that as the k grows, all algorithms require more time to identify and link more nodes. Overall, although iSummary is only 0.13 times slower than Random, 14 times faster than GLIMPSE, and 40 times faster than PPR, it dominates largely all baselines in terms of coverage.

4.6. Qualitative evaluation and interpretability.

To complement the quantitative coverage results, we also report qualitative feedback collected during the *International Semantic Web Conference (ISWC 2023)* demonstration of iSummary [11]. Ten conference participants interacted with the live system on DBpedia and WikiData, following a guided scenario: each user selected a seed node (e.g., `Researcher`, `Organization`), varied the summary size κ, and inspected the generated subgraph. Afterwards, participants filled in a short evaluation form assessing the interpretability and perceived quality of the produced summaries. Results indicate that 90% of the participants found the summaries *intuitive and semantically*

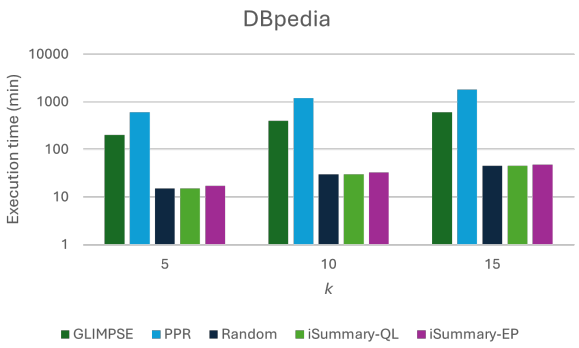

Figure 6. Execution time for the various k and algorithms.

meaningful, 80% stated that the summaries *accurately captured the main context* of the selected entity, and all participants (100%) found the graph visualization *clear and easy to interpret*. Overall, participants appreciated that the system balanced informativeness and readability, and confirmed that the resulting summaries aligned well with their expectations about the underlying data. This qualitative feedback reinforces our conclusion that iSummary produces summaries that are not only high in coverage but also interpretable and useful to end-users.

4.7. *Qualitative example on DBpedia (real entities)*

Consider now an example from DBpedia where the user provides the following seeds S={dbr:Barack_Obama, dbr:Michelle_Obama, dbr:Harvard_Law_School} (so $\lambda = 3$) and would like to retrieve a summary with size $\kappa = 6$. We work over the (small) induced subgraph G containing the following true DBpedia facts (IRIs shortened with dbr:/dbo:):

```
(dbr:Barack_Obama, dbo:spouse, dbr:Michelle_Obama)
(dbr:Barack_Obama, dbo:almaMater, dbr:Harvard_Law_School)
(dbr:Michelle_Obama, dbo:almaMater,dbr:Harvard_Law_School)
(dbr:Barack_Obama, dbo:almaMater, dbr:Columbia_University)
(dbr:Michelle_Obama, dbo:almaMater, dbr:Princeton_University)
```

(i) Exact solution on the KG (Steiner tree): The minimum tree that connects the three seeds in G uses two edges and is the following: {(dbr:Barack_Obama, dbo:spouse, dbr:Michelle_Obama),(dbr:Michelle_Obama, dbo:almaMater, dbr:Harvard_Law_School)}.

This is a Steiner tree with zero Steiner nodes (all terminals are directly linked). It yields a connected component on the three seeds. Any other two-edge choice (e.g., replacing the second edge with Barack_Obama $\xrightarrow{\text{dbo:almaMater}}$ Harvard_Law_School) is also optimal w.r.t. size; if node weights favor Michelle_Obama $\twoheadrightarrow$ Harvard_Law_School, the chosen variant maximizes $W(\cdot)$. Computing this exact solution amounts to the Steiner problem over the induced G (NP-complete in general). In this example, we assume that all nodes have the same weight, as in essence, we cannot expect that users will have assigned a weight to the nodes.

(ii) CHINS on the KG (Cheapest Insertion heuristic): Starting from one seed (e.g., dbr:Barack_Obama), CHINS adds the closest terminal, here dbr:Michelle_Obama via dbo:spouse. Next it inserts dbr:Harvard_Law_School through the cheapest insertion (one hop from either Obama). In this graph, CHINS returns the same two-edge tree as above. In larger graphs CHINS runs in $O(2^{\kappa}|V|\log|V|)$ and is a fast, classical Steiner approximation.

(iii) CHINS guided by query logs (iSummary): Instead of computing paths on the KG, iSummary extracts *most frequent shortest patterns* from the query log restricted to the seeds and then resolves variables (Alg. 1). Concretely:

1. **Select expansions by frequency (Sec. 3.1).** Among nodes co-occurring with the seeds in the log, the top ones for this case typically include `dbr:Harvard_Law_School`, `dbr:Columbia_University`, `dbr:Princeton_University`.
2. **Link by the most frequent shortest patterns (Sec. 3.2).** For (`Barack_Obama`, `Michelle_Obama`) the pattern `?x dbo:spouse ?y` is the most frequent shortest connector; for each Obama to `Harvard_Law_School`, the pattern `?x dbo:almaMater dbr:Harvard_Law_School` dominates.
3. **Resolve variables (Sec. 3.3).**
   - **QL mode:** instantiate variables using mappings seen in other queries from the log.
   - **EP mode:** issue bounded SPARQL lookups to the endpoint to bind remaining variables (LIMIT 1), yielding concrete triples deterministically up to endpoint order.

Algorithmically, we first add the λ seeds and then up to $\kappa - \lambda$ high-frequency neighbors, linking each insertion with its most frequent shortest pattern (Alg. 1, lines 10–22). Variable resolution follows Alg. 2 (QL) or Alg. 3 (EP). For this seed set, both modes select the same *backbone* as the Steiner/CHINS KG solutions:

$$\texttt{Barack\_Obama} \xrightarrow{\texttt{dbo:spouse}} \texttt{Michelle\_Obama} \xrightarrow{\texttt{dbo:almaMater}} \texttt{Harvard\_Law\_School}.$$

With $\kappa = 6$, iSummary then uses the remaining budget to add the most frequent co-occurring universities for each person and their frequent connectors, e.g., $\texttt{Barack\_Obama} \xrightarrow{\texttt{dbo:almaMater}} \texttt{Columbia\_University}$, $\texttt{Michelle\_Obama} \xrightarrow{\texttt{dbo:almaMater}} \texttt{Princeton\_University}$.

Thus, a typical iSummary (EP) output is the 6-node tree on {`Barack_Obama`, `Michelle_Obama`, `Harvard_Law_School`, `Columbia_University`, `Princeton_University`, (one seed picked first)}, linked only by `dbo:spouse`/`dbo:almaMater` paths found as the most frequent shortest patterns in the log. EP and QL produce near-identical structures; EP can resolve residual variables even when the log lacks explicit instantiations (Sec. 3.3).

The exact Steiner on KG identifies the minimal connector among the seed nodes (two edges in this case), providing an optimal solution but remaining intractable for large graphs. The CHINS on KG approach reproduces the same connector in this example, offering a scalable heuristic that operates directly on the knowledge graph. Finally, iSummary (CHINS + logs) bypasses KG traversal altogether by linking nodes through frequent shortest query patterns extracted from query logs and resolving variables via QL or EP. This method is deterministic under fixed tie-breaking and exhibits linear complexity in the number of queries, as discussed in Sections 3–4).

4.8. *Discussion*

In this subsection, we provide further discussion on selected aspects of our approach.

Applicability on small graphs. The proposed method does not require any modification for smaller knowledge graphs. Its computational complexity grows linearly with the number of nodes and edges visited and with the size of the query log. However, the number of logged queries needed to achieve stable coverage depends on the ratio between the query log and the graph size. For smaller graphs, the same number of queries typically yields higher coverage because entities and relations are revisited more frequently, resulting in faster convergence of the coverage metric. Conversely, larger graphs require proportionally more queries to achieve comparable coverage. Hence, no algorithmic adaptation is needed for small graphs; only the relative amount of available log data affects convergence speed.

On the comparability of query-log and KG-based methods. We emphasize that the knowledge graph (KG) and the query log capture complementary aspects of the data ecosystem. The KG represents the complete factual structure of the domain (e.g., WikiData), while the query log reflects user demand, which may be partial, biased, or semantically inconsistent with the underlying graph. Accordingly, our evaluation does not aim to establish that one data source is inherently superior, but rather to quantify how summarization based on user interactions (iSummary) differ from purely structural summarization (e.g., PPR, GLIMPSE) in practice. The comparison thus highlights the

distinct nature of the summaries: query-log–based summaries focus on frequently accessed, semantically salient regions, whereas KG-based summaries emphasize global connectivity.

Query log privacy considerations. The query logs exploited by iSummary are public, anonymized, and aggregated datasets (such as those released for DBpedia and WikiData), which contain no personally identifiable information or user metadata. Our approach operates solely on query structures and frequencies, without recording or inferring individual user behavior, thereby ensuring full compliance with standard data privacy practices.

Nevertheless, the approach presupposes the availability of a sufficiently large query workload to estimate reliable co-occurrence statistics. Empirically, stable coverage values were observed once the query log reached approximately 18-30K entries for graphs of comparable scale to DBpedia or WikiData. For smaller knowledge graphs, proportionally fewer queries are typically enough, as the coverage function saturates faster due to the smaller space of possible entities and relations.

5. Related Work

Following the taxonomy introduced by Cebirić et al. [6], semantic graph summarization methods can be broadly divided into *quotient* and *non-quotient* approaches. Quotient-based methods generate summaries by merging nodes that share similar types or structural properties, producing a quotient graph where each summary node represents an equivalence class of entities. These approaches yield compact global overviews but cannot provide selective, instance-level representations. In contrast, non-quotient methods construct summaries as subgraphs of the original knowledge graph, retaining the identity of selected nodes and edges and focusing on relevance or importance. Our approach, iSummary, belongs to the latter category: it is a *structural, non-quotient, and selective* summarization method that allows users to select nodes of interest for the summary to be based on, and then identifies and links existing nodes based on their prominence in real query workloads. As such, in the rest of the section, we focus on selective, structural, and non-quotient summaries. For a complete overview of the works in the area, the interested reader is forwarded to relevant surveys available in the domain [6], [23].

Among the first works that focused on generating selective non-quotients is [15], which returns a summary of an RDF Sentence Graph. An RDF Sentence Graph is a weighted, directed graph where each vertex represents an RDF sentence. A link between two sentences exists if an object of one sentence belongs to another sentence as well. The creation of a sentence graph is customized by domain experts, who provide as input the desired summary size and their navigation preferences, i.e., weights in the links they are most interested in. However, as we already explained, this is impractical and does not scale, as different weights should be assigned to the graph nodes for each distinct user selection. iSummary directly mines those weights, automatically from query logs.

The method by Queiroz-Sousa et al. [24] builds personalized ontology summaries by computing node relevance through a weighted combination of centrality and closeness measures and then applying the Broaden Relevant Paths (BRP) algorithm to connect the most relevant ontology concepts into a coherent subgraph. Personalization arises from user-defined parameters such as mandatory concepts or relevance thresholds. However, the approach depends heavily on explicit user input and static ontology structure, without leveraging real query behavior or workload information. In contrast, iSummary eliminates the need for user-provided relevance parameters by automatically deriving importance from the statistical properties of query logs. It thus learns implicit user interests collectively from observed workloads rather than from manually set preferences. Furthermore, while Queiroz-Sousa et al. operate directly on ontologies and compute concept centrality at the schema level, iSummary operates on the whole knowledge graph and constructs workload-aware, non-quotient summaries that preserve only the most frequent and semantically meaningful paths observed in user queries. Finally, iSummary provides formal approximation guarantees and achieves linear scalability with respect to the number of queries, enabling it to handle large-scale datasets such as DBpedia and WikiData, which are far beyond the scope of the ontology-level summaries targeted by Queiroz-Sousa et al.

GLIMPSE [9] is the most relevant work to our approach and focuses on constructing selective summaries of KG containing only the facts most relevant to individuals' interests. However, it requires the user to provide a set of relevant queries that would like relevant information to be included in the summary, whereas the corresponding algorithms are directly executed on the KG. However, it is difficult for a user to provide these queries in each case,

and although the corresponding algorithm has linear complexity in the number of edges in the KG, it still faces scalability problems, as it has to actually visit the KG. As shown is not able to run for big KGs. iSummary, on the other hand, exploits query logs instead of the KG and does not require the user to specify a set of queries each time for his/her exploration.

Finally, WBSUM [14] and iSummary share the fundamental idea of exploiting query workloads to produce structural, non-quotient summaries of knowledge graphs, but they differ substantially in purpose, granularity, and design. WBSUM constructs global schema-level summaries by identifying the most frequently queried schema nodes and connecting them via an approximate Steiner tree. In contrast, iSummary focuses on generating selective, personalized summaries tailored to a specific seed set of entities. Instead of summarizing the KG as a whole, it produces workload-aware subgraphs that reflect user-relevant entities and relations observed in real query logs. Furthermore, while WBSUM models summary construction as a global optimization problem on the schema graph, iSummary integrates workload-driven path selection with formal coverage guarantees and deterministic behavior, scaling linearly with the number of processed queries. Conceptually, WBSUM addresses the macro-level problem of understanding the structure of entire datasets, whereas iSummary tackles the micro-level problem of producing fine-grained, context-specific summaries that remain both interpretable and efficient at web scale.

Table 2

Comparison of related summarization approaches.

Approach	**Personalization / User Input**	**Information Source**	**Scalability and Complexity**	**Main Limitations / Distinct Features**
Zhang et al. (2007)	Manual: domain experts define weights and summary size	RDF Sentence Graph constructed from ontology triples	Moderate; depends on ontology size and expert parameters	Requires expert-provided weights; impractical for large-scale or dynamic settings.
Queiroz-Sousa et al. (2013)	Semi-automatic: user provides parameters, mandatory concepts, or thresholds	Ontology structure (graph of concepts and relations)	Polynomial; depends on number of ontology concepts	Personalized but relies on explicit user input; no workload exploitation.
GLIMPSE	Explicit: user provides representative query set	Entire knowledge graph; submodular optimization of query utility	Linear in number of edges; high runtime for large KGs	Requires user queries; scalability limited by direct KG traversal.
WBSUM	No personalization	Query workload over the schema graph	Efficient;	Produces global schema summaries; not selective or entity-focused.
iSummary	Minimal: only seed node(s) provided by user	Generic query logs; no KG traversal required	Linear in number of queries; deterministic and reproducible	Generates fine-grained, personalized, workload-aware summaries with theoretical coverage guarantees.

Table 2 presents an overview of the related work, contrasting it with iSummary. Overall, our work is the first, structural, non-quotient, workload-based selective summarization method. Our work accepts minimal user input and exploits query workloads to generate high-quality summaries. Further, our algorithm is linear in the number of queries available and, as such, efficient and scalable. Even when exploiting the SPARQL endpoint it requires a small number of queries to be answered with minimal impact on execution time.

6. Conclusions

In this paper, we present a summarization method able to construct selective, workload-based, semantic summaries with high quality. We formulate the problem of (λ, κ)-Selective Summaries and provide a simple algorithm for resolving the problem in its general form, linear in the number of queries available in the query logs with theoretical guarantees. As the produced algorithm might include variables from user queries, we present two methods for replacing them with concrete resources. Our algorithm effectively identifies different weight assignments for different inputs and is able to efficiently and effectively identify how to link the selected nodes based on the available queries. Further, it exploits two methods for replacing variables with concrete resources either from the query logs or from the KG endpoint.

We experimentally show that even 18-30K queries are enough for generating high-coverage summaries, and we compare our approach with several baselines. We demonstrate that our approach strictly dominates all baselines in

terms of query coverage (18-109% better coverage than Random, 18-135% better coverage than PPR and 46-89% better coverage than GLIMPSE) and it is highly efficient, being orders of magnitude faster than relevant approaches working directly on the KG.

Future Work. As future work, we intend to study how selecting more related instances from the KG would enhance the quality of the result. An option would be, for example, to identify the most frequent resource or the one present in the mappings of the other triple patterns of the same query.

Further, when query logs are unavailable or insufficiently representative, a complementary approach could be to revert to a structure-driven variant that derives edge weights directly from the knowledge graph (e.g., based on degree, centrality, or semantic relatedness). This alternative could be effective for ensuring connectivity and coverage, but might lack the user-driven weighting that guides iSummary toward frequently explored or semantically salient regions. Nevertheless, for entirely new domains or niche user interests not yet reflected in logs, this fallback might provide a functional starting point that we intend to explore next. Future work will investigate adaptive hybridization, where structural signals from the KG are dynamically balanced with query-log statistics to maintain robustness across varying data conditions.

Another interesting direction is that selective summaries change over time for specific user input. As users' interests drift in time, it would be interesting to identify how the selective summaries also change, considering that queries' focus changes through time due to specific events, disasters, seasonality, or occasions.

Finally, as (λ, κ)-Selective Summaries are not necessarily unique, introducing the element of diversity would be interesting so that the users are not always presented with the same selective summary.

References

[1] The Linked Open Data Cloud, Accessed: 2022-09-22.

[2] G. Troullinou, H. Kondylakis, M. Lissandrini and D. Mottin, SOFOS: Demonstrating the Challenges of Materialized View Selection on Knowledge Graphs, in: *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, G. Li, Z. Li, S. Idreos and D. Srivastava, eds, ACM, 2021, pp. 2789–2793. doi:10.1145/3448016.3452765.

[3] A. Pappas, G. Troullinou, G. Roussakis, H. Kondylakis and D. Plexousakis, Exploring Importance Measures for Summarizing RDF/S KBs, in: *The Semantic Web - 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 - June 1, 2017, Proceedings, Part I*, E. Blomqvist, D. Maynard, A. Gangemi, R. Hoekstra, P. Hitzler and O. Hartig, eds, Lecture Notes in Computer Science, Vol. 10249, 2017, pp. 387–403. doi:10.1007/978-3-319-58068-5_24.

[4] G. Troullinou, H. Kondylakis, K. Stefanidis and D. Plexousakis, Exploring RDFS KBs Using Summaries, in: *The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part I*, D. Vrandecic, K. Bontcheva, M.C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L. Kaffee and E. Simperl, eds, Lecture Notes in Computer Science, Vol. 11136, Springer, 2018, pp. 268–284. doi:10.1007/978-3-030-00671-6_16.

[5] G. Troullinou, H. Kondylakis, K. Stefanidis and D. Plexousakis, RDFDigest+: A Summary-driven System for KBs Exploration, in: *Proceedings of the ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, USA, October 8th - to - 12th, 2018*, M. van Erp, M. Atre, V. López, K. Srinivas and C. Fortuna, eds, CEUR Workshop Proceedings, Vol. 2180, CEUR-WS.org, 2018.

[6] Š. Čebirić, F. Goasdoué, H. Kondylakis, D. Kotzinos, I. Manolescu, G. Troullinou and M. Zneika, Summarizing semantic graphs: a survey, *The VLDB journal* **28**(3) (2019), 295–327.

[7] A. Alzogbi and G. Lausen, Similar structures inside rdf-graphs, in: *LDOW*, 2013.

[8] G. Wu, J. Li, L. Feng and K. Wang, Identifying potentially important concepts and relations in an ontology, in: *International Semantic Web Conference*, Springer, 2008, pp. 33–49.

[9] T. Safavi, C. Belth, L. Faber, D. Mottin, E. Müller and D. Koutra, Personalized Knowledge Graph Summarization: From the Cloud to Your Pocket, in: *2019 IEEE International Conference on Data Mining, ICDM 2019, Beijing, China, November 8-11, 2019*, J. Wang, K. Shim and X. Wu, eds, IEEE, 2019, pp. 528–537. doi:10.1109/ICDM.2019.00063.

[10] G. Vassiliou, F. Alevizakis, N. Papadakis and H. Kondylakis, iSummary: Workload-Based, Personalized Summaries for Knowledge Graphs, in: *The Semantic Web - 20th International Conference, ESWC 2023, Hersonissos, Crete, Greece, May 28 - June 1, 2023, Proceedings*, C. Pesquita, E. Jiménez-Ruiz, J.P. McCusker, D. Faria, M. Dragoni, A. Dimou, R. Troncy and S. Hertling, eds, Lecture Notes in Computer Science, Vol. 13870, Springer, 2023, pp. 192–208. doi:10.1007/978-3-031-33455-9_12.

[11] G. Vassiliou, N. Papadakis and H. Kondylakis, iSummary: Demonstrating Workload-based, Personalized Summaries for Knowledge Graphs, in: *Proceedings of the ISWC 2023 Posters, Demos and Industry Tracks: From Novel Ideas to Industrial Practice co-located with 22nd International Semantic Web Conference (ISWC 2023), Athens, Greece, November 6-10, 2023*, I. Fundulaki, K. Kozaki, D. Garijo and J.M. Gómez-Pérez, eds, CEUR Workshop Proceedings, Vol. 3632, CEUR-WS.org, 2023. https://ceur-ws.org/Vol-3632/ISWC2023_paper_435.pdf.

[12] W3C, Resource Description Framework.
[13] W3C Recommendation, SPARQL Query Language for RDF, Accessed: 2019-10-09.
[14] G. Vassiliou, G. Troullinou, N. Papadakis and H. Kondylakis, WBSum: Workload-based Summaries for RDF/S KBs, in: *SSDBM 2021: 33rd International Conference on Scientific and Statistical Database Management, Tampa, FL, USA, July 6-7, 2021*, Q. Zhu, X. Zhu, Y. Tu, Z. Xu and A. Kumar, eds, ACM, 2021, pp. 248–252. doi:10.1145/3468791.3468815.
[15] X. Zhang, G. Cheng and Y. Qu, Ontology summarization based on rdf sentence graph, in: *Proceedings of the 16th international conference on World Wide Web*, 2007, pp. 707–716.
[16] F.K. Hwang and D.S. Richards, Steiner tree problems, *Networks* **22**(1) (1992), 55–89. doi:10.1002/net.3230220105.
[17] S. Voß, Steiner's Problem in Graphs: Heuristic Methods, *Discrete Applied Mathematics* **40**(1) (1992), 45–72. doi:10.1016/0166-218X(92)90021-2.
[18] A. Bonifati, W. Martens and T. Timm, An analytical study of large SPARQL query logs, *VLDB J.* **29**(2–3) (2020), 655–679. doi:10.1007/s00778-019-00558-9.
[19] S. Malyshev, M. Krötzsch, L. González, J. Gonsior and A. Bielefeldt, Getting the Most Out of Wikidata: Semantic Technology Usage in Wikipedia's Knowledge Graph, in: *The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part II*, D. Vrandecic, K. Bontcheva, M.C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L. Kaffee and E. Simperl, eds, Lecture Notes in Computer Science, Vol. 11137, Springer, 2018, pp. 376–394. doi:10.1007/978-3-030-00668-6_23.
[20] G.E. Trouli, A. Pappas, G. Troullinou, L. Koumakis, N. Papadakis and H. Kondylakis, SumMER: Structural Summarization for RDF/S KGs, *Algorithms* **16**(1) (2023), 18. doi:10.3390/a16010018.
[21] G.E. Trouli, G. Troullinou, L. Koumakis, N. Papadakis and H. Kondylakis, SumMER: Summarizing RDF/S KBs using Machine LEaRning, in: *Proceedings of the ISWC 2021 Posters, Demos and Industry Tracks: From Novel Ideas to Industrial Practice co-located with 20th International Semantic Web Conference (ISWC 2021), Virtual Conference, October 24-28, 2021*, O. Seneviratne, C. Pesquita, J. Sequeda and L. Etcheverry, eds, CEUR Workshop Proceedings, Vol. 2980, CEUR-WS.org, 2021.
[22] G. Vassiliou, G. Troullinou, N. Papadakis, K. Stefanidis, E. Pitoura and H. Kondylakis, Coverage-Based Summaries for RDF KBs, in: *The Semantic Web: ESWC 2021 Satellite Events - Virtual Event, June 6-10, 2021, Revised Selected Papers*, R. Verborgh, A. Dimou, A. Hogan, C. d'Amato, I. Tiddi, A. Bröring, S. Maier, F. Ongenae, R. Tommasini and M. Alam, eds, Lecture Notes in Computer Science, Vol. 12739, Springer, 2021, pp. 98–102. doi:10.1007/978-3-030-80418-3_18.
[23] K. Kellou-Menouer, N. Kardoulakis, G. Troullinou, Z. Kedad, D. Plexousakis and H. Kondylakis, A survey on semantic schema discovery, *VLDB J.* **31**(4) (2022), 675–710. doi:10.1007/s00778-021-00717-x.
[24] P.O. Queiroz-Sousa, A.C. Salgado and C.E.S. Pires, A Method for Building Personalized Ontology Summaries, *J. Inf. Data Manag.* **4**(3) (2013), 236–250. https://sol.sbc.org.br/journals/index.php/jidm/article/view/1495.