Extracting Relevant Subgraphs from Graph Navigation

Valeria Fionda[1], Claudio Gutierrez[2], Giuseppe Pirró[1]

[1] KRDB, Free University of Bozen-Bolzano, Bolzano, Italy
[2] DCC, Universidad de Chile, Santiago, Chile

Abstract. The main goal of current Web navigation languages is to retrieve set of nodes reachable from a given node. No information is provided about the fragments of the Web navigated to reach these nodes. In other words, information about their connections is lost. This paper presents an efficient algorithm to extract relevant parts of these Web fragments and shows the importance of producing subgraphs besides of sets of nodes. We discuss examples with real data using an implementation of the algorithm in the EXpRESs tool.

Key words: Navigation, Subgraph extraction, Linked Open Data

1 Introduction

Daisy is a researcher interested in a new topic. She is aware of a relevant paper and to find other relevant papers she starts looking at the bibliographic references within. After manually browsing for a while this *graph* of citations, for instance by using Google Scholar, and reaching a relevant paper she wonders: how this particular paper has been reached? How it is connected with the original paper? How two other relevant papers encountered during the navigation are linked in this citation graph? The problem becomes more challenging with automatic navigation where Daisy, by using one of the existing navigation languages (e.g., NautiLOD [3]), can write a complex expression involving the traversal of several data sources. For instance, she wants to collect in the FOAF graph people she knows directly or indirectly that are interested in her new topic. While current graph navigation languages enable to retrieve this set of people, they do not provide information about the structure of the fragment of the FOAF graph where Daisy is linked with these people.

These examples underline the need to augment current navigation languages with capabilities to extract *fragments* (i.e., subgraphs) of the graphs being navigated besides of sets of nodes. Solving this problem is not trivial: there can be an exponential number of paths connecting a seed node with each node reachable according to a navigation expression. Besides, only relevant paths should be kept in the fragment. In a Web setting, which is the focus of this paper, the problem becomes even more challenging since the graph structure is discovered during the navigation.

We present an efficient algorithm to extract Web fragments, describe its pseudo-code and report on its complexity. We present some real world examples to motivate the need of having fragments besides of sets of nodes. Although we focus on the Web, our results are valid for any other graph navigation language. The algorithm has been implemented in the EXpRESs tool available online at `http://swget.wordpress.com/express`.

2 Real World Examples

We present some examples with real data to underline the importance of extracting graph fragments besides of sets of nodes. In Section 3 we describe an efficient algorithm to achieve this goal. The algorithm has been implemented in the EXpRESs tool, which also enables to visualize Web fragments. To illustrate the examples, we use the following subset of the NautiLOD language [3] through which it is possible to specify navigation expressions with semantic control over the data sources visited.

```
path ::= pred | path[test] | (path)? | (path)* | (path|path) | path/path
pred ::= an RDF predicate
test ::= an ASK-SPARQL query
```

Example 1 *(**Co-author subgraph**). Daisy wants to extract the co-author subgraph of Alberto Mendelzon (AM) up to depth 2 only with publications between 1980 and 1990. The following* NautiLOD *expression enables to retrieve the set of AM's co-authors.*

```
dblp:Alberto_O._Mendelzon -p (<foaf:maker>/Q/<foaf:maker>)<2-2>
```

where `Q=[ASK {?p<dc:issued> ?y.FILTER(?y>1980. ?y<1990).}]` serves to keep only papers, reached by expanding the predicate `<foaf:maker>`, published in the interval considered. The notation <2-2> means that the expression within parentheses is repeated two times. From papers reached after evaluating `Q` it is possible to get their authors, which are direct or indirect co-authors of *AM*. This expression enables to collect the set of co-authors. However, information about their connections is lost. For instance, while *R. Fagin* and *J. Ullman* appear in the results, it is not possible to know whether they are direct or indirect co-authors of *AM* or through which sequence of papers they are connected. On the other hand, EXpRESs enables to extract the Web fragment shown in Fig. 1 where it can be observed that *AM*, *R. Fagin* and *J. Ullman* are direct co-authors in 1982 of a paper appeared in the *TODS* journal.

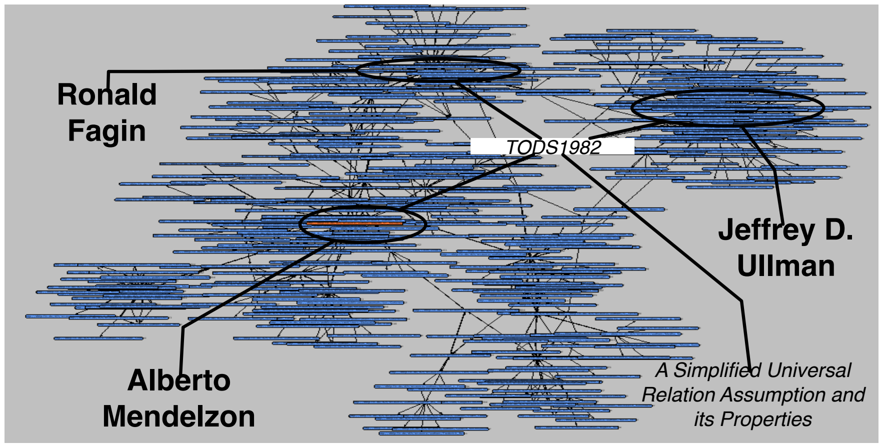
Ronald Fagin
TODS1982
Jeffrey D. Ullman
Alberto Mendelzon
A Simplified Universal Relation Assumption and its Properties

Fig. 1. An excerpt of the co-author subgraph from Alberto Mendelzon.

Example 2 ***(Related work subgraph)****. Daisy is interested in query and navigation languages for the Web. She is aware of the paper Querying the World Wide Web published in 1998 and writes a navigation expression to find papers up to 2 cite-link away. She decides that only papers having some keywords in the title have to be considered.* As citation data are not completely available as linked data, HTML data and links can be exploited. By using Google Scholar identifiers for papers, the following NautiLOD expression cast in the standard Web enables to find papers that respect the criteria.

```
scholar:scholar?cluster=12837463148248684776 -p(<citedBy>/Q)<2-2>
```

where `Q=[title.contains(‘‘Quer*,Web’’)]` exploits the function `title.contains` to check if the title of a paper contains specific words. The first part of the expression is the URL associated to the paper *Querying the world Wide Web* while `<citedBy>` is the label associated to HTML `href` links pointing to papers citing a given paper. Even in this example, while the paper *Formal Models of Web Queries* is in the result set, Daisy cannot see its connections. EXpRESs enables to extract and visualize the fragment including citation paths that lead from the initial paper to this paper. This fragment provides Daisy with an overview on the topic. For sake of space we do not report the results, which are online available at `http://swget.wordpress.com/express`.

3 Extracting Relevant Subgraphs

As discussed before, there is the need to enhance current graph navigation languages with capabilities to deal with Web fragments. We discuss the case of the NautiLOD language even if the same reasoning applies to any other graph query language. The semantics of NautiLOD [3] has been modified for outputting subgraphs instead of set of URIs. Each NautiLOD expression e has associated a Non Deterministic Finite state Automaton (NFA) $\mathtt{A}$, whose size is linear in the size of the expression $|\mathtt{A}| = O(|e|)$. The NFA $\mathtt{A}$ is used to recognize the strings (i.e., concatenations of RDF predicates that lead to a result) that belong to the language defined by e. The evaluation of an expression is done in two steps. The first step corresponds to the building of a Web fragment G containing *all* the triples encountered during the navigation (Algorithm 1). NautiLOD expressions are memoryless, which means that at each step of their evaluation information about previous “states” is not carried on. Thus, G contains several useless paths, that is, paths for which no final states of the automaton have been reached. Cleaning G in order to maintain only meaningful paths is the aim of the second step performed by the extraction algorithm (Algorithm 2). The desideratum is a subgraph G^* induced only by the set of paths matching e. To achieve this goal two data structures are needed: *i)* a hash-table-like structure S where the key is a `uri` $u \in G$ and the value is a set of automaton states at which u has been reached; *ii)* a data structure F, the same as S, where only final states are added. To access the set of states for a `uri` u we use the notation $S(u)$ (resp., $F(u)$). S and F are populated in the *evaluate* algorithm (Algorithm 1, lines 4-6), and are exploited by the *extract* algorithm (Algorithm 2) that keeps only relevant parts of G, thus obtaining G^*.

The space complexity of both S and F is $O(|V| \cdot |e|)$, where V is the number of URIs in G. The overall space complexity is $O(|V| \cdot |e| + |E| + |E_a|))$ where E is the number of edges in G and $|E_a|$ is the number of edges in the NFA $\mathtt{A}$. The overall time complexity is $O(|V| \cdot |e|^2)$.

Algorithm 1: evaluate(NautiLOD expression e)

```
Input: NautiLOD expression $e$, seed URI $seed$
Build: $G$= RDF graph including all the triples retrieved, Automaton A, Data Structures $S$ and $F$
1  $A$= build the NFA from e;
2  add the pair $< seed, A.initial >$ to the set of pairs to be looked-up;
3  while there is a pair $< u, q_k >$ to be looked up s.t. $< u, q_k >$ has not been yet looked up do
4      add $q_k$ to $S(u)$;
5      if $q_k$ is final then
6          add $q_k$ to $F(u)$;
7      $descr = dereference(u)$
8      for each transition $<q_k, p, q_j>$ in the NFA A do
9          if the triple $(u, p, u')$ or $(u', p, u) \in descr$ then
10             add the triple $(u, p, u')$ or $(u', p, u)$ to $G$;
11             add the pair $< u', q_j >$ to the set of pairs to be looked up;
12     for each transition $<q_k, \epsilon, q_j>$ in the NFA A do
13         add the pair $< u, q_j >$ to the set of pairs to be looked up;
14     for each transition $<q_k$, test, $q_j>$ in the NFA A do
15         if test evaluates true over $descr$ then
16             add the pair $< u, q_j >$ to the set of pairs to be looked up;
```

Algorithm 2: extract(Graph G, Set S, Set F, Automaton A)

```
Input  : RDF graph $G$; Sets of states and final states $S$,$F$; NFA A associated to an expression
Output: $G^*$= RDF subgraph of G containing only paths from the seed to URIs in the results
1  for each pair $(u, q_k)$ s.t. $q_k$ belongs to $F(u)$ and $(u, q_k)$ has not been already processed do
2      for each transition $<q_j, p, q_k>$ in the NFA A do
3          if the triple $(u', p, u)$ or $(u, p, u') \in G$ and the state $q_j \in S(u')$ then
4              add $q_j$ to the set of final states $F(u')$;
5              add the triple $(u', p, u)$ or $(u, p, u')$ to $G^*$;
6      for each transition $<q_j, \epsilon, q_k>$ in the NFA A do
7          add $q_j$ to the set of final states $F(u)$;
8      for each transition $<q_j$, test, $q_k>$ in the NFA A  do
9          if $q_j \in S(u)$ then
10             add $q_j$ to the set of final states $F(u)$;
   return $G^*$;
```

4 Related Work

The main goal of current Web navigation languages is to retrieve set of nodes. Information about the structure of subgraphs spanned by these nodes is lost. Note that SPARQL CONSTRUCT builds graphs from collections of nodes that do not reflect the structure of the navigation. Extracting this subgraph structure is challenging: a naive approach remembering all the paths is not feasible since they can be exponential in number. In a Web context this is even more challenging since the structure of the graph being navigated is not known a priori. We presented an efficient subgraph extraction algorithm for NautiLOD that can be extended to any other graph query language. It has been inspired by Earley's parser for Context-free Grammars [2]. It is also related to some work in XPath (e.g., XPlainer [1]). The main difference is that our approach deals with (a priori unknown) graphs instead of (known) trees.

References

1. Consens M. P., Liu J. W. S., Rizzolo F.: XPlainer: Visual Explanations of XPath Queries. In *Proc. of ICDE*, pp. 636-645, (2007).
2. Earley, J.: An Efficient Context-Free Parsing Algorithm. *CACM*, 13(2), pp. 94-102, (1970).
3. Fionda V., Gutierrez C., Pirró G.: Semantic Navigation on the Web of Data: Specification of Routes, Web Fragments and Actions. In *Proc. of WWW*, pp. 281-290, (2012).