LaSaS: an Aggregated Search based Graph Matching Approach

Ghizlane ECHBARTHI
Université de Lyon, CNRS
Université Lyon 1, LIRIS, UMR5205, F-69622, France
Email: ghizlane.echbarthi@univ-lyon1.fr

Hamamache KHEDDOUCI
Université de Lyon, CNRS
Université Lyon 1, LIRIS, UMR5205, F-69622, France
Email: hamamache.kheddouci@univ-lyon1.fr

***Abstract*—Graph querying is crucial to fully exploit the knowledge within the widely used graph datasets. However, graph datasets are usually noisy which makes the *approximate graph matching* tools favored to overcome restrictive query answering. In this paper, we introduce a new framework of approximate graph matching based on aggregated search called Label and Structure Similarity Aggregated Search (LaSaS). LaSaS enables effective and efficient graph querying without considering any fixed schema of the data graph by *(i)* using the aggregated search strategy to increase the number of answers, *(ii)* using a lightweight graph similarity metric that takes into account nodes label and graph structure similarity to enable finding approximate matches and also by *(iii)* using a simple graph weight update routine instead of computing the maximum common subgraph which reduces the overall computation cost. We evaluated our proposed approach over the real-life DBpedia graph and results show the effectiveness and stability of the approach on different parameter settings. Moreover, results also show that LaSaS yields more precise matches in a shorter amount of time when compared to state-of-the-art related approaches.**

***Keywords*—Graph querying, Graph matching, Graph similarity metric, Aggregated search.**

I. Introduction

Nowadays, a steep increase in data production is witnessed, which favors the use of graphs as a storage support to fully exploit the knowledge within the dataset. In fact, graphs are a popular data model that enables efficient data processing benefiting from all the graph theory findings and technics. In considering this matter, graph querying has attracted the interest of many researchers as it represents an essential task to exploring the knowledge in these datasets.

In order to query these graph databases, generally, a *graph matching* task is performed using either *graph isomorphism* to find exact answers to the query, or other technics that allows *approximate graph matching*. In the case of graph isomorphism, seeking exact answers to the query can be very expensive as well as restrictive since actual datasets are usually noisy. Moreover, it requires the user to have a complete knowledge of the data structure which is not always the case. To bypass these restrictions, approximate graph matching is widely used in many real life applications such as web anomaly detection [1], search result classification [2] and spam detection [3] to name a few.

However, few works on graph querying have addressed the graph matching problem by building an answer to the query based on the aggregation of heterogeneous graphs. The idea is to find a matching to a given query by combining several subgraphs that when aggregated, they form an approximate match for the query. This search paradigm is known as aggregated search in graphs [4], [5] *i.e.*, given a query graph q and a set B of graphs, aggregated search aims to find matches to q by combining or aggregating subgraphs in B. The aggregated search is different from the classical graph matching problem where all occurrences of the query are to be found in one target graph G. In contrast to the graph context, *aggregated search* is a popular search paradigm in Information Retrieval where an answer to a query is given by combining contents of heterogeneous sources, *e.g.*, text, image, and video instead of giving the classical list of relevant links. The concept of aggregation has rarely been tackled in graph databases, though it represents various advantages compared to the classical graph matching problem. Furthermore, aggregated graph matching represents a powerful matching paradigm, as it brings about significant improvements when considering the number of findable solutions to a given query, where a simple graph matching can not find any. A motivating example for the aggregated search in graph databases is plagiarism detection: the aggregated graph search can detect a plagiarism even when the cheater takes several small shards from different documents and combines them. The aggregated graph search can detect these plagiarism cases.

Generally, most works on graph querying adhere to costly operations such as graph isomorphism [6], [7], or maximum common subgraph search [4], which are too restrictive. Similarly, approximate matching frameworks consist in either strict label similarity [8], or structural similarity [9] which restricts the number of findable solutions. Moreover, to the best of our knowledge, approximate graph matching using the aggregated search paradigm is novel and has never been tackled.

Broadly motivated by addressing these shortcomings, we propose in this paper a new framework for approximate graph matching called Label and Structure Similarity Aggregated Search (LaSaS). The proposed framework enables an effective RDF graph querying without any knowledge of neither the de facto SPARQL language nor the schema of the data graph. Our contribution is threefold: First, we use the aggregated search paradigm to enrich the set of answers. Second, a lightweight graph similarity metric that takes into account both the graph label and graph structure similarity to enable finding approximate matches. Third, we propose a simple graph weight update that replaces the maximum common subgraph search task and reduces the complexity cost.

The remaining of the paper is organized as follows: Section

DOI reference number: 10.18293/SEKE2017-046

2 presents an overview of the related work about graph querying. In Section 3, we present preliminaries and necessary notions for the understanding of our LaSaS method, which is entirely presented in Section 4. Section 5 reports and discusses our experimental results. Section 6 concludes with a summary of our contributions and raises issues for future work.

II. Related Work

Graph querying is a crucial task as most of the actual datasets are being stored and exploited as graphs. Many graph querying techniques exist in the literature and the proposed framework is closely related to aggregated search and approximate (sub)graph matching, the latter being considered as an elementary operation in our matching framework. So in this section, we provide a brief description of the graph querying methods that are based on (sub)graph matching and the aggregated search in graphs.

Subgraph matching. Subgraph matching is a well-studied problem with a rich literature. Two main categories fall under the subgraph matching problem, the *exact* and the *inexact subgraph matching*. Exact subgraph matching finds exact answers to the query via graph isomorphism [7], [6]. Approaches in this group are criticized for their intractability [10], their cost prohibitive characteristic and for being restrictive *w.r.t.* to the number of answers to the query.

On the other hand, inexact graph matching allows slight differences between the query and the matches which is highly suitable for actual needs as graph datasets are usually noisy, and relevant answers could be found using approximate matching. Under the category of inexact (sub)graph matching, we find subgraph matching based on graph simulation [11], [12], which can be determined in quadratic time and is defined as a *relation* between the query nodes and the target nodes. However, it suffers a considerable loss of structural similarity which makes it untailored for many applications such as in bioinformatics.

Another variant of inexact graph matching is the graph homomorphism [13]. The idea is to find mappings to the query such that node labels difference falls under a threshold whereas the query edges are mapped to paths of a given length.

More related to our work, Tian *et al.* proposed a tool called SAGA [14]. The approach allows approximate matching by breaking the query into subgraphs and finding small hits that are assembled to form a match. Although SAGA method is based on an efficient indexing to speed up the processing, it actually reduces to solving the maximum clique problem which represents a costly operation.

It is also worth mentioning that there are parallel works related to the approximate graph matching in bioinformatics. This category of methods includes PathBlast [15], NetAlign [16] and IsoRank [17] to name but a few. These methods are found to be efficient in tasks related to the domain application such protein interaction networks alignment [16], [15].

Besides, a different type of approximate structural matching works has been proposed in [18], [8]. These works are based on concept propagation [19] and spreading activation [20] instead of classical matching schemes (graph isomorphism and similarity metric). Nevertheless, these works consider a strict label node matching, which is still too restrictive.

Although not pursued here, approximate and inexact (sub)graph matching encompasses decades of research work. Hence, we encourage interested readers to [21] as well as references therein for further details about the problem.

Querying RDF graphs. In the realm of querying RDF graphs, SPARQL is a widely used query language. It requires a complete knowledge of the schema of the graph database, *i.e.*, the structure, node labels and types of entities in the graph. Moreover, writing queries in SPARQL proves to be a complicated task that requires users to be familiar with the language. In order to alleviate this, Zou *et al.* studied the problem of answering SPARQL queries via subgraph matching and proposed gStore [9], which allows approximate node label matching but adheres to strict structural matching leading the method to be restrictive.

Aggregated search in graph databases. Aggregated search is a familiar search paradigm in Information Retrieval in the context of documents [22]. However, few works tackled the problem of aggregated search in graph databases. Elghazel *et al.* [4], [5] studied the problem of aggregated search in labeled graphs and their problem formulation is close to ours. Nevertheless, their method looks for exact matches which is restrictive on one hand, and on the other hand very expensive due to the maximum common subgraph search and the maximum clique detection, which are considered as being among the most expensive operations on graphs.

Most of the aforementioned works altogether lack several critical points: (*i*) They present a restrictive tool for query answering either by imposing a strict node label similarity or by a strict structural similarity, which does not allow approximate relevant answers to be discovered, (*ii*) the graph matching is usually done by checking the maximum common subgraph which is a costly computational task. (*iii*) Methods querying semi-structured data requires a complete knowledge of the graph schema and are often complicated. (*iv*) Few works considered a query answering by several graphs in concomitant, and if so, they seek exact matches which is still restrictive. To address these shortcomings, we propose a new graph matching framework that aims to answer a query by allowing approximate label and structural similarity through a lightweight similarity metric which alleviate the task of maximum common subgraph search. Moreover, our proposed framework enables efficient RDF graph querying without having any knowledge about the schema of the graph. Last but not least, our proposed framework is based on the aggregated search to enable a joint query answering by several heterogeneous graphs in order to benefit from the information wealth within these graphs.

III. Preliminaries

This section is dedicated to introducing notions used in our proposed method. First, we give definitions about the usual data structures used: the query, the target graphs and the answer set. Then we introduce a new graph similarity metric and the objective function of the proposed matching scheme, we end this section by giving the problem formulation of aggregated grah search.

A. Query, Target graph, Answer

Query. The query q is an undirected labeled graph $q = (V_q, E_q, \mu_q)$ where V_q represents the vertex set, and E_q the edge set. μ_q is a function $\mu_q : V_q \to L_{V_q}$ that associates labels to vertices, with L_{V_q} the vertex label set.

Target graphs. Target graphs are represented by the set B of p graphs which are referred to as fragments f_i, $i \in [p]$. Fragments $f_i = (V_i, E_i, \mu_i)$ are undirected labeled graphs *s.t.* V_i (resp. E_i) is the vertex set (resp. edge set) of f_i. μ_i is a function $V_i \to L_{V_i}$ associating labels to vertices, with L_{V_i} is the vertex label set of f_i.

Answer set. The expected answer set A is defined as follows: $A = \{a_1, \cdots, a_k\}$ where a_i are called aggregates. We have $a_i = \bigcup_{j=0}^{p_i} f_{i_j}$ *s.t.* p_i is the number of fragments in a_i and we have $p_i \leq p$, *i.e.*, a_i is constituted by as few fragments as possible.

Though our method focuses on undirected labeled graphs, it is straightforward to extend it to process other kinds of graphs.

B. Objective function

Graph similarity metric. We first introduce a graph similarity metric that computes the similarity between two graphs based on two main features: *(i)* the node label similarity and *(ii)* the structure similarity. The similarity function is denoted by $s(f_i, q)$ where $s : B \times \{q\} \to [0, 1]$ is a function that quantifies the similarity that f_i shares with q in terms of label and structure similarity *s.t.* the closest s to 1, the more similar are f_i and q. In the following, we present the components of the similarity metric s: Label similarity and structure similarity components.

1) Label similarity component: Label similarity of fragment f_i and query q is denoted by $\Delta_L(f_i, q)$ and is computed as follows:

$$\Delta_L(f_i, q) = \frac{1}{n_i} \cdot \sum_{v \in V_{f_i}} J(\mu(v), \mu(Q(v))) \quad (1)$$

where n_i is the number of vertices in f_i, and J is the jaccard similarity coefficient such that $J(l_1, l_2) = \frac{W_{l_1} \cap W_{l_2}}{W_{l_1} \cup W_{l_2}}$, with W_{l_1} (resp. W_{l_2}) is the words set in label l_1 (resp. l_2). Q is an application $Q : V_i \to V_q$ that associates vertices from the fragment to their counterpart in the query and we have: $Q(v) = u$ iff $J(\mu_i(v), \mu_q(u)) > \tau$ where τ is a user fixed threshold.

2) Structure similarity component: On the other hand, $\Delta_D(f_i, q)$ denotes the structure similarity between fragment f_i and query q as follows:

$$\Delta_D(f_i, q) = \frac{1}{n_i} \cdot \sum_{u,v \in V_{f_i}, u<v} \delta_t(d(u, v), d(Q(u), Q(v))) \quad (2)$$

where

$$\delta_t(x, y) \begin{cases} 1 & \text{if } |x - y| < t \\ 0 & \text{otherwise.} \end{cases}$$

$d(u, v)$ denotes the distance between vertices u and v. Δ_D computes the structure similarity in terms of distances in f_i and q. Δ_D increases each time the distance between two nodes in the target graph is equal or near to the distance between their corresponding nodes in the query graph. More precisely, Δ_D increases when the distance difference is under a threshold t called the *structure disparity threshold*. The value assigned to threshold t should be small in order to avoid disparate matches and to preserve the structure similarity. Besides, t should not neither be equal to zero otherwise it would be very restrictive as only strikingly similar matches will be favored.

Taking these both components into account, we define the similarity function s as follows:

$$s(f_i, q) = \lambda.\Delta_L(f_i, q) + (1 - \lambda).\Delta_D(f_i, q) \quad (3)$$

where λ is a tunable parameter in $[0, 1]$.

Objective function. Let η denote the matching that associates the aggregates a_i in the answer set A to the query q (matching query nodes to target nodes). We define the objective function of the matching η as follows:

$$\Psi(\eta) = \frac{1}{k} \cdot \sum_{i=0}^{k} s(a_i, q) \quad (4)$$

C. Problem Formulation

Given a set of fragments B, a query q. The aggregated graph search problem consists in finding the matching η that associates the answer set A of k aggregates to q *s.t.* $\Psi(\eta) = 1$.

Proposition 1. *Aggregated graph search problem is NP-Hard.*

Proof: Let us consider the simple case of $k = 1$. That is to say, the expected answer set consists of one aggregate a_1 *s.t.* $a_1 = \bigcup_{j=0}^{p_1} f_j$, where $\Psi(\eta) = 1$, *i.e.*, $s(a_1, q) = 1$, with η is the matching that associates aggregate a_1 to q. By definition, the similarity function $s(a_1, q) = 1$ means that we are looking for minimum elements from B (see section III-A) *s.t.* their union covers q: label similarity component equals one means that all nodes are matched and their labels are exactly similar, on the other hand, structure similarity component should be one, meaning that the query and the aggregate have similar structure. This reduces to resolving the NP-Hard set cover problem where the universe is the query q that we aim to cover with minimum elements from B. This ends the proof. ■

IV. Query Processing Algorithm

In this section, we present our query processing algorithm named LaSaS, which is a heuristic solution intended to solve the problem of aggregated graph search. We expose in details the proposed algorithm and its different steps.

LaSaS algorithm (described in algorithm 1) works in three distinct steps: first, the *Selection step* aims to select the most relevant fragments from the fragment set B based on the similarity metric s (see section III-B). Second, the *Aggregation step* combines the relevant fragments found by the *Selection* step to form the aggregate a. Third, the *Refinement step* enhances the quality of the aggregate by pruning irrelevant nodes and by mapping paths of a thresholded length to unmapped edges if any. Note that this threefold process is necessary for obtaining one aggregate, whereas in the general case of k aggregates, this process will repeat k times. In the following, we present each step in details.

1) Selection step: The first step consists in selecting the most relevant fragments, as many as necessary to cover the whole query q. The selection is based on successive iterations of two main substeps: *(i)* similarity checking and *(ii)* query updating until a stopping condition is verified. In the similarity checking, a rank is associated to each fragment f_i in B which is given by the similarity metric $s(f_i, q)$ (see section

Algorithm 1 *Label and Structure Similarity Aggregated Search*

```
Input: Fragments set $B$, Query graph $q$, number of expected aggregates $k$.
Output: Answer set $A$.
1.  $i = 0$;
2.  while $i < k$
3.      Select potential fragment from B that are similar to $q$;
4.      Add the selected fragments to set $S$;
5.      Aggregate all fragments in $S$ to form an aggregate $a_i$;
6.      Refine $a_i$;
7.        Prune irrelevant nodes;
8.        If there are unmapped edges in $a_i$:
9.          Map paths to missing edges;
10.     $A = A \cup \{a_i\}$;
11.     $i = i + 1$;
12.     $S = \emptyset$;
13. end while
14. return $A$;
```

III-B), then the fragment f_{max} having the maximal similarity is selected and we have $f_{max} = \underset{f_i}{argmax}\ s(f_i, q)$ where $f_i \in B$. The query is then updated according to the best-ranked fragment f_{max} such that the query part that has been covered by f_{max} is withdrawn according to several conditions which will be detailed in the following. When the query is updated, the selection process repeats: similarity checking and query update substeps are performed until one of these stopping conditions are verified: *(i)* $|V_q| < \epsilon$, meaning that almost all query nodes have been matched and/or *(ii)* the fragment set B is empty, given that a fragment is removed from B once chosen during the selection step.

Query Update. The foremost role of query update step is to ensure complementarity among selected fragments. This step updates the query by removing nodes that are covered by the selected fragment, thus enabling subsequent selections to choose fragments covering complementary parts of the query. The query update can be done in two distinct ways: using Maximum Common Subgraph (MCS), and using Weight Update (WU).

Query update using MCS. As aforementioned, the query is updated given the best fragment f_{max}. First, the maximum common subgraph (MCS) of q and f_{max} is computed, then the MCS is withdrawn from q while keeping the boundary nodes that belongs to the MCS as explained in the following.

Let q' denote the updated query q. We have:

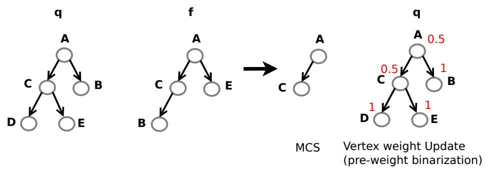
- $V_1(q') = \{v | v \in V_q \ \&\ v \notin MCS(q, f_{max})\}$
- $V_2(q') = \{u | (u, v) \in E_q \ \&\ v \notin MCS(q, f_{max})\}$

The vertex set of q' is $V_{q'} = V_1(q') \cup V_2(q')$, and the edge set is $E_{q'} = \{(u, v) \in V_{q'}^2 \mid (u, v) \in E_q\}$.

Query Update using Weight Update (QUWU). Computing the maximum common subgraph is known to be NP-complete by reduction from the maximum clique problem. Hence, it is cost prohibitive to use it in our method in such a repetitive operation as the query update. Alternatively, we perform a weight update on the query s.t. removable nodes on the query will be weighted by 0 and by 1 otherwise. In a nutshell, QUWU works in 2 steps: First, covered nodes will have their weights set to 0.5 as follows: $\forall (u, v) \in E_{f_{max}}$: if $(Q(u), Q(v)) \in E_q$ then $w(u) = w(v) = 0.5$, where $w(u)$ is the weight of vertex u. Figure 1 depicts the equivalent result to computing the MCS. Second, weight binarization is performed, *i.e.*, among covered nodes, removable nodes will be weighted by 0, otherwise by 1. QUWU process will not be further detailed due to lack of space, however, it is worth mentioning that it has a polynomial time complexity *w.r.t.* the number of query nodes.

Consequently, the stopping condition of the selection step would be that all vertices of q are weighted by 0, *i.e.* $W \leq \epsilon$, where $W = \sum_{v \in V_q} w(v)$.

Fig. 1. Vertex weight update vs. Maximum Common Subgraph.



Since QUWU does not reduce the query size as it does not withdraw vertices but update their weights, it is necessary to tweak the similarity function s (see equations 5 and 6) in order to consider only vertices that are weighted by 1, *i.e.*, not previously covered.

$$\Delta_L(f_i, q) = \frac{1}{n_i} \cdot \sum_{\substack{v \in V_{f_i} \\ w(Q(v)) \neq 0}} J(\mu(v), \mu(Q(v))) \quad (5)$$

$$\Delta_D(f_i, q) = \frac{1}{n_i} \cdot \sum_{\substack{u, v \in V_{f_i}, u < v \\ w(Q(v)) \neq 0}} \delta_t(d(u, v), d(Q(u), Q(v))) \quad (6)$$

2) Aggregation step: The fragments obtained upon successful completion of the selection step are stored in the solution set denoted by S. The aggregation step constructs an aggregate a_i from the solution set S as follows:

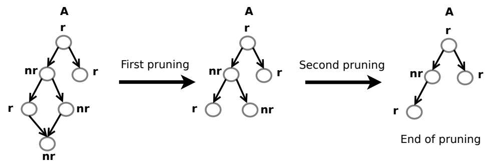
$$V_{a_i} = \bigcup_{f_i \in S} V_i \ \ and \ \ E_{a_i} = \bigcup_{f_i \in S} E_i$$

3) Refinement step: Refinement of aggregate a_i is achieved by *(i)* connecting the aggregate whenever it is disconnected, and *(ii)* by pruning irrelevant nodes from a_i.

Connecting the aggregate. Cases when a_i is disconnected may occur when selected fragments cover disjoint areas of query q leaving some query edges unmapped. Our algorithm maps these edges to paths by performing a path search in B in order to interconnect the connected components (*ccs*) in a_i. To this end, we search for a path between any two nodes belonging to different *ccs* of a_i in the graph G, where $G = \bigcup_{i=0}^{p} f_i$. For an optimal path finding task, we customized the Dijkstra algorithm by adding a constraint thresholding the path length, where only paths having a relatively small length are considered in order to speed up the search time, as long paths search will be abandoned once the threshold exceeded and, on the other hand, to preserve semantic relevance within a_i. Finally, the path search stops once a_i becomes connected.

Pruning. The aggregate a_i is further refined by withdrawing irrelevant nodes. To do so, we iteratively prune irrelevant leaves from a_i until no irrelevant leaf is left, where a leaf is a vertex of degree 1. As depicted in figure 2, not all irrelevant nodes are pruned from A by the end of the pruning process, however, by limiting the pruning to irrelevant leaves, the connectivity of the aggregate is preserved as the removal of an irrelevant node with a degree>1 may disconnect the aggregate.

Fig. 2. Pruning process of the aggregate.



V. Experimental evaluation

This section shows the empirical evaluation of the proposed approach. We first describe the experimental set up along with the graph dataset and the evaluation methodology. Then we present the evaluation metrics. Finally, we present and discuss the obtained results.

A. Experimental set up

Graph Dataset. In order to evaluate our method, we used the real life dataset DBpedia Knowledge Base [23] that consists of RDF triples extracted from Wikipedia. We considered a total of 666043 triples from ten different entities.

Fragments generation. In order to constitute the set of fragments B, we partition the dataset used in such a way that no fragment is isomorphic to the query q. To avoid crossing edges between fragments, nodes belonging to the crossing edges are duplicated in the fragments involved.

Query generation. Two parameters are necessary to generate a query: the number of nodes denoted by n and the query diameter d which represents the largest distance between any two nodes. A query is generated by extracting a subgraph from the dataset and introducing some label and structure noise to it. The label noise is generated by randomly modifying some words in the original label while the structure noise is provided by randomly removing or adding some edges.

Evaluation metrics. The F1-measure is used as the main evaluation metric to show the effectiveness of our method. It combines the recall (R) and precision (P) where the recall shows the ratio of the correctly found node matches overall correct matches, and the precision is the ratio of correctly found matches overall found matches. F1-measure is as follows:

$$F1 = \frac{2}{(1/R + 1/P)}$$

In addition to he F1-measure, the runtime is also reported.

Methodology. Our experiments are based on the following guidelines: We create 4 query sets, and we generate 100 query under each set. First, we assess the influence of the structural noise on our method by fixing the label noise and varying the structural noise for all the 4 query sets. Similarly, to assess the label noise influence on our method, we fix the structural noise and vary the label noise on all the 4 query sets. We vary the ratio of the matching nodes in the query, *i.e.*, the query nodes that are surely present in the fragments set B, and see the repercussion on the performances of our method. Last but not least, we compare LaSaS to state-of-the-art tools: SAGA and BLINKS [24], a keyword graph search method.

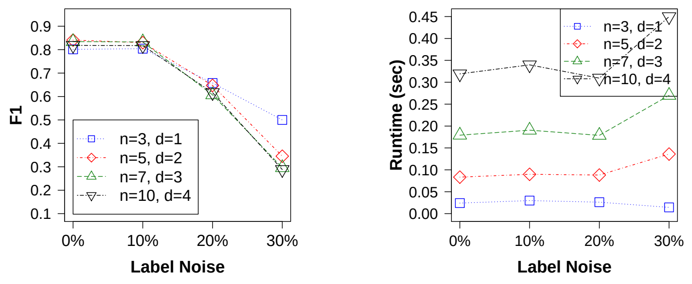
All algorithms have been implemented in C++, and all experiments were performed on a single machine, with Intel Xeon(R) CPU at 1.9GHz, and 16GB of main memory.

B. Experimental results

Label noise influence. We vary the label noise and report the F1-measure in figure 3. Having the structural noise fixed to 10%, F1-measure does not reach 1 when label noise equals 0%. Results show that LaSaS maintains the same performances for 0% to 10% of label noise, meaning that it efficiently discovers the correct matches despite the noise. However, when the label noise goes up to 30%, F1-measure drops considerably, which shows that LaSaS is label noise sensitive.

In figure 3, we show the runtime for all query sets while varying the label noise. Results show that for each query set, the runtime is even for different values of label noise, however, a slight increase of the runtime is noticed when the label noise reaches 30%, which is normal as the method spends additional time to look for other candidate matches. Altogether, increasing the label noise does not incur a considerable computational cost.

Fig. 3. F1-measure and Runtime (in seconds) reported upon four different query sets with 100 query within each set while varying the label noise, and the structural noise was set to 10%.



Structural noise influence. Figure 4 reports results for different values of structural noise while label noise was set to 10%. When no structural noise is added, the F1-measure does not attain 1 due to the label noise being set to 10%. Moreover, the F1-measure does not have an abrupt variation for all query sets (with the lower bound being 0.8 and the upper bound being 0.86), which translates the capacity of LaSaS to find the correct matches despite the added noise on the structure.

Figure 4 reports the runtime while we vary the structural noise, and results show that the runtime is almost steady for all query sets, that is to mention that the computational cost incurred by the added structural noise is negligible.

Ratio of query matching nodes. We refer to the ratio of matching nodes that are already in the fragments set B by Φ, and we investigate its influence on the performances of LaSaS. Figure 5 reports the F1-measure and shows that performances are optimal when the ratio of the matching nodes is equal to 100%, and it decreases gradually when the ratio of the matching nodes decreases. This shows the effectiveness of our method, as it finds the best results when they are available.

The reported runtime as shown in figure 5 increases with the ratio Φ, which means that the method takes more time to process as there are additional relevant answers in the B set.

LaSaS *vs.* SAGA and BLINKS. In Table I, we compare our method LaSaS to two state-of-the-art tools SAGA and BLINKS. Results in table I show that LaSaS outperforms both SAGA and BLINKS in effectiveness by achieving a greater F1-measure and in efficiency by finding results faster.

Fig. 4. F1-measure and Runtime (in seconds) reported upon four different query sets with 100 query within each set while varying the structural noise, and the label noise was set to 10%.

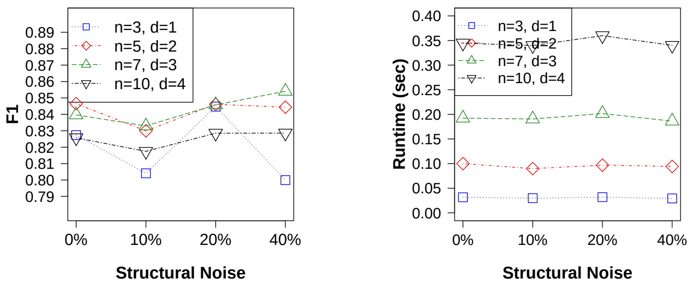
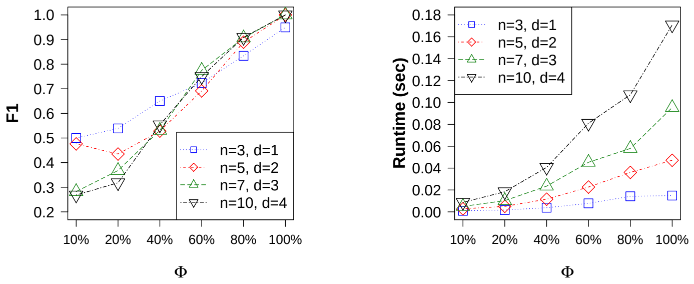


Fig. 5. F1-measure and Runtime (in seconds) reported upon four different query sets with 100 query within each set while varying the matching nodes ratio in the B set.



VI. Conclusion

In this paper, we discussed a novel framework for approximate graph matching based on aggregated search called Label and Structure Similarity Aggregated Search (LaSaS). The proposed approach enables an effective graph querying without any knowledge of the schema of the data graph. The framework joins ideas from aggregated search and graph matching to effectively find approximate matches for a query from a set of heterogeneous graphs. LaSaS is based on three key ideas: *(i)* aggregated search strategy in order to enrich the set of answers *(ii)* a lightweight graph similarity metric that takes into account both the nodes label and graph structure similarity to enable finding approximate matches *(iii)* a graph weight update that replaces the maximum common subgraph search task and reduces the complexity cost. Our method allows approximate matching by allowing slight label difference and by mapping edges to paths with a thresholded length. This feature makes our method unrestrictive and hence enable it to find more answer results compared to existing strict matching schemes like graph isomorphism. Empirical evaluation over the real life DBpedia graph [23] illustrates the effectiveness of our method over different parameter settings and corporates the stability of LaSaS. Moreover, the experimental results indicate that the proposed method outperforms the state-of-the-art related approaches by finding more precise matches. Future works will be conducted to build an index on the query and fragments to further accelerate the process of the selection step. We also plan to extend the empirical study to compare our method to additional matching tools on supplementary graph datasets.

Acknowledgements.

This work is partially funded by the French National Agency of Research project: Contextual and Aggregated Information Retrieval (ANR-14-CE23-0006).

TABLE I. Comparison results of LaSaS, SAGA and BLINKS: F1-measure in terms of nodes and graphs and runtime for finding the best match (top-1) over 100 queries of set 2 (n=5, d=2). Label noise and structure noise where both set to 10%.

	LaSaS	SAGA	BLINKS
F1-measure (nodes)	**0.93**	0.74	0.62
F1-measure (graphs)	**0.9**	0.7	0.58
Runtime (sec)	**0.15**	10.83	1.81

References

[1] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina, "Web graph similarity for anomaly detection (poster)," ser. WWW '08.

[2] A. Schenker, M. Last, H. Bunke, and A. Kandel, "Classification of web documents using graph matching," *IJPRAI*, 2004.

[3] M. Aery and S. Chakravarthy, "emailsift: Email classification based on structure and content," ser. ICDM '05.

[4] H. Elghazel and M.-S. Hacid, "Aggregated search in graph databases: preliminary results," in *International Workshop on Graph-Based Representations in Pattern Recognition*, 2011.

[5] T.-H. Le, H. Elghazel, and M.-S. Hacid, "A relational-based approach for aggregated search in graph databases," in *DASFAA*, 2012.

[6] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, 2004.

[7] H. Shang, Y. Zhang, X. Lin, and J. X. Yu, "Taming verification hardness: An efficient algorithm for testing subgraph isomorphism," *VLDB*, 2008.

[8] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao, "Neighborhood based fast graph search in large networks," in *Proc. of the 2011 ACM SIGMOD*.

[9] L. Zou, J. Mo, L. Chen, M. T. Özsu, and D. Zhao, "gstore: Answering sparql queries via subgraph matching," *Proc. VLDB Endow.*, 2011.

[10] J. R. Ullmann, "An algorithm for subgraph isomorphism," *J. ACM*, 1976.

[11] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu, "Graph pattern matching: From intractable to polynomial time," *VLDB*, 2010.

[12] S. Ma, Y. Cao, W. Fan, J. Huai, and T. Wo, "Capturing topology in graph pattern matching," *Proc. VLDB Endow.*, 2011.

[13] W. Fan, J. Li, S. Ma, H. Wang, and Y. Wu, "Graph homomorphism revisited for graph matching," *Proc. VLDB Endow.*, vol. 3.

[14] Y. Tian, R. C. Mceachin, C. Santos, J. M. Patel *et al.*, "Saga: a subgraph matching tool for biological graphs," *Bioinformatics*, 2007.

[15] B. P. Kelley, B. Yuan, F. Lewitter, R. Sharan, B. R. Stockwell, and T. Ideker, "Pathblast: a tool for alignment of protein interaction networks," *Nucleic Acids Res*, 2004.

[16] Z. Liang, M. Xu, M. Teng, and L. Niu, "Netalign: A web-based tool for comparison of protein interaction networks," *Bioinformatics*, 2006.

[17] R. Singh, J. Xu, and B. Berger, "Pairwise global alignment of protein interaction networks by matching neighborhood topology," in *Proc. of RECOMB'11*.

[18] V. S. Cherukuri and K. S. Candan, "Propagation-vectors for trees (pvt): Concise yet effective summaries for hierarchical data and trees," in *Proc. of the LSDS-IR'08 ACM Workshop*.

[19] J. W. Kim and K. S. Candan, "Cp/cv: Concept similarity mining without frequency information from domain describing taxonomies," in *Proc. of CIKM'15*.

[20] J. R. Anderson, "A spreading activation theory of memory," *Journal of verbal learning and verbal behavior*, 1983.

[21] B. Gallagher, "Matching structure and semantics: A survey on graph-based pattern matching," *AAAI FS*, 2006.

[22] A. Kopliku, K. Pinel-Sauvagnat, and M. Boughanem, "Aggregated search: A new information retrieval paradigm," *ACM Comput. Surv.*, 2014.

[23] "http://dbpedia.org."

[24] H. He, H. Wang, J. Yang, and P. S. Yu, "Blinks: Ranked keyword searches on graphs," in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '07.