Space Efficient Processing for Label-Constrained Graph Reachability Queries

Mohammadsadegh Najafi[1], **Chris Hickey**[2]

[1]Seoul National University, Department of Computer Science
[2]Seoul National University, Department of Cognitive Science

December 2, 2019

Table of Contents

- (LCR) Given s and t of graph G and L' subset of L, determine whether or not there exists a path from s to t in G using only edges with labels in L'.

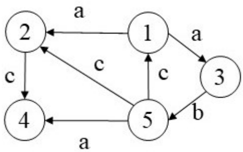


Figure 1: Example Graph with $|V| = 5$, $|E| = 7$ and $L = \{a, b, c\}$

Problem Statement

- *GIVEN*: A graph G, a source vertex s, target vertex t, and an LCR query with label set L' which is a subset of all labels present in the graph L;
- *FIND* a way to develop a simple, practical, efficient algorithm which can respond *true* or *false* as to whether a path exists between s and t, in order to,
- *DETERMINE* whether a more memory efficient means of responding to LCR then alternative methods such as Landmark Indexing

Landmark Indexing [1]

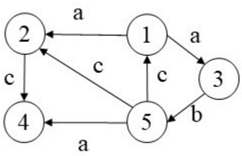
- This work is a current advancement in LCR query processing.
- Naive indexing method:
  - Given the input graph $G = (V, E, L)$
  - Store every pair $(u, L') \in V \times 2^L$ as an index for v if there exists an L'-path from v to u.
  - We can answer any LCR query (v, u, L'') by checking the pair of (u, L') in index of v.
- What are landmarks?
  - They make the indexing time and index size sufficiently small.
  - Only construct indices for small number of vertices called **landmarks**.

Space Efficient Approach

- Landmark Indexing method stores indices in $O(((n+m)2^{l}-1)\ \#\ ofLandmarks)$
- All Label Combination removes # of landmarks and stores in $O((n+m)2^{l}-1)$

All Labels Combinations

- We introduce the all labels combinations (ALC) method.
- Given G with label set L of size l, the ALC method constructs $2^l - 1$ unlabeled subgraphs.
  - Corresponds to every non-empty subset of L.
- For instance, for the graph shown in Figure 1, $\mathrm{L} = \{a, b, c\}$, and the non-empty label subsets are $\{a\}$,$\{b\}$,$\{c\}$,$\{a, b\}$,$\{a, c\}$,$\{b, c\}$ $\{a, b, c\}$.



All Labels Combinations - Example

The subgraphs corresponding to each of these subsets.

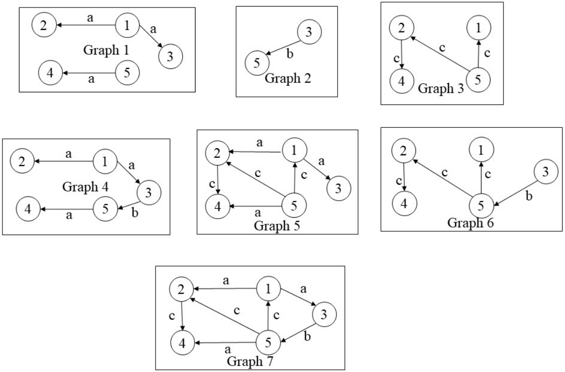


Figure 2: Graphs generated based on label combinations

All Labels Combinations - Decomposing Algorithm

Algorithm 1 All Label Combinations

Input: A given Edge-Labeled Graph G and all label selections L

Output: $2^l - 1$ Number of Separate Unlabeled Subgraphs Stored in Map Data Structure ALC $< CombinationSet\ ls, SubgraphG' >$

for Each labelset $ls \in L$ **do**
    Initialize subgraph G'
    for Each edge $e \in G.E$ **do**
        if $e.label \;==\; ls$ **then**
            Add e to G'
        end if
    end for
    Add $< ls, G' >$ to ALC
end for

All Labels Combinations - ALC Query Algorithm

Algorithm 2 Query for ALC

Input: A given start vertex s, target t, and labelset ls.

Output: **true** if there is a path between s and t, otherwise **false**.

```
Find graph G' corresponding to ls form ALC < ls, G' >
for each v ∈ G.V do
    v.status= false
end for
Queue q is an empty queue
q.push(s)
while q is not empty do
    u = q.pop()
    u.status= true
    if u == t then
        return true
    end if
    for each w ∈ G.Adj[u] do
        if w.status == false then
            q.push(v)
        end if
    end for
end while
return false
```

ALC + Strongly Connected Components [8]

- The main weakness of the proposed ALC algorithm is slow query time : up to O(m+n); especially slow for *false* queries
- To address this weakness, we integrated *Strongly Connected Components* with ALC.
- A subset of vertices C in G is called strongly connected if there is a path between all pairs of vertices u and $v \in C$
- The advantages of integrating SCC with ALC are:
  - Decrease query response time for both *true* and *false* queries.
  - Decreased memory space usage by decreasing the number of vertices that need to be stored for answering LCR queries.

Decomposition of Labeled Graph through ALC+SCC

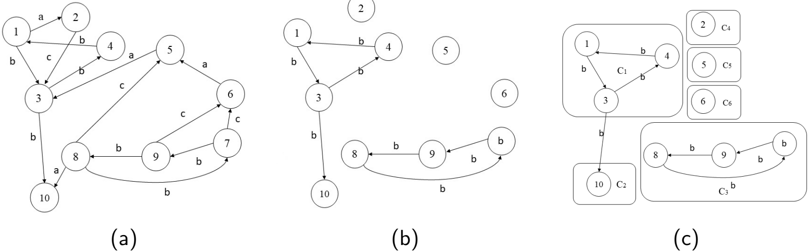


Figure: (a) A synthetic edge-Labeled graph (b) The result of ALC for selection $\{b\}$ (c) determined the corresponding graph SCC, C_is shows strongly connected components.

Algorithm 3 Combined Strongly Connected Components and All Label Combinations

Input: $2^l - 1$ Number of subgraphs with strongly connected components stored in Map Data Structure ALC $< \mathit{LabelSet\ ls}, \mathit{SubgraphG'} >$

Output: $2^l - 1$ Number of Separate Unlabeled Subgraphs Stored in Map Data Structure StronglyConnectedComponentsALC $< \mathit{LabelSet\ ls}, \mathit{SubgraphG''} >$

1: Compute DFS for subgraph G' to compute finish times $v.f$ for each vertex v
2: Compute transpose graph G' as G'^T
3: Call DFS for G'^T, In DFS performance consider $v.f$ in decreasing order and ad the result to G''
4: The result of line 3 is a tree. if a node has at least one child, set *outPortal* of that node *true*.
5: The result of line 3 is a tree G'' which each vertex of the tree is a separate strongly connected components of G''
6: Add $< ls, G'' >$ to StronglyConnectedComponentsALC

ALC + SCC - ALC+SCC Query Algorithm

Algorithm 4 Query Algorithm for Synthesis of SCC and ALC Approach

Input: A given start vertex s, target t, and labelset ls.

Output: **true** if there is a path between s and t, otherwise **false**.

```
Find graph G'' corresponding to l form StronglyConnectedComponentsALC
< ls, G' >
if s.SSCid == t.SSCid then
    return true
end if
if s.SSCid.oudPortal == false then
    return False
end if
for each v ∈ G''.V do
    v.status= false
end for
Queue q is an empty queue
q.push(s)
while q is not empty do
    u = q.pop()
    u.status= true
    if u.SSCid == t.SSCid then
        return true
    end if
    if s.SSCid.oudPortal == false then
        return False
    end if
    for each w ∈ G''.Adj[u] do
        if w.status == false then
            q.push(v)
        end if
    end for
end while
return false
```

Experiments

- Experiments were conducted comparing both ALC and LI algorithms .
- ALC, ALC+SCC and LI were evaluated on 3 criteria:
  - *Response Time*: Time of response in microseconds
  - *Memory Usage*: Amount of memory space required to service queries.
  - *Construction Time*: Length of pre processing time required to respond to queries.

Experiments - Datasets

Table: Information about all used datasets. The last column shows whether the graph is directed or not. The fifth column is dedicated to indicating whether the labels of the graph are augmented.

DataSets	$\|V\|$	$\|E\|$	ℓ	Augmented Labels	Directed/Undirected
Synthetic 1	100	242	4	Yes	Directed
Synthetic 2	500	2485	4	Yes	Directed
Synthetic 3	5000	12492	4	Yes	Directed
Synthetic 4	100	485	8	Yes	Directed
Synthetic 5	1000	2994	8	Yes	Directed
Synthetic 6	2000	5994	8	Yes	Directed
Synthetic 7	5000	24985	8	Yes	Directed
robots	1400	2900	4	No	Directed
Advogato	14010	70472	4	No	Directed
webGoogle	875K	5.1M	8	Yes	Directed
webStanford	281K	2.3M	8	Yes	Directed
WebBerkstan	658K	7.6M	8	Yes	Directed
Youtube	15K	10.7M	5	No	Directed
StringFC	15K	2M	7	No	Undirected
BioGrid	64k	1.5M	7	No	Undirected
StringHS	16K	1.2M	7	No	Undirected

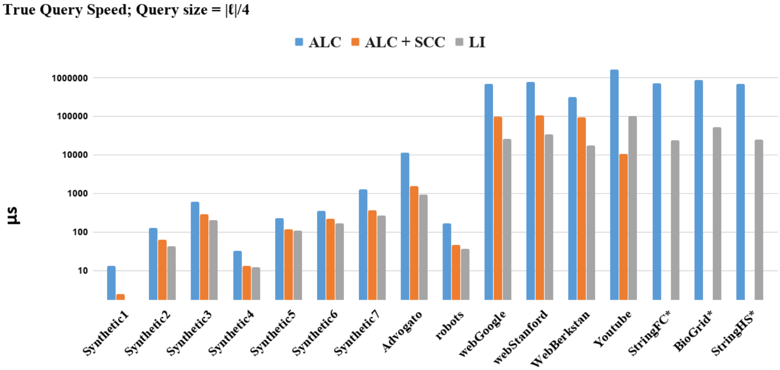
Experiments - Response Time (ALC)

- ALC method was between 4-9 times slower than Landmark indexing for *true* queries, and 6-16 times slower for *false* queries on synthetic graphs.
- ALC method was between 4-29 times slower than Landmark indexing for *true* queries, and 16-134 times slower for *false* queries on real world directed graphs.

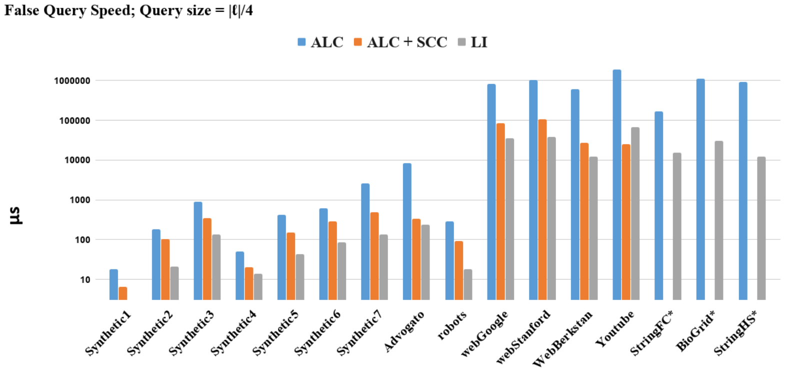
Experiments - Response Time (ALC+SCC)

- ALC+SCC method was between 1.3-1.4 times slower than Landmark indexing for *true* queries, and 1-2.5 times slower for *false* queries on synthetic graphs.
- ALC+SCC method was at worst 5 times slower than Landmark Indexing for *true* queries and 6 times slower *false* queries on real world directed graphs.
- ALC+SCC method outperformed Landmark Indexing for both *true* and *false* queries on some real world graphs:
  - YouTube graph which had 60% of nodes in one single SCC
- ALC+SCC method responds to queries on undirected graphs in constant time

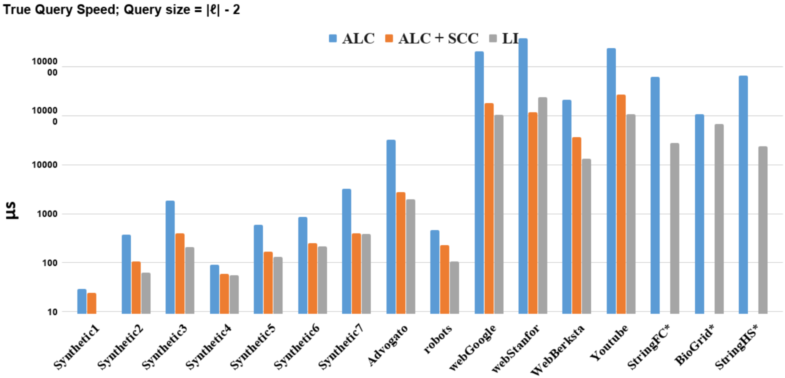
Experiments - Response Time (1)



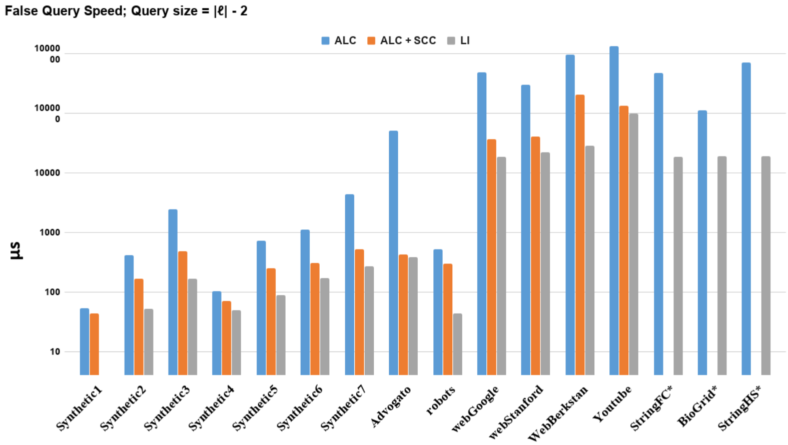
Experiments - Response Time (2)



Experiments - Response Time (3)



Experiments - Response Time (4)



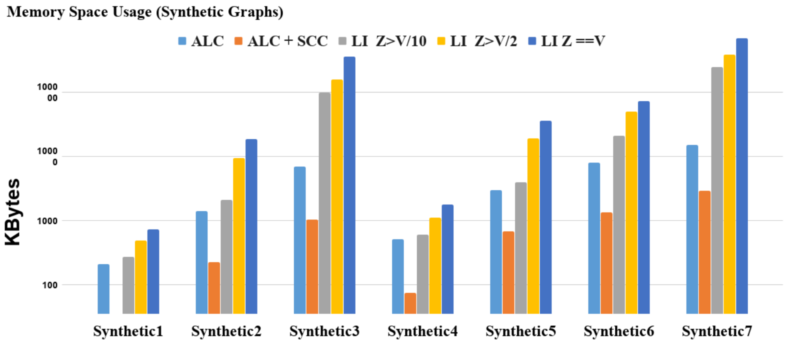
Experiments - Memory Usage (ALC)

- The difference in memory space usage depended on the amount of landmarks Z used by the algorithm, the size of the graph, and the total number of labels in the graph, and the number of SCC's in the graph.
- With number of landmarks in LI set to $Z > \frac{V}{2}$, LI required between 1.5 and 26 times more memory space usage than ALC on synthetic graphs.
- With number of landmarks in LI set to $Z > \frac{V}{10}$, LI required between 6 and 46 times more memory space usage than ALC on real world graphs.

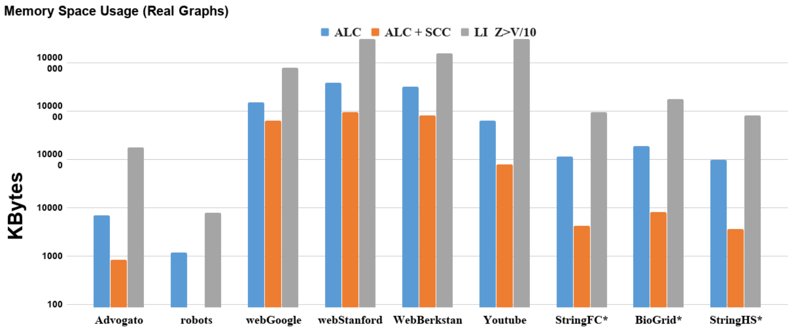
Experiments - Memory Usage (ALC+SCC)

- With number of landmarks in LI set to $Z > \frac{V}{2}$, LI required between 13 and 130 times more memory space usage than ALC+SCC on synthetic graphs.
- With number of landmarks in LI set to $Z > \frac{V}{10}$, LI required between 91 and 392 times more memory space usage than ALC+SCC on real world directed graphs.
- With number of landmarks in LI set to $Z > \frac{V}{10}$, LI required up to 220 times more memory space usage as ALC+SCC on real world undirected graphs.

Experiments - Memory Usage (1)



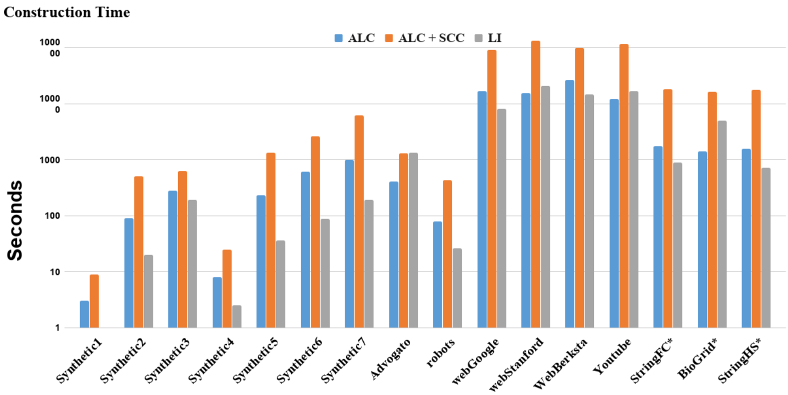
Experiments - Memory Usage (2)



Experiments - Construction Time

- ALC algorithm took between 3-7 times longer construction time than LI on synthetic graphs.
- ALC algorithm construction time was comparable to LI on real world graphs.
- ALC+SCC algorithm took between 9-30 times longer construction time than LI on synthetic graphs.
- ALC+SCC algorithm took between 1-25 times longer construction time than LI on real world graphs.
- All construction time experiments were run with number of landmarks in LI set to $Z > \frac{V}{10}$,

Experiments - Construction Time



Conclusion

- The paper proposes two new algorithm for solving Label Constrained Reachability queries - the All Labels Combined algorithm and the Strongly Connected Components All Labels Combined Algorithm.
- This ALC algorithm is up to 46 times more space efficient than LI, however response time can be over 100 times as long for LCR queries on real world graph.
- This ALC+SCC algorithm provides comparable response times to LI, while requiring up to 392 times less memory space usage for some large real world graphs.
- Construction time was faster for LI in general when compared to both ALC and ALC+SCC.
- Application of ALC could be used for evaluation of practical query languages such as openCypher and SPARQL1.1.

References

1 Valstar, L. D., Fletcher, G. H., & Yoshida, Y. (2017, May). Landmark indexing for evaluation of label-constrained reachability queries. In Proceedings of the 2017 ACM International Conference on Management of Data (pp. 345-358). ACM.

2 Bonchi, F., Gionis, A., Gullo, F., & Ukkonen, A. (2014, March). Distance oracles in edge-labeled graphs. In EDBT (pp. 547-558).

3 Zou, L., Xu, K., Yu, J. X., Chen, L., Xiao, Y., & Zhao, D. (2014). Efficient processing of label-constraint reachability queries in large graphs. Information Systems, 40, 47-66.

4 Cheng, J., Huang, S., Wu, H., & Fu, A. W. C. (2013, June). TF-Label: a topological-folding labeling scheme for reachability querying in a large graph. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (pp. 193-204). ACM.

References

5 Jin, R., Hong, H., Wang, H., Ruan, N., & Xiang, Y. (2010, June). Computing label-constraint reachability in graph databases. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data (pp. 123-134). ACM.

6 Yu, J. X., & Cheng, J. (2010). Graph reachability queries: A survey. In Managing and Mining Graph Data (pp. 181-215). Springer, Boston, MA.

7 Leskovec, J., & Sosič, R. (2016). Snap: A general-purpose network analysis and graph-mining library. ACM Transactions on Intelligent Systems and Technology (TIST), 8(1), 1.

8 Leiserson, C. E., Rivest, R. L., Cormen, T. H., & Stein, C. (2001). Introduction to algorithms (Vol. 6). Cambridge, MA: MIT press.

Thank you for your attention.