Scalable Link Discovery for Modern Data-Driven Applications

Kleanthi Georgala

Universität Leipzig, Institut für Informatik, AKSW,
Postfach 100920, D-04009 Leipzig, Germany,
georgala@informatik.uni-leipzig.de

Abstract. The constant growth of volume and velocity of knowledge bases on the Linked Data Web has led to an increasing need for scalable linking techniques between resources. Modern data-driven applications often have to integrate large amounts of data relaying on fast but accurate Link Discovery solutions. Hence, they often operate under time or space constraints. Additionally, most Link Discovery frameworks rely on complex link specifications to determine candidates for links, in which the scalability of execution is of significant importance. The main focus of our work is the implementation of time efficient and scalable data linking approaches by utilizing Semantic Web technologies. In this work, we address these Link Discovery challenges by presenting a novel approach for time constraint linking, efficient computation and scalable execution of link specifications with applications towards periodically updated knowledge bases.

1 Problem Statement

Over the last years, the Linked Data Web has grown to contain billions of triples distributed over hundreds of knowledge bases (KBs) [**?**]. Recent technological progress in hardware development and network infrastructures have led to the collection of large amounts of data in scenarios as diverse as monitoring industrial plants [**?**], monitoring open SPARQL endpoints [**?**], implementing the Internet of Things (IoT) and Cloud Computing [**?**]. Efficient identification of links between KBs is one of the most important key challenges when creating a linked data set, as reflected by the fourth Linked Data principle.[1]

Current LD frameworks utilize complex link specifications (LSs) to identify links between KBs by implementing a set of independent steps: the initial LS is potentially re-written, then planned and finally executed. Most planners create a static plan for the initial LS, by using a set of cost functions to estimate the runtime of the LS. However, this linear process towards the execution of a LS is lacking of one important aspect that influences significantly the performance of the LD framework: the execution engine knows more about the runtimes of a LS than the planner itself.

Furthermore, the increasing need for scalable and time-efficient linking approaches comes with the cost of completeness. In real-time applications, such as Linked-Data-driven manufacturing infrastructures and complex-event-processing (CEP), the data

[1] http://www.w3.org/DesignIssues/LinkedData.html

changes rapidly and events are updated periodically. As a result, LD and CEP rule-based frameworks must complete their learning and update process within a strict time-frame. Additionally, performing accurate LD with time constrains has been addressed by various machine learning algorithms. Herein, the main challenge is identifying efficiently the appropriate set of link candidates for training the machine learning algorithms.

In this work, we present a novel approach for scalable LD for Modern Data-Driven Applications, focusing on time efficient approaches towards the execution of link specifications with respect to time constraints pertaining to the expected recall of the LD task. In contrast to the state-of-the-art approaches, we transform the LS execution task into a dynamic process, where the planner re-plans a LS using information provided by the execution engine. Furthermore, we propose the (to the best of our knowledge) first partial-recall LD approach by computing a portion of the links returned by a LS efficiently while achieving a guaranteed expected recall.

2 Relevancy

Scalable execution of LSs and time-efficient computation of links are able to provide solutions towards key issues of the scientific community and industry. The constant growth of Semantic Web technologies has led to a quadratic evolution rate of Linked data sets. For example, data sets such as *LinkedGeoData*[2] have grown to more than 30 billion triples over the years. Additionally, independent providers are constantly publishing data sets that pertain to related or even the same real-world entities. Linking and integrating those constantly increasing KBs is a task that requires both fast and efficient LD techniques.

Additionally, identifying an appropriate set of link candidates is a major challenge for machine learning algorithms for LD. Supervised and semi-supervised learning algorithms often have to train under time or space constrains, or in the absence of a full set of training examples. Therefore, our proposed method for sampling links can be proven beneficial for such algorithms. Another important research area that our approach can be applied to is rule-based CEP. The increased amount of sensor data along with its heterogeneous nature require scalable and flexible methods. The benefits for rule-based CEP models are similar to the benefits of the machine learning algorithms, since these approaches focus on detecting and classifying new events based on the knowledge obtained from classifiers on the existing sensor data.

3 Hypotheses and Research Questions

This thesis includes the following hypotheses and formulated research questions involving fast and scalable LD:

(H_1): Given an initial LS, a recall constrain and a refinement time constrain, a presumable subsumed LS will achieve a lower execution runtime compared to the initial LS while complying to the input recall limitation. (Q_1): Will the subsumed LS be more efficient that the initial LS, in terms of time complexity? (Q_2): How will the different

[2] `http://linkedgeodata.org`

values of the predefined recall constrain influence the performance of the algorithm? (Q_3): To which extent will the different values of the refinement time constrain influence the overall runtime of our approach? (Q_4): How much will the sampling of links influence supervised machine learning for LD?

(H_2): The execution engine knows more about the runtimes of a LS than the planner itself, therefore by introducing a dynamic flow of information between the engine and the planner, a LD framework will execute a LS faster. (Q_1): Will the overall execution time of LS be significantly improved by dynamic planning? (Q_2): How much time will the dynamic planner spend planning? (Q_3): How will the different sizes of a LS influence the overall performance of the dynamic planner? (Q_4): How does the dynamic planning algorithm perform compared to the state-of-the-art static approaches?

4 Proposed Approach

In this section we present the main idea behind our approach for fast and scalable LD. We begin by giving a brief but formal description of the fundamental concepts of LD. Then, we explain in detail our approach for dynamic planning and ensuring selectivity when executing a link specification.

4.1 Preliminaries

A knowledge base K is a set of triples $(s, p, o) \in (\mathcal{R} \cup \mathcal{B}) \times \mathcal{P} \times (\mathcal{R} \cup \mathcal{B} \cup \mathcal{L})$, where $\mathcal{R}$ is the set of all RDF resources, $\mathcal{P} \subseteq \mathcal{R}$ is the set of all RDF properties, $\mathcal{B}$ is the set of all RDF blank nodes and $\mathcal{L}$ is the set of all literals. A LD framework aims to compute the set $M = \{(s,t) \in S \times T : R(s,t)\}$ where S and T are sets of RDF resources and R is a binary relation. Given that M is commonly difficult to compute directly, LD frameworks commonly compute an approximation $M' \subseteq S \times T \times \mathbb{R}$ of M by executing a LS. An *atomic LS* L is a pair $L = (m, \theta)$, where m is a similarity measure that compares properties of pairs (s,t) from $S \times T$ and θ is a similarity threshold. LS can be combined by means of operators and filters. Here, we consider the binary operators $\sqcup$ (union), $\sqcap$ (intersection) and $\backslash$ (difference). Filters are pairs (f, τ), where (1) f is either empty (denoted ϵ) or a combination of similarity measures by means of specification operators and (2) τ is a threshold. A *complex* LS L is a triple $(f, \tau, \omega(L_1, L_2))$ where ω is a specification operator, (f, τ) is a filter, L_1 and L_2 are the *left* and the *right child* of L resp. Note that an atomic specification can be regarded as a filter (f, τ, X) with $[[X]] = S \times T$. We call (f, τ) the filter of L and denote it with $\varphi(L)$. We denote the operator of a LS L with $op(L)$. For $L = (f, \tau, \omega(L_1, L_2))$, $op(L) = \omega$. We denote the semantics (i.e., the results of a LS for given sets of resources S and T) of a LS L as $[[L]]$ and call it a *mapping*.

4.2 Approach

As we have introduced in Sect. **??**, our hypothesis consists of two parts: 1) discover a presumable subsumed LS L in order to partially compute the links returned by the initial LS efficiently, while achieving a guaranteed expected recall and 2) dynamic planning that changes the existing static infrastructure of execution in order to further improve the runtime of the subsumed LS.

Linking with Guaranteed Partial Recall. Given an initial LS L_0 , the main goal of our approach is to discover a subsumed LS L from L_0, that will be given as input to our dynamic execution infrastructure and once it will be executed the result mapping will be a portion of the links retrieved by L, achieving a guaranteed expected recall. Our approach relies on a refinement operator, which allows exploring the space of potential solutions to this problem efficiently.

Definition 1 (Subsumption of Specifications). *The LS L' is subsumed by the LS L (denoted $L \sqsubseteq L'$) when $[[L]] \subseteq [[L']]$ for all possible S and T.*

A key observation that underlies our approach is that if the interpretation of $L_1 = (m(p_s, p_t), \theta)$ and $L_2 = (m(p_s, p_t), \theta')$ are carried out on the same sets S and T, then the following holds:

Proposition 1. $\forall \theta, \theta' \in [0, 1]\ \theta > \theta' \rightarrow (m(p_s, p_t), \theta) \sqsubseteq (m(p_s, p_t), \theta')$.

Proposition 2. *$\sqsubseteq$ is a quasi-ordering (i.e., reflexive and transitive) on $2^{\mathcal{LS}}$.*

Definition 2 (Refinement Operator and Properties). *In the quasi-ordered space $(\mathcal{LS}, \sqsubseteq)$, we call any function f: $\mathcal{L} \rightarrow 2^{\mathcal{LS}}$ an (LS)* operator. *A* downward refinement operator *ρ is an operator such that for all $L \in \mathcal{LS}$ we have $L' \in \rho(L)$ implies $L' \sqsubseteq L$. L' is called a* specialisation *of L. We denote $L' \in \rho(L)$ with $L \rightsquigarrow_\rho L'$. A refinement operator r over the quasi-ordered space $(S, \preccurlyeq)$ adibes by the following criteria: (1) r is finite iff $r(s)$ is finite for all $s \in S$. (2) r is proper if $\forall s \in S, s' \in r(s) \Rightarrow s \neq s'$. (3) r is said to be complete if for all s and s', $s' \preccurlyeq s$ implies that there is a s'' with $s'' \preccurlyeq s' \wedge s' \preccurlyeq s''$ such that a refinement chain between s'' and s exists. (4) A refinement operator r over the space $(S, \preccurlyeq)$ is redundant if two different refinement chains can exist between $s \in S$ and $s' \in S$.*

We define our refinement operator over the space $(2^{\mathcal{LS}}, \sqsubseteq)$ as follows:

$$\rho(L) = \begin{cases} \emptyset & \text{if } L = L_\emptyset, \\ L_\emptyset & \text{if } L = (m(p_s, p_t), 1), \\ (m(p_s, p_t), next(\theta)) & \text{if } L = (m(p_s, p_t), \theta) \wedge \theta < 1, \\ (\rho(L_1) \sqcup L_2) \cup (L_1 \sqcup \rho(L_2)) & \text{if } L = L_1 \sqcup L_2, \\ (\rho(L_1) \sqcap L_2) \cup (L_1 \sqcap \rho(L_2)) & \text{if } L = L_1 \sqcap L_2, \\ (\rho(L_1) \backslash L_2) & \text{if } L = L_1 \backslash L_2. \end{cases} \quad (1)$$

In words, our operator works as follows: If L *is the empty specification* $L_\emptyset$, then we return an empty set of specifications, ergo, L is not refined any further. If L *is an atomic specification* with a threshold of 1, our approach returns $L_\emptyset$. By these means, we can compute refinement chains from $L = L_1 \sqcup L_2$ to L_1 and L_2. If $\theta < 1$, our approach alters the threshold θ by applying the $next$ function. Formally, for a given set on input data sets S and T and any θ, $next$ always returns values from $N(m(p_s, p_t)) = \{n : \exists (s, t) \in$

$S \times T : n = m(p_s, p_t)\} \cup \{\varnothing\}$. Given a threshold θ, $\varnothing$ is returned if L is to be refined to the empty specification. Else, $next$ returns the smallest value from $N(m(p_s, p_t))$ that is larger than θ. Note that $(m(p_s, p_t), next(\theta)) \sqsubseteq (m(p_s, p_t), \theta)$ always holds. If L *is complex*, then the refinement depends on the operator of the specification. To explicate the set of LS returned by ρ, we extend the semantics of $op(\rho(L_1), L_2)$ resp. $op(L_1, \rho(L_2))$ to be the set of all specifications that can be computed by using *op* on all $L' \in \rho(L_1)$ resp. $L' \in \rho(L_2)$. If $op = \sqcap$ or $op = \sqcup$, then ρ returns the union of all specifications that can be generated by apply ρ to one child of L and combining these with the other child. If $op = \backslash$, then we combine L's right child with all refinements of L's left child. The reason for which we do not do this the other way around for this particular operator is simply ρ would not be a refinement operator if we did so, as we could not guarantee that $\rho(L) \sqsubseteq L$.

In our initial implementation, our algorithm (*C-RO*) takes as input the desired expected recall k, where $k \in [0, 1]$, and a refinement time constrain $maxOpt$, then estimates the selectivity of L_0 using an oracle and computes the desired selectivity as its fraction using k. The approach starts by initialising a refinement tree with the given LS L_0 and proceeds on selecting the previously unvisited LS of the tree that (1) has the lowest expected run time and (2) abides by the settings provided by the user. Our algorithm computes the whole refinement of this LS by virtue of ρ's finiteness. Redundant refinement results are subsequently detected (as ρ is redundant) and not added into the tree. Our algorithm terminates if the refinement tree has been explored completely or the time threshold for running the refinements is met. Additionally, we have implemented variation on the approach that makes uses of the potential monotonicity of the run times of specifications (*RO-MA*).

Dynamic Planning. The basic idea behind our novel planning approach, is to construct a plan derived from a LS with small expected runtime in a non-linear fashion. Our method incorporates a cost function $cost$, which approximates the expected runtime of a plan P of a LS. The basic insight behind the approach is that if P corresponds to a specification L that is complex, i.e., $L = (f, \tau, \omega(L_1, L_2))$, then running a plan for L_1 or L_2 can help overwriting the initial approximation of the cost function with a better cost approximation and thus lead to the plan for L to be reshaped and made more efficient. Additionally, through the constant information flow of information between the engine and the planner, our method re-uses previous result sets, ensuring that duplicated steps are executed exactly once. Our existing implementation consists of two basic functions: *plan* and *execute*.

Plan Given a LS L as input, the *plan* function returns a plan $P(L)$ with the smallest expected cost based on current $cost(P)$ function. Our *plan* function distinguishes between executed and non-executed LSs. Therefore, if a LS has been executed previously, it will not be planned again. Then, if L is atomic, its plan will consist of a simply run command. If $L = (f, \tau, \omega(L_1, L_2))$ then, the algorithm will derive plans for L_1 and L_2 sequentially, compute possible plans for L and then decide the least costly plan. At this point, the *plan* function discriminates the possible plans for L given its operator. If $op(L) = \sqcup$, then $P(L)$ will consist of executing the plans for L_1 and L_2 and then perform union between the resulting sets. If $op(L) = \sqcap$ or $\backslash$, the algorithm can choose between two alternatives 1) executing both plans for L_1 and L_2 and then perform the corresponding set operation or 2) execute the plan of one of the children and use the other one as a filter on the resulting mapping. The final plan for L will be chosen based the cost function. For further optimization, if both children are executed, then the algorithm will be forced to choose the first alternative, since the estimated cost of this plan will be 0. In case one of the child LS is executed, then this child cannot be used a filter operator and the possible set of plans change accordingly.

Execute Similarly to the *plan* function, *execute* takes as input a LS L and returns the corresponding mapping M. Initially, the execution engine is informed from the planner whether or not the plan of a LS has been executed and retrieves the cached set of links. In case of a non-executed LS, *execute* checks whether a LS L' with $[[L]] \subseteq [[L]]'$ has already been executed, retrieves the set of links and performs a filtering function using $(f, \tau) = \varphi(L)$. If such LS does not exist, then the algorithm discriminates between atomic and complex LSs. In case of a atomic LS, then the engine executes the corresponding plan returned by the *plan* function. In case of a complex LS, then the algorithm, calls the *plan* function, executes the first sub-plan assigned to $P(L)$, re-plans the remaining steps of L by invoking again the *plan* function and then proceeds in executing the second sub-plan of $P(L)$, that can vary given from the initial given the intermediate executed steps of the first plan.

5 Evaluation Plan

In order to test our hypotheses and research questions in practice, we will use three sets of datasets: (1) the first set consists of a set of benchmark datasets, Abt-Buy, Amazon-Google Products, DBLP-ACM and DBLP-Scholar described in [**?**], and (2) the second set of data (MOVIES, TOWNS and VILLAGES) was constructed in order to test the scalability of our methodology using real the data sets DBpedia, LinkedGeodata and LinkedMDB. [3] All LSs used during our experiments will be generated automatically by the unsupervised version of the genetic-programming-based machine learning approach EAGLE [**?**].

For the partial-recall algorithm, we plan to report the overall runtime of our partial-recall algorithm (the initial approach ***C-RO*** and its alternative ***RO-MA***) along with the *baseline* method of running the original LS and compare their performance in terms of time efficiency. Additionally, we will exploit performance of our method for the different values of (1) the desired expected recall k and (2) the maximum time ($maxOpt$) for finding a subsumed LS. Finally, we would like to study the loss of F-measure of a machine-learning approach when presented with the results of partial-recall link discovery in comparison with the F-measure it would achieve using the full results. For the dynamic planning approach CONDOR, we will compared the execution time of our dynamic method with that of the state-of-the-art CANONICAL and HELIOS planners as implemented in LIMES [**?**]. Note that the CANONICAL planner serves as the baseline method since it incorporates the static, linear execution infrastructure.

[3] `http://www.linkedmdb.org/`

6 Preliminary Results

Figure **??** presents preliminary results of applying our methods for scalable LD on the DBLP-ACM dataset. The left plot includes execution time results for the *baseline*, *C-RO* and *RO-MA* for $k = 0.2$ and $maxOpt = 1.6\,s$. The right plot includes comparison of runtimes of CANONICAL, HELIOS and CONDOR. In both plots, the x-axis represents the number of specifications, the y-axis represents the cumulative execution times in seconds.

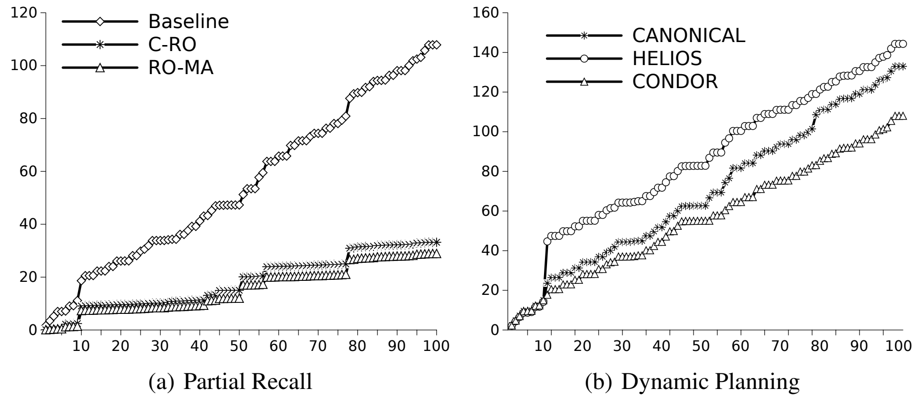

Fig. 1. Preliminary Results for DBLP-ACM

7 Related Work

Over the last few years, a large number of frameworks such as SILK [**?**], LIMES [**?**] and KnoFuss [**?**] were developed to address the scalability of solutions to link discovery and rely on scalable approaches for computing simple and complex specifications. For example, the SILK framework implements MultiBlock [**?**], a multi-dimensional blocking approach. KnoFuss [**?**] on the other hand implements classical blocking approaches derived from databases. Zhishi.links [**?**] is another framework that scales (through an indexing-based) approach. These approaches are not guaranteed to achieve result completeness. Such theoretical and practical guarantees of completeness and efficiency are given by the LIMES framework. LIMES reduces the time-complexity of the LD procedure by combining techniques such as *PPJoin+* [**?**] and HR^3 [**?**] with set theoretical operators and planning algorithms. The problem of identifying appropriate LSs using machine learning techniques has been explored in various previous papers that focus on minimizing the need of the training examples. *EAGLE* [**?**], *RAVEN* [**?**], *AGP* [**?**] are some of the approaches that request less labeled examples but maintain a high level of accuracy. The only method that focus on identifying informative link candidates for training is *COALA* [**?**].

8 Conclusions and Future Work

In this thesis, we present an approach for fast and scalable LD for Data-Driven applications using Semantic Web technologies. Our contribution will be two-fold: (1) the first partial-recall LD approach using a refinement operator and insights pertaining to the subsumption of LS in order to detect LS with guaranteed expected recall efficiently and (2) a dynamic planning with sumbsumptions and result caching. During the completion of this PhD, we are aiming to investigate and evaluate the hypotheses and research question that we proposed, by comparing our method with the state of the art.