Landmark indexing for evaluation of label-constrained reachability queries

Lucien Valstar†, George Fletcher†, Yuichi Yoshida‡

†TU Eindhoven (Netherlands),
‡National Institute of Informatics
and Preferred Infrastructure, Inc. (Japan)

SIGMOD 2017
Chicago, 16 May 2017
Big **graph** data sets are ubiquitous

- social networks (e.g., LinkedIn, Facebook)
- scientific networks (e.g., Uniprot, PubChem)
- knowledge graphs (e.g., DBPedia, MS Academic Graph)
- transportation and utility networks
- ... 

Focus is on “**things**” (i.e., nodes, vertices) and their **relationships** (i.e., labeled directed edges)
We study **Label-Constrained Reachability (LCR) Queries** on networks:

*Given vertices* $s$ *and* $t$ *of labeled graph* $G$ *and a subset* $L$ *of the set of all edge labels* $\mathcal{L}$ *of* $G$, *determine whether or not there is a path from* $s$ *to* $t$ *using only edges with labels in* $L$.

When such a path exists, we denote this by $s \overset{L}{\sim} t$. 
Example. The query $(v_1, v_5, \{\text{friendOf}\})$ is true.

The query $(v_1, v_3, \{\text{friendOf}\})$ is false.
Label-constrained reachability queries on networks

Example. The query $(v_1, v_5, \{\text{friendOf}\})$ is true.

The query $(v_1, v_3, \{\text{friendOf}\})$ is false.

**LCR Queries**

- Natural generalization of reachability queries.
- An important fragment of the language of regular path queries.
- Implemented in W3C’s SPARQL 1.1, Neo4j’s Cypher, and Oracle’s PGQL.

Landmark indexing for LCR query evaluation (SIGMOD 2017, Chicago)
Despite the importance of LCR queries, current solutions do not scale to large graphs occurring in practice.

There are two approaches to solving LCR queries: exhaustive search using state-of-the-art methods such as direction-optimizing BFS (DBFS)

or graph indexing for accelerated search
  - Jin et al. *SIGMOD* 2010
  - Bonchi et al. *EDBT*, 2014
Our contributions. New indexing methods for LCR queries exploiting landmark vertices.

- Scales to orders of magnitude larger graphs than current indexing methods.
- Up to orders of magnitude faster query evaluation than current solutions.
- Our implementation is publicly available as open-source at https://github.com/DeLaChance/LCR
Naive Idea (FULL-LI)
Given a graph $(V, E, \mathcal{L})$, for each vertex $v \in V$, store in an index the pairs $\{(w, L) \mid w \in V, L \subseteq \mathcal{L}, \text{ and } v \overset{L}{\sim} w\}$. 
Naive Idea (\texttt{FULL-LI})
Given a graph $\langle V, E, \mathcal{L} \rangle$, for each vertex $v \in V$, store in an index the pairs $\{(w, L) \mid w \in V, L \subseteq \mathcal{L}, \text{ and } v \xrightarrow{L} w\}$.

Given a query $(s, t, L)$, just check whether or not $(t, L)$ is in the index for $s$. 
Example. The Full-LI index entry for $v_2$:

- $(v_3, \{\text{likes}\})$
- $(v_3, \{\text{friendOf}, \text{likes}\})$
- $(v_3, \{\text{friendOf}, \text{follows}, \text{likes}\})$
- $(v_4, \{\text{friendOf}, \text{follows}\})$
- $(v_5, \{\text{friendOf}\})$
- $(v_5, \{\text{friendOf}, \text{follows}\})$.
Landmark indexing for LCR: naive solution

Example. The Full-LI index entry for $v_2$:

$(v_3, \{\text{likes}\})$, $(v_3, \{\text{friendOf, likes}\})$, $(v_3, \{\text{friendOf, follows, likes}\})$, $(v_4, \{\text{friendOf, follows}\})$, $(v_5, \{\text{friendOf}\})$, $(v_5, \{\text{friendOf, follows}\})$.

Naive Idea (Full-LI)

- Excellent query performance.
- Does not scale to large graphs.
Landmark Index (LI)
Only build indexes for a select small number of landmark vertices
  ▶ e.g., top $k$ vertices of highest degree
Furthermore, only store entries $(w, L)$ such that $L$ is a minimal label set connecting $v$ to $w$
  ▶ that is, there is no $L'$ strictly contained in $L$ such that $v \stackrel{L'}{\sim} w$. 
Landmark Index (LI)
Only build indexes for a select small number of landmark vertices
  ▶ e.g., top $k$ vertices of highest degree
Furthermore, only store entries $(w, L)$ such that $L$ is a minimal label set connecting $v$ to $w$
  ▶ that is, there is no $L'$ strictly contained in $L$ such that $v \overset{L'}{\leadsto} w$.

Given a query $(s, t, L)$, perform BFS from $s$ only using edges with labels in $L$. When we hit a landmark vertex, we use its index to obtain the answer immediately.
Example. The LI index entry for $v_2$:

$\langle v_3, \{\text{likes}\} \rangle$,
$\langle v_4, \{\text{friendOf, follows}\} \rangle$,
$\langle v_5, \{\text{friendOf}\} \rangle$.

Half as many entries as FULL-LI entry for $v_2$. 
Landmark indexing for LCR: selective landmarking

Example. The LI index entry for $v_2$:

$\langle v_3, \{\text{likes}\} \rangle$,
$\langle v_4, \{\text{friendOf, follows}\} \rangle$,
$\langle v_5, \{\text{friendOf}\} \rangle$.

Half as many entries as FULL-LI entry for $v_2$.

Landmark index (LI)

- Balances space/time.
- Can significantly reduce index size.
- Still obtain the benefits of accelerated search.
Extended Landmark Index (LI$^+$)

Two extensions to make LI more efficient.

(1) It may take a long time before finding a landmark. We can remedy this by building an incomplete index for non-landmarks: for each non-landmark $v$, we insert a small number of entries $(v', L)$ where $v'$ is a landmark and $v \leadsto^L v'$. These provide shortcuts to landmarks during search.
Extended Landmark Index ($LI^+$)

Two extensions to make LI more efficient.

(2) There is a strong asymmetry in evaluation of true- and false-queries. A true-query can stop after finding a landmark, whereas a false-query often needs to explore larger parts of the graph.

To remedy this, we can maintain for each landmark $v$ and label set $L$ the “reachable-by” set $R_L(v) = \{ w \in V \mid v \xrightarrow{L} w \}$. 
Example.

\[ R_{\{\text{friendOf}\}}(v_1) = \{v_2, v_4, v_5\}. \]
Extended Landmark Index (LI⁺)
Two extensions to make LI more efficient.

(2, cont.) During evaluation of query \((s, t, L)\), suppose we have found \(s \xRightarrow{L} v\) and \(v \xleftarrow{L} t\), for some landmark \(v\).

Then, for every \(w \in R_L(v)\), we must have \(w \xleftarrow{L} t\). Hence, we can mark and never visit any vertex of \(R_L(v)\) during the rest of the search.
Extended Landmark Index (LI$^+$)
Two extensions to make LI more efficient.

(2, cont.) During evaluation of query $(s, t, L)$, suppose we have found $s \xrightarrow{L} v$ and $v \xrightarrow{L} t$, for some landmark $v$.

Then, for every $w \in R_L(v)$, we must have $w \xrightarrow{L} t$. Hence, we can mark and never visit any vertex of $R_L(v)$ during the rest of the search.

For practical purposes, we only keep a subset of the $R_L(v)$’s, and only for relatively small label sets.
Experimental study

**Settings**

- Linux server with 258GB of memory and a 2.9GHz 32-core processor.
- Fourteen real networks, ranging from social networks to biological networks.
- The number of landmarks is $1250 + \sqrt{n}$, where $n$ is the number of vertices. The budget for non-landmarks is 20.
- Generated sets of 1000 true-queries and 1000 false-queries.

See our paper for full details and also details of performance on synthetic graphs where we study the impact of graph density, label set size, and graph structure.
Experimental study

Settings

- Linux server with 258GB of memory and a 2.9GHz 32-core processor.
- Fourteen real networks, ranging from social networks to biological networks.
- The number of landmarks is $1250 + \sqrt{n}$, where $n$ is the number of vertices. The budget for non-landmarks is 20.
- Generated sets of 1000 true-queries and 1000 false-queries.

We report here highlights of results on indexing costs and query performance. See our paper for full details and also details of performance on synthetic graphs where we study the impact of graph density, label set size, and graph structure.
Experimental study: indexing costs

<table>
<thead>
<tr>
<th>Dataset</th>
<th>LI</th>
<th></th>
<th>LI⁺</th>
<th></th>
<th>Full-LI</th>
<th></th>
<th>ZOU</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IT</td>
<td>IS</td>
<td>IT</td>
<td>IS</td>
<td>IT</td>
<td>IS</td>
<td>IT</td>
<td>IS</td>
</tr>
<tr>
<td>robots</td>
<td>0.1</td>
<td>5</td>
<td>0.1</td>
<td>5</td>
<td>0.1</td>
<td>5</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>advogato</td>
<td>4</td>
<td>135</td>
<td>3</td>
<td>131</td>
<td>7</td>
<td>369</td>
<td>11,867</td>
<td>369</td>
</tr>
<tr>
<td>epinions</td>
<td>272</td>
<td>2,903</td>
<td>205</td>
<td>2,091</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NotreDame</td>
<td>242</td>
<td>2,424</td>
<td>193</td>
<td>1,895</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BioGrid</td>
<td>58</td>
<td>1,410</td>
<td>50</td>
<td>1,302</td>
<td>36</td>
<td>3,207</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>webGoogle</td>
<td>5,887</td>
<td>33,931</td>
<td>5,665</td>
<td>33,497</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Youtube</td>
<td>3,121</td>
<td>336</td>
<td>2,841</td>
<td>300</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>socPokec</td>
<td>9,762</td>
<td>77,155</td>
<td>9,461</td>
<td>75,698</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>wikiLinks(fr)</td>
<td>24,873</td>
<td>98,125</td>
<td>25,641</td>
<td>103,414</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Number of edges: robots 2.9k, advogato 51k, epinions 840k, NotreDame 1.4M, BioGrid 1.5M, webGoogle 5.1M, Youtube 10.7M, socPokec 30.6M, wikiLinks(fr) 102.3M.
## Experimental study: query performance

<table>
<thead>
<tr>
<th>Dataset</th>
<th>true LI</th>
<th>true LI⁺</th>
<th>true DBFS (µs)</th>
<th>false LI</th>
<th>false LI⁺</th>
<th>false DBFS (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>robots</td>
<td>17.63</td>
<td>17.63</td>
<td>1.77</td>
<td>6.95</td>
<td>6.95</td>
<td>0.70</td>
</tr>
<tr>
<td>advogato</td>
<td>84.25</td>
<td>93.08</td>
<td>20.6</td>
<td>3.33</td>
<td>1.89</td>
<td>0.67</td>
</tr>
<tr>
<td>opinions</td>
<td>69.18</td>
<td>52.10</td>
<td>106</td>
<td>0.00</td>
<td>0.58</td>
<td>1.91</td>
</tr>
<tr>
<td>NotreDame</td>
<td>22.27</td>
<td>7.27</td>
<td>159</td>
<td>5.17</td>
<td>49.44</td>
<td>555</td>
</tr>
<tr>
<td>BioGrid</td>
<td>16.98</td>
<td>20.79</td>
<td>848</td>
<td>0.18</td>
<td>37.95</td>
<td>709</td>
</tr>
<tr>
<td>webGoogle</td>
<td>338.26</td>
<td>184.52</td>
<td>1,340</td>
<td>0.00</td>
<td>0.57</td>
<td>9.65</td>
</tr>
<tr>
<td>Youtube</td>
<td>16.10</td>
<td>21.10</td>
<td>2,000</td>
<td>0.53</td>
<td>12.19</td>
<td>3,880</td>
</tr>
<tr>
<td>socPokec</td>
<td>9.20</td>
<td>10.81</td>
<td>1,290</td>
<td>0.00</td>
<td>0.25</td>
<td>39.8</td>
</tr>
<tr>
<td>wikiLinks(fr)</td>
<td>54.53</td>
<td>44.54</td>
<td>3,120</td>
<td>0.00</td>
<td>0.42</td>
<td>38.8</td>
</tr>
</tbody>
</table>

Landmark indexing for LCR query evaluation (SIGMOD 2017, Chicago)
General observations.

- For true-queries, LI and LI$^+$ are always advantageous, accelerating query evaluation up to two orders of magnitude over state-of-the-art search methods. For false-queries, LI$^+$ was within the same order of magnitude or better for the majority of cases.
  - Often we found that search failure occurred much closer to the target, to the advantage of DBFS. This indicates the need in future work to study direction-optimizing variants of our solutions.
General observations.

- For true-queries, LI and LI$^+$ are always advantageous, accelerating query evaluation up to two orders of magnitude over state-of-the-art search methods. For false-queries, LI$^+$ was within the same order of magnitude or better for the majority of cases.
  - Often we found that search failure occurred much closer to the target, to the advantage of DBFS. This indicates the need in future work to study direction-optimizing variants of our solutions.

- LI and LI$^+$ are the only indexing strategies which can handle large graphs, up to four orders of magnitude larger than current indexing strategies.
Concluding remarks

Our contributions: New landmark-based indexing solutions for scalable evaluation of LCR queries, scaling to orders-of-magnitude larger graphs and orders of magnitude faster evaluation time.
Concluding remarks

Looking ahead:

▶ finer analysis of the impact of graph topology and complexity on the performance and further optimization of our solutions, e.g., graphs of bounded treewidth.

▶ study of landmark-based evaluation methods for extensions to the class of LCR queries.

▶ the study of landmark indexing in multi-core environments.

▶ study of alternative search strategies for improving performance on false queries.

▶ applications of our indexes for evaluation of practical query languages such as SPARQL 1.1 and openCypher.
Landmark indexing for evaluation of label-constrained reachability queries

Lucien Valstar, George Fletcher, and Yuichi Yoshida

TU Eindhoven (Netherlands)
National Institute of Informatics
and Preferred Infrastructure, Inc. (Japan)

https://github.com/DeLaChance/LCR

Thank you!