ALEX: Automatic Link Exploration in Linked Data

Ahmed El-Roby
University of Waterloo
aelroby@uwaterloo.ca

Ashraf Aboulnaga
Qatar Computing Research Institute, HBKU
aaboulnaga@qf.org.qa

***Abstract*—There has recently been an increase in the number of RDF knowledge bases published on the Internet. These rich RDF data sets can be useful in answering many queries, but much more interesting queries can be answered by integrating data from different data sets. This has given rise to research on automatically linking different RDF data sets representing different knowledge bases. This is challenging due to the scale and semantic heterogeneity of these data sets. Various approaches have been proposed, but there is room for improving the quality of the generated links.**

In this demonstration, we showcase `ALEX`, a system that aims at improving the quality of links between RDF data sets by using feedback provided by users on the answers to linked data queries. `ALEX` starts with multiple RDF data sets that are linked using any automatic linking algorithm. `ALEX` enables the user to issue queries that integrate data from different data sets, and to provide feedback on the answers to these queries. `ALEX` uses this feedback to eliminate incorrect links between the data sets and discover new links. In this demonstration, we show `ALEX` in action on multiple data sets from the Linked Open Data cloud.

I. Introduction

Recent advances in the field of information extraction have helped in automating the construction and publishing of large RDF knowledge bases, which can span different domains, or be domain-specific. Publishing these data sets is part of the Semantic Web vision of making the web accessible and processable by machines [1]. In addition to publishing these data sets, linking them to each other is vital to exploit their rich semantics [3]. When these data sets are linked, it is possible to answer queries that cannot be answered using one RDF data set alone. For example, consider the query "Find all New York Times articles about the 44^{th} president of the United States". Articles about people are available from the New York Times RDF knowledge base. However, we need to know who the "44^{th} president of the United States" is, since this information cannot be found in the New York Times data set. Another data set like DBpedia could have the information that "Barack Obama" is the "44^{th} president of the United States". One can use the Web Ontology Language (OWL)[1] to define an *owl:sameAs* relation linking the two entities representing "Barack Obama" from both data sets. This relation indicates that the two entities refer to the same individual, and enables the system to return all articles about "Barack Obama" from the New York Times data set.

Some work has been done on automatically linking equivalent entities from different data sets (e.g., [2], [6]). That work aims to automatically introduce *owl:sameAs* links between two data sets. However, automatic linking approaches are best effort in nature, with no guarantees on output quality. They try to infer semantics automatically based on syntax, which is a difficult task in the absence of human guidance. As such, automatic linking of RDF knowledge bases can greatly benefit from user feedback on the quality of the generated links.

In this demonstration, we showcase `ALEX` (Automatic Link Exploration in Linked Data) [5], a system that improves the quality of links between linked data RDF data sets by utilizing feedback that users provide on answers to their queries. `ALEX` allows its users to issue SPARQL queries over multiple RDF data sets that are linked using *any* automatic linking algorithm. If the answer to a query is produced using multiple data sets, `ALEX` accepts feedback from the user on this query answer. The user can either approve or reject the answer, and `ALEX` interprets this feedback as approving or rejecting the link used to produce this answer. `ALEX` removes incorrect links rejected by the user, but the main focus of `ALEX` is to find new links that are *similar* to the links approved by the user. `ALEX` uses a stochastic machine learning technique that generalizes the feedback provided by the user and is resilient to errors in that feedback (approving a wrong answer or rejecting a correct answer), whether the errors in user feedback are due to errors in the data or errors made by the user.

II. Overview of ALEX

A. User Feedback Model

`ALEX` starts with a set of automatically generated links (referred to as *candidate links*) that can be produced using any automatic linking algorithm. Federated queries are issued over the linked data. A federated query is a query whose answer is not available in one data source, but can be answered using multiple data sources. The query is decomposed into subqueries for each data source involved. However, this fact is hidden from the user who issues the query as if all the data is in one RDF graph. Answers are retrieved using a federated query processing system (e.g., FedX [8]). A user then gives her feedback on the returned answers. The feedback is as simple as approving or rejecting the returned answer. In this demonstration, we assume that the feedback provided by users is correct. One way to ensure correct feedback is to use feedback that is aggregated over a period of time from multiple users [4], [7]. However, in all cases we cannot assume the feedback to be perfect, so `ALEX` also has techniques to recover from incorrect feedback [5].

B. Link Representation

An entity in an RDF data set is represented by a URI. Each entity has a set of attributes (RDF predicates), and values corresponding to these attributes (RDF objects). We represent a link between two linked entities from different

[1] http://www.w3.org/2001/sw/wiki/OWL

data sets by a set of *features*, sf, made up of the attributes of the two entities. For example, consider the two entities E_1 and E_2 with n and m attributes, respectively, $E_1 = \{(p_{11}, o_{11}), (p_{12}, o_{12}), \ldots, (p_{1n}, o_{1n})\}$, and $E_2 = \{(p_{21}, o_{21}), (p_{22}, o_{22}), \ldots, (p_{2m}, o_{2m})\}$, where each attribute is the pair (predicate label, predicate value). An example entity is {(name, "Barack Obama"), (birth date, 1961), (age, 53), $\cdots$}. We first construct a similarity matrix between the two entities using a similarity function that returns a score in the range $[0, 1]$. An element in the similarity matrix is $((p_{1x}, p_{2y}), score)$ where p_{1x} is a predicate from the first data set, p_{2y} is a predicate from the second data set, and $score$ is the similarity score between the objects associated with these predicates, $score = sim(o_{1x}, o_{2y})$. Scores that are less than a specific threshold are discarded. ALEX uses a generic similarity function that depends on the type of the attributes to be compared (string, integer, float, date, etc.). The link feature set sf is then constructed by choosing the maximum value for each row in the similarity matrix if $n > m$ or each column if $m > n$.

C. Actions in ALEX

ALEX represents each link as a *state* starting from which it takes an action after receiving user feedback. In the case of negative feedback, the wrong link is removed from the set of candidate links. In the case of positive feedback, the action of ALEX can be viewed as an exploration of an area surrounding the current state (the link between two entities) in a particular direction (one feature of the feature set). ALEX explores links in a space that contains feature sets. This space is populated in a pre-processing step, with a feature set for every pair of entities in the two data sets. Given a state that is represented by a feature set of n features, the action a is also a feature set af of n features with a single non-zero feature that represents the offset by which ALEX should explore and retrieve links. Formally, ALEX finds all the links that have similarity value between sf and $sf \pm af$. For example, consider the feature set $sf(E_1, E_2)$ ={((*label*, *name*), 0.8), ((*birth*, *year*), 0.6), ((*age*, *year*), 0.4)}. The feature set means that the predicate *label* from the first entity maps to the predicate *name* from the second entity, and the similarity score between the predicate values is 0.8. A possible action can be represented by the action feature set $af(E_1, E_2)$ ={((*label*, *name*), 0.05), 0, 0}. Links that have a similarity score between attributes *label* and *name* in the range $[0.75, 0.85]$ should be added to the set of candidate links for future queries and possible feedback opportunities.

D. Reinforcement Learning

An important question that ALEX needs to answer is: *Which feature to explore around for a given approved link?*. Exploring around a random feature is not effective since it incorrectly assumes that all features are of equal importance in determining whether the entities are equivalent. At the same time, ALEX has no prior knowledge of which features may be important, and the best feature to explore around can depend on the link being explored. Thus, ALEX needs a way to identify the feature to explore around for any approved link between any two entities.

We propose that this problem can be solved using *Monte Carlo reinforcement learning* methods [10], where the system can *learn* which feature to explore around for different links. Using the terminology of reinforcement learning, ALEX learns from interacting with users in order to identify the best action to take (choosing a feature to explore around with some exploration distance) in order to maximize reward (positive user feedback). The feature is chosen by ALEX using a stochastic policy that is iteratively improved.

E. Policy Evaluation

The feedback given by a user over any link is translated into a reward for the state-action pairs that led to exploring that link. Recall that a state is a link, and an action is exploring around this link in the direction of a chosen feature. This reward is positive in the case of an approved link (positive feedback), and negative in the case of a rejected one (negative feedback). The value of the reward can be the same for positive or negative feedback, or negative feedback can be penalized more.

Initially, there is no specific policy that ALEX follows, so it chooses arbitrary actions whenever a state (i.e., link) is encountered because there is no prior knowledge of what actions to take. Rewards are collected for each state-action pair encountered, and are aggregated to estimate the value of the state-action pair. The value of the state-action pair is the total reward (positive feedback - negative feedback) that can be collected from this state-action pair, taking into consideration the states that are likely to be encountered in the future when starting from this state. This is called *policy evaluation* in the terminology of reinforcement learning. Since ALEX does not have a model of the universe that tells it the estimated value of an action, it uses a Monte Carlo method to estimate the value of state-action pairs.

F. Policy Improvement

Policy evaluation takes place until a predetermined number of feedback items is collected. We call this a feedback *episode*. At the end of an episode, *policy improvement* takes place. Policy improvement is an important step in reinforcement learning, in which ALEX modifies the policy so that actions that maximize the reward are taken most of the time, while sometimes taking a sub-optimal action in order to explore the space of possible actions. Such an exploration step is taken with some low probability, and it is important in order to ensure continuous exploration.

G. Iterative Improvement

These last two steps of policy evaluation and policy improvement are repeated until convergence. ALEX converges when the set of candidate links does not change after an iteration of policy evaluation - policy improvement or when a maximum number of iterations is reached.

More details about ALEX can be found in [5]. That paper presents more on the nature of the interaction between policy evaluation and policy improvement, a proof of the soundness of ALEX, and a description of the optimizations implemented by ALEX to speed up its execution.

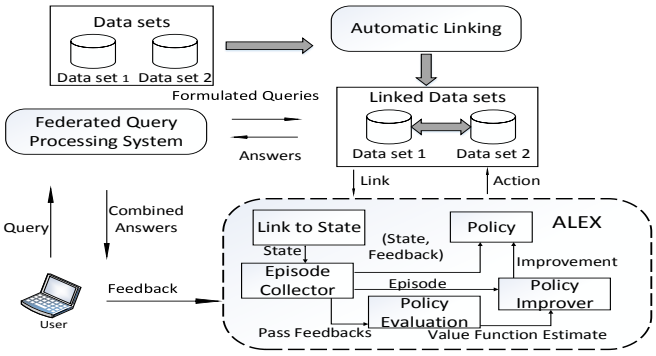

Fig. 1. Architecture of a federated query system with ALEX.

H. System Architecture of ALEX

Figure 1 shows the architecture of ALEX, in which a user issues a query over RDF data sets that are linked using some automatic linking algorithm. The query is answered by a federated query processing system that is able to answer queries that span multiple data sources. ALEX currently uses FedX [8] as the federated query processing system. The user can give feedback on the answer to her query. ALEX collects feedback items into an episode. ALEX also uses its current policy that is learned using the reinforcement learning algorithm to take an action based on the current state (link). After an episode of feedback is collected, the policy used is improved and the new policy is used in the next episode. We emphasize that a user is not *required* to provide feedback on each query answer; if no feedback is provided on an answer, this answer will simply not trigger an action by ALEX.

III. Demonstration Scenario

ALEX has been evaluated in two modes [5]: 1. Batch Mode: A service provider (such as, say, Google) can give users the ability to query multiple, large linked data RDF data sets. In this setting, the service provider collects feedback from many users over a large number of links between most parts of the data sets. ALEX applies the feedback in batches, using a large episode size, in order to ensure that there is sufficient feedback over different parts of the data sets. In this setting, a typical episode size is 1000 (e.g., 1000 users providing 1 feedback item each). 2. Specific Domains: Individual users can develop applications that target more specific domains, either small data sets or subsets of large data sets. The user feedback is focused on a specific domain (e.g., a small part of the DBpedia data set), and the user expects to see quick improvement in link quality based on her feedback. In this setting, a typical episode size is 10. The specific domains mode is more suitable for interactive use of ALEX, so it is the one that we use in this demonstration.

Demonstration participants will be able to use ALEX to issue queries on multiple data sets and observe their answers. Participants will be able to accept or reject query answers, and observe the links removed and added by ALEX based on their feedback. Participants can also see the effect of the changes made by ALEX to the set of links on query answers, and observe how ALEX improves the quality of links.

In the demonstration, ALEX will be used to query two pairs of data sets: The first pair is DBpedia and NYTimes. DBpedia contains structured data extracted from Wikipedia. NYTimes contains data about locations, people, and organizations. A subset of DBpedia about presidents of the United States of America has been extracted and linked to the NYTimes data set. It would have been possible to use the entire DBpedia data set linked with NYTimes in this demo. The problem with this approach is that DBpedia is a large data set and feedback could be dispersed over different parts of the data set. In this case, ALEX would not be able to show noticeable improvement in the span of a few queries issued by a demonstration participant. Using only a subset of DBpedia makes it more likely that demo participants will see noticeable improvement in quality. The second pair of data sets is DBpedia and Semantic Web Dogfood. Semantic Web Dogfood contains data about conferences and workshops about the Semantic Web.

For the purpose of the demonstration, we have carefully linked each of the two pairs of data sets using a combination of (a) links that already exist in the Linked Open Data cloud, (b) automatic linking tools (specifically, PARIS [9]), and (c) manual effort. This exercise gives us a set of ground truth links between each pair of data sets. Links obtained by the user using ALEX can be compared against these ground truth links in two ways: (a) precision, recall, and F-measure can be computed over the set of links obtained by ALEX by comparing these links to the ground truth, and (b) the user query can be issued against the ground truth links to compare the answer to the query answer obtained when the query is issued over the links discovered by ALEX.

The demonstration participants will have three methods to interact with ALEX:

- The user can write a SPARQL query for ALEX to answer. ALEX can answer SPARQL queries that involve multiple data sets and return the answers to the user, who can give her feedback over any returned answer. Figure 2 shows an example of this method. The figure shows an example query that is interested in Barack Obama's party affiliation and the NYTimes news pages about him. A user can click on any of the returned links to see the corresponding entity. After returning the answer, ALEX asks the user for her feedback about whether the answer is correct.
- ALEX can explicitly show the user current existing candidate links. The user can choose an entity and see the links to this entity one at a time, approving or rejecting each link. Alternately, ALEX can show the user one link at a time at random, regardless of the linked entities. The user can view the two entities from the different data sets and decide whether they represent the same real-world entity or not. Figure 3 shows an example of this method.
- For the purpose of the demonstration, ALEX can suggest a set of SPARQL queries for the user to choose from, based on a set of predefined query templates. ALEX generates these queries ensuring that they involve the current set of candidate links. These queries are dynamic in the sense that they change according to the change in the set of candidate links. For example, in a similar fashion to the query in Figure 2, ALEX can generate a similar query for each entity in DBpedia that is linked to an entity in NYTimes. That is, finding party affiliation and the NYTimes news pages about

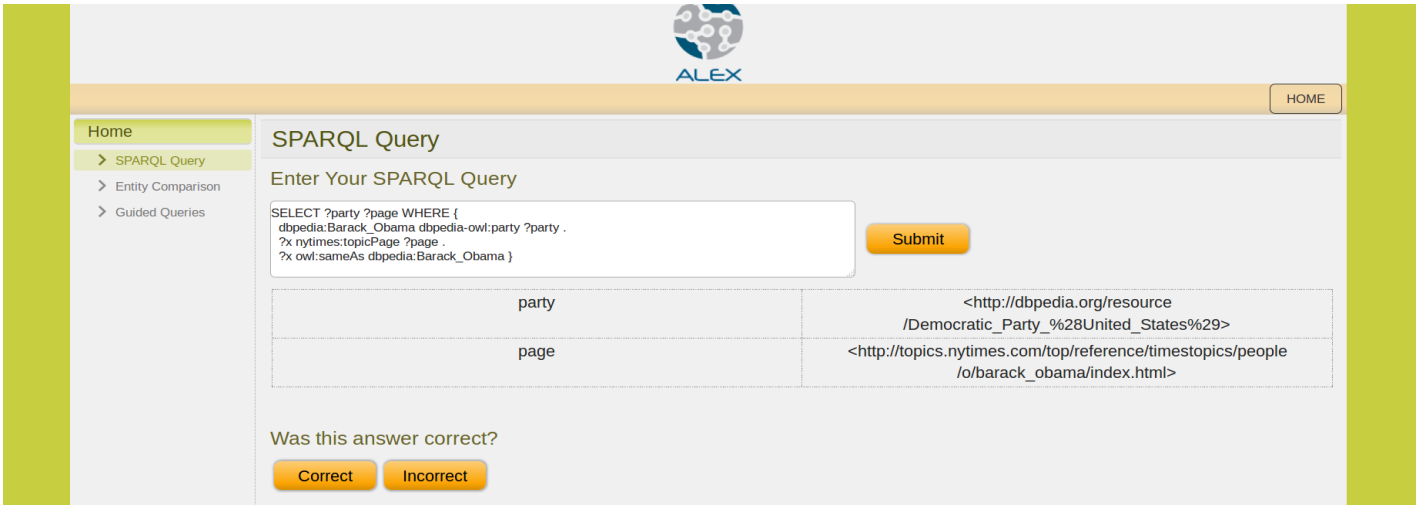

Fig. 2. Issuing SPARQL queries and giving feedback over the returned answer.

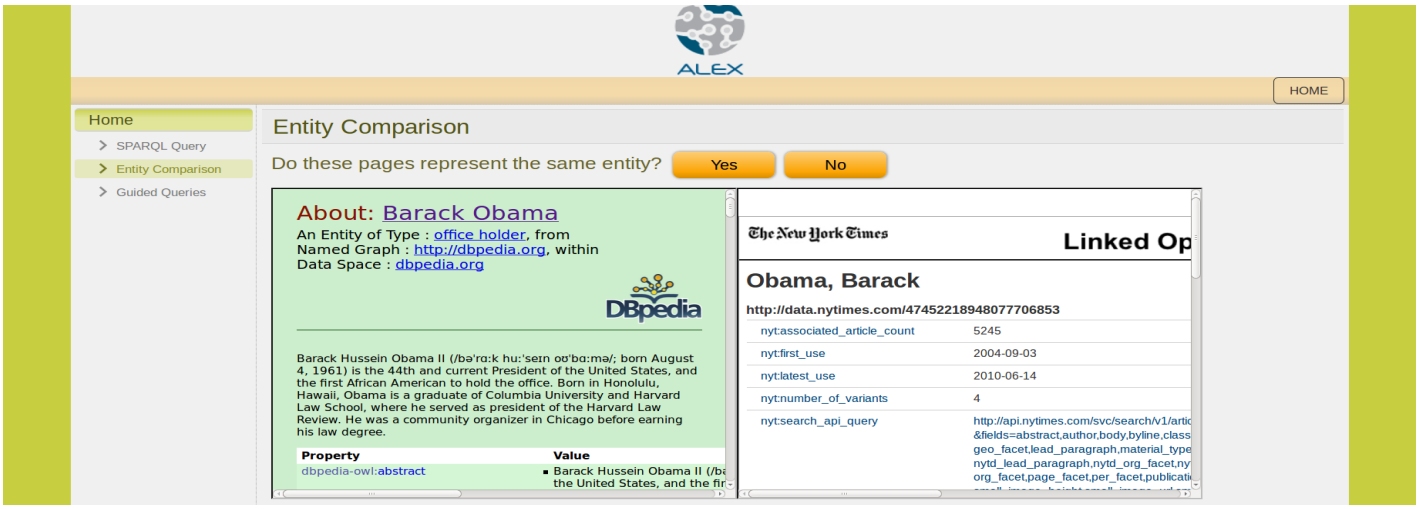

Fig. 3. Showing the entities that are connected with an *owl:sameAs* relation. A user can explicitly approve or reject the link.

every president of the United States. This query mode, which we call Guided Queries, aims to save participants the effort required to make up SPARQL queries during the demonstration.

After receiving a feedback item, ALEX removes the link used to answer the query if the feedback is negative, or tries to discover new links that are similar to the link used to answer the query if the feedback is positive. When ALEX takes an action, this is reflected in the query answer and the precision, recall, and F-measure of the links (which will also be shown to the demonstration participants). After an episode is collected (e.g. 10 feedback items), the policy that ALEX uses to explore new links is improved. The demonstration participant is able to see the policy at any time, and is able to check when ALEX converges.

Leveraging user feedback is important for any RDF linking task. This demonstration will enable participants to better appreciate the effect of user feedback through ALEX, a full fledged working system for link exploration and refinement.

REFERENCES

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 2001.

[2] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *TKDD*, 2007.

[3] C. Bizer, T. Heath, K. Idehen, and T. Berners-Lee. Linked data on the web. In *WWW*, 2008.

[4] A. Doan and R. McCann. Building data integration systems: A mass collaboration approach. In *IIWeb*, 2003.

[5] A. El-Roby and A. Aboulnaga. ALEX: Automatic link exploration in linked data. In *SIGMOD*, 2015.

[6] W. Hu, J. Chen, and Y. Qu. A self-training approach for resolving object coreference on the semantic web. In *WWW*, 2011.

[7] R. McCann, W. Shen, and A. Doan. Matching schemas in online communities: A web 2.0 approach. In *ICDE*, 2008.

[8] A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. FedX: Optimization techniques for federated query processing on linked data. In *ISWC*. 2011.

[9] F. M. Suchanek, S. Abiteboul, and P. Senellart. PARIS: probabilistic alignment of relations, instances, and schema. *Proc. VLDB Endow. (PVLDB)*, 5(3), 2011.

[10] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.