Abstract
A top-k shortest path algorithm finds the k shortest paths of a given graph ordered by length. Interpreting graphs as RDF may lead to additional constraints, such as special loop restrictions or path patterns. Thus, traditional algorithms such as the ones by Dijkstra, Yen, or Eppstein cannot be applied without further ado. We therefore implemented a solution method based on Eppstein’s algorithm which is thoroughly discussed in this paper. Using this method we were able to solve all tasks of the ESWC 2016 Top-k Shortest Path Challenge while achieving only moderate overhead compared to the original version. However, we also identified some potential for improvements. Additionally, a concept for embedding our algorithm into a SPARQL endpoint is provided.

Approach

two tasks in ESWC top-k shortest path challenge:
1. top-k shortest valid paths
2. every path should have the edge P as the outgoing edge of all P
   - a path is valid only if it contains unique triples (do not use the same triple twice in one path)
   - vertices can be visited multiple times (Yen’s algorithm which forbids loops would not find such paths)
   - approach based on Eppstein algorithm (k shortest path algorithm with loops)
      - compute single destination shortest path tree T (e.g. Dijkstra) – all other edges are called sidetracks (G – T)
      - build a graph P(G) based on heaps \( H_v(T) \) which orders all sidetracks on the shortest path from v to the destination
      - most vertices in P(G) correspond to a valid path (this results in a moderate overhead compared to the Eppstein algorithm)
      - a path is built up by using all activated sidetracks, otherwise use shortest path
e.g. example graph:

   ![Shortest Path Tree](image)

   ![Sidetracks (G – T)](image)

- corresponding graph P(G):

   ![](image)

Introduction
- shortest path: obvious link between resources
- subsequent shortest paths: maybe even more interesting relationships
- interpreting graphs as RDF may lead to additional constraints
- common algorithms (e.g. Dijkstra, Yen, Eppstein, …) cannot be applied without further ado

Evaluation
- evaluation based on the training set of the ESWC 2016 Top-k Shortest Path Challenge
- successfully solved all queries in the challenge
- training set: 9,996,917 triples, 3,598,913 of them being literal statements, 394,085 multiple edges, 407 reflexive edges, 181,702 duplicate statements, average out degree of vertices 6.03 (s.d. 4.51)
- high k results in a high overhead since more potentially invalid paths are found
- the relation between k and the overhead of all queries is depicted in Figure 2
- algorithm’s overhead increases with the number of loops

SPARQL
- SPARQL endpoint (FUSEKI) which can extract top k shortest paths
- example query:

   ```sparql
   SELECT * 
   WHERE { 
     FILTER(?r1 = dbp:after). 
     LIMIT 2 
   }
   ```

- example response:

```
---|-----------------|-----------|-------------------|-----------|-----|------------|---------------------|
```

Conclusion
- approach for finding top-k shortest paths based on Eppstein’s algorithm which induces only moderate overhead
- successfully solved all tasks given in the ESWC 2016 Top-k Shortest Path Challenge
- in future versions we intend to lazily build heaps and try to predict invalid paths earlier

Acknowledgement
This work was partially funded by the BMBF project Multimedia Opinion Mining (MOM: 01W15002).

Contact:
Msc. Sven Hertling
Researcher - Knowledge Management Department
German Research Center for Artificial Intelligence (DFKI GmbH)
Phone: +49 631 20575 - 205
Mobile: +49 175 103 12 15
Email: sven.hertling@dfki.de
Website: http://km.dfki.de/sven-hertling-msc

Fig. 1. Modified Eppstein’s Algorithm

Fig. 2. Relation of k and corresponding overhead

<table>
<thead>
<tr>
<th>h</th>
<th>k</th>
<th>overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>10^1</td>
<td>10</td>
<td>0.00</td>
</tr>
<tr>
<td>10^2</td>
<td>10</td>
<td>0.01</td>
</tr>
<tr>
<td>10^3</td>
<td>10</td>
<td>0.10</td>
</tr>
<tr>
<td>10^4</td>
<td>10</td>
<td>1.00</td>
</tr>
<tr>
<td>10^5</td>
<td>10</td>
<td>10.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>k</th>
<th>overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>0.00</td>
</tr>
<tr>
<td>4</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>0.00</td>
</tr>
</tbody>
</table>

![Diagram](image)