Synthesising Strongly Connected Components and All Labels Combined Method for Label-Constrained Reachability Queries

Mohammad Sadegh Najafi
Dept. of Computer Science
Seoul National University
najafi@tcs.snu.ac.kr

Chris Hickey
Dept. of Cognitive Science
Seoul National University
chris.hickey@ucdconnect.ie

December 2, 2019

Abstract

Real world network graphs such as social networks, semantic verbs and citation networks can often contain varying different edge labels connecting vertices. For example, in the context of social networks, nodes can be connected via 'likes', 'comments', 'friendship' etc. What is the fastest, most space efficient way to query whether a path exists between two vertices in a network, using only a subset of vertices? This present research proposal aims to contribute to this question by comparing performance of the current state of the art label-constrained reachability query 'Landmark Indexing' (LI) algorithm, to two label-constrained reachability algorithms which we have developed ourselves. The first algorithm we refer to as the 'All Label Combinations' (ALC) algorithm. The second algorithm - SCC-ALC algorithm - is an extension of the initial ALC algorithm which utilises Strongly Connected Components (SCC) to speed up query response time. These algorithms will be compared both in terms of time complexity and memory space efficiency. Experiment results found that the ALC+SCC algorithm was capable of responding to LCR queries in response times comparable to LI, while using up to 392 less memory space on real world graphs. Furthermore, ALC+SCC was capable of responding to LCR queries on undirected graphs in constant time. On directed graphs where a single SCC contained a large number of the networks nodes, ALC+SCC was also able to respond to LCR queries in response times that was faster than LI.

1 Introduction

As graph databases and real world network graphs continue to increase in popularity and expand in size, the ability to efficiently and reliably query information from these graphs

will also increase in importance. One such family of queries which are garnering notable amounts of research of late are *reachability queries*. Reachability queries work as follows. Given any two nodes in a graph v_1 and v_2, does a path exist between the two nodes. As outlined by [17] in their survey of reachability queries, these type of queries are used across a wide range of use cases including web usage mining, web site analysis, and biological network analysis. For example in bioinformatics, graph vertices may be either molecules, reactions, or interactions in living cells. Edges in such a biological network will represent how these various biological elements interact with each other. In this context, reachablity queries help biologists determine which genes are influenced (either directly or indirectly) by any given molecule.

Label-constrained reachability (LCR) queries represent a subset of reachability queries that have been attracting much interest of late. This subset of reachability queries is concerned with finding a path between two vertices in a graph, using only a subset of the labels that exist between vertices in the graph. For example, in the context of a social network, people may be connected with each other via 'likes', 'comments', or 'friendship.' In this context, an example of a graph reachability query would be to investigate whether there exists a path between any two people (vertices) in the social network. On the other hand, an example of a label constrained reachability query would be to investigate whether there exists a path between two people in the social network via only 'likes' and 'comments'.

As highlighted by Zho et al.[19], there are two extreme solutions to answering any reachability query. They claim that the first approach is to create the transitive closure of a graph across all potential subsets of edge labels. This would enable all reachability queries to be answered extremely efficiently time wise. The other possible approach is to perform a Breadth First Search (BFS) over the graph in response to any given query at runtime. As pointed out by Zho and colleagues[19] Both these approaches on their own are not feasible in real world graphs at scale. Even in the context of standard reachability queries, the closure approach would require $O([V]^2)$ memory space to store each transitive closure to respond to queries. Furthermore, the BFS approach would require $O([V + E])$ time complexity in order to respond to each individual query. As such, these two methods can not work on real world massive graphs on their own as either their time or space complexity would be too large to work in practice. Any practical reachability query solution should aim to strike some sort of compromise between the two solutions. This trade off can be summarized in Figure 1 below.

The Landmark Indexing (LI) algorithm recently proposed by Valstar, Fletcher and Yuichi in 2017[15] aims to provide such a compromise for solving LCR queries. Landmark Indexing works by constructing indices for a small group of vertices. These vertices are subsequently referred to as "Landmarks". Landmarks are chosen based on their degree, with larger degree nodes being more likely to be chosen as landmarks. Therefore, given a graph G=(V, E, L) (see Table 1 for notation meaning), for each landmark vertex v, an index is stored of $(w, L') \in V \times 2^{\ell}$ if there exists an L'-path from v to w. For queries where the source vertex is a landmark, the query can be responded to by simply checking the index. Otherwise, a BFS is conducted across the restricted labeled graph until a landmark is reached. In this regard, the Landmark Indexing algorithm can be seen as somewhat of a compromise between time

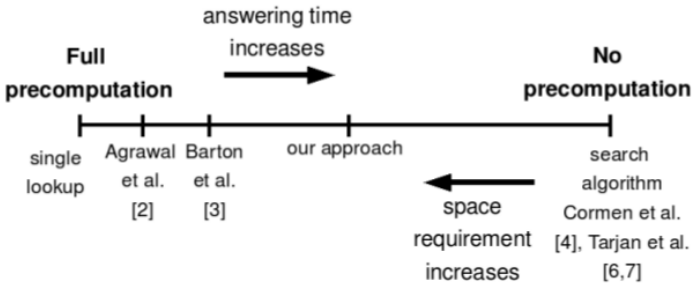

Figure 1: Figure taken from [5] illustrating the trade off between speed and memory space in reachability queries.

efficiency and memory efficiency approaches.

This paper proposes an alternative method for responding to LCR queries. We refer to this approach as the *All Labels Combinations* (ALC) algorithm. However initial experimentation showed that while this method was effective in reducing memory space usage for LCR queries, response times were not fast enough to make this algorithm practically useful on large graphs. In order to address this weakness, we subsequently expanded the ALC method to use precomputed Strongly Connected Components (SCC)[6] in order to improve query response time. We refer to this version of ALC augmented using SCC as ALC-SCC. We shall explain this new proposed methodology later in the 'Proposed Method' section of this paper.

The main problem being solved by the present research proposal is as follows:

- *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

This is an important problem, because as discussed before, reachability queries have important use cases ranging greatly in diversity from biological networks to social networks. However depending on the size of the graph, different approaches may be more or less feasible. The best, most efficient way to run a LCR query on a graph remains an open question in computer science. As such, testing different LCR algorithms on graphs of different sizes is vital in order to understand which queries work best on small, medium and large sized graphs with varying characteristics.

The contributions of this project are as follows:

- We aim to offer two alternative LCR method to the current state of the art Landmark Indexing algorithm for performing LCR queries.

- In terms of the memory space/time complexity trade off, the proposed algorithm ALC and ALC-SCC algorithms are vastly more space efficient than the Landmark Indexing approach. Furthermore, the ALC-SCC algorithm provides query speed that is comparable to Landmark Indexing
- We aim for our proposed ALC an ALC-SCC methods to be more suitable for queries on extremely large graphs, where the memory demands of the Landmark Indexing approach may make this approach unfeasible.

Symbol	Definition
G	Graph being queried
V	A finite set of vertices in the graph
E	The set of all edges (v, w, L') where an edges exists from v to w with some label L'
L	Set of all labels in the graph
ℓ	Length of L
L'	Any subset of labels from L

Table 1: Symbols and definitions

2 Survey

Next we list the papers that each member read, along with their summary and critique.

2.1 Papers read by Mohammad Sadegh Najafi

First and foremost, apart from the Landmark indexing, which its evaluation is the main focus of this work, we largely review all recent developments in the Label Constrained Reachability area.

2.1.1 Shortest Path Label-Constrained [4]

- *Main idea*: the main focus of this study is to present label-constrained shortest path(LCSP) method in order to discover the shortest path between the source and target only with constrained labels. with regard to the suggested method, two approaches with an approximation of finding the precise shortest path while false negative is possible.
- *Use for our project*: the paper proposes a recent improvement on Label Constrained Reachability queries. LCSP provides a general methodology for processing reachability queries. The author proposes two methods to evaluate LCSP queries. The proposed methods are an approximation of the actual distance.
- *Shortcomings*: due to the fact that the final result of the evaluation is an approximation of the actual distance, they are not useful as well as we do not consider this work on our discussion.

2.1.2 Tree-based Index Framework [7]

- *Main idea*: bauilding a full transitive closure (TC) of the underlying graph to answer LCR queries. A tree-based index framework which contains a spanning tree T and a partial transitive closure PT of the graph. PT and T have the information to derive full TC.
- *Use for our project*: the Landmark Indexing (LI) method has some properties in common with Jin's method. the better understanding the structure of this work would help us accurately analyze properties of LI. LI does not construct a full TC and it attempts to keep the balance between BFS and constructing full TC. As a result, the running time for query evaluation declines.
- *Shortcomings*: there is a real downside for Jin's method. The offered method is not practical for dense graphs owing to the comparatively small size of spanning-tree T in comparison with the whole graph.

2.1.3 Approximate Regular-simple-path Reachability [16]

- *Main idea*: all recent techniques have limitations to some extent form network growth to efficiency for a limited number of labels. They generalize regular simple path queries for LCR. They aim to develop an almost optimal index-free to evaluate LCR.
- *Use for our project*: this novel approach evaluates Landmark Indexing as well as propose a state-of-the-art model to dynamically adapt the method to the network growth. Our expectation for the ACL is the more comparable time complexity compares to Landmark Indexing. With regards to this innovative approach, we can allocate further researches to decrease the time complexity of ACL.
- *Shortcomings*: their method has improved several downsides of previous works including adaptability to network growth and independence from pre-computing processes.

2.2 Papers read by Chris Hickey

The first paper I read was a general introduction to reachability queries provided by Yu and Cheng [17].

- *Main idea*: This is simply an introductory chapter into reachability queries in general. It also offers a very clear definition of the problem: Given directed graph $G = (V, E)$ with n nodes and m edges, you are asked to find a path between two nodes v and u. If a path exists between the two nodes, the query will return true, otherwise it will return false. This can be redefined as, given two nodes v and u, and the edge transitive closure TC of graph G, is $(u, v) \in TC$. The rest of the chapter was dedicated to current best approaches on how to solve reachability queries in unlabeled constrained directed graphs.
- *Use for our project*: as somebody with no previous experience in reachability queries, this introductory chapter gave me an excellent insight into why reachability queries in general are important in domains ranging from social networks to bioinformatics.

The main takeaway I had from this chapter were the different time complexities, index construction time complexities and index size of the current best approaches to solving standard reachability queries. These complexities really drove home the point that reachability queries are all about striking a compromise between time complexity and space complexity. These can range drastically between a transitive closure approach, where query time is $O(1)$ but memory space is $O(n^2)$, to a 3-Hop Cover algorithm where query time is $O(logn + k)$ but memory space is $O(nk)$.

- *Shortcomings*: The authors fail to give examples as to which approaches are best suited to which graphs. For example, the authors stress that 2-hop and 3-hop approaches are difficult to compute. I presume this means such approaches are not suitable to rapidly changing graphs like social networks. I would have preferred if the authors gave specific examples of which reachability query approaches were best suited to different types of graphs. This question will be key to our research.

The second paper I read was about Land Mark Indexing as a solution to LCR queries[15]. This will be used as the benchmark algorithm for our project.

- *Main idea*: by selecting the vertices with the largest degree, and constructing an index such that given a query (s, t, L'), the query can be answered immediately if s is a landmark, the majority of label-constricted queries can by answered. This index is constructed by for every landmark index v, storing every pair $(u, L') \in V \times 2^{\ell}$, where L' is all possible subset labels for the label super set L. For queries where the source vertex is not a landmark, an optimized breath-first search algorithm is used until a landmark is found, then the query is responded to.
- *Use for our project*: As one of the more recent development in LCR queries, this serves as an excellent benchmark for our proposed algorithm. Indeed the entire goal of our project is to offer an improvement on this algorithm, at least for certain graphs.
- *Shortcomings*: One glaring issue with landmark indexing is it's performance for non landmarked nodes. The authors note that especially for LCR queries on unlandmarked nodes where the answer is 'false', the algorithm performs no better than BFS. Given the indexing memory space taken up by the landmarks, this algorithm represents a 'worst of both worlds' scenario for LCR queries which return false, performing bad both in terms of speed and memory space.

The third paper I read focused on attempts to optimize BFS [3]. BFS plays an instrumental role both in Landmark Indexing, and our proposed algorithm.

- *Main idea*: The authors extend the standard top down BFS as follows. In standard BFS, starting from the source vertex, all of the vertices at the same depth are visited before any vertex at the next next depth is visited. The authors combine this approach with a novel top up approach, where unvisited vertices attempt to find any parent among its neighbors which are still currently in the active 'frontier' of possible intermediary nodes between the source vertex and the target vertex. Experiments showed this approach to be between 2-5 times faster than normal BFS on real world social network graphs.

- *Use for our project*: Both our proposed ALC approach and the Landmark Indexing algorithms depend on BFS in some form or another. As such, understanding how to best implement BFS is vital to getting the best performance out of both algorithms
- *Shortcomings*: The authors explan why this approach is useful for 'small world' phenomenon graphs, like social networks. I would be interested to know if there are graphs where this hybrid Directional BFS approach is less useful than conventional BFS.

3 Proposed Method

In this section, we demonstrate all proposed methods and their performance and efficiency for answering LCR queries. We first explain the first proposed method called All Label Combinations (ALC), its related query algorithm, and the time and space complexity involved in the algorithm. Then we, outline the improvements made to this algorithm through combining ALC with Strongly Connected Components. We refer to this augmented version of ALC as ALC-SCC. We then discuss the time and space complexity related to this ALC-SCC approach. Prior to experimentation on proposed methods, we expected our approaches to have the following contrasts when compared to the Landmark Indexing algorithm:

- *Memory Space*: Both proposed algorithms will be much more efficient in terms of memory space when compared to Landmark Indexing.
- *More suitable for sparse networks*: Unlike Landmark Indexing, both proposed method treat all nodes equally. As such, we expect our methods will perform equally well on sparse graphs or on graphs where no vertex has a notably higher degree than any other vertex.
- *Speed Performance*: The Landmark Indexing algorithm was expected to perform faster than the ALC algorithm. However we expect these query time issues will be addressed by the second proposed ALC-SCC algorithm. We expect that the ALC-SCC implementation would be able to provide query response time comparable to Landmark indexing.

3.1 All Label Combinations

In this section, we explain the overall intuition of the ALC approach in Section 3.1.1, then its decomposing algorithm Section 3.1.2 and its query algorithm Section 3.1.3 . A detail of time complexity and space usage are provided in Section 3.1.4.

3.1.1 Overall Idea

The main contribution of this paper is to offer an alternative method to Landmark Indexing for LCR queries. The first step in the implementation of such an alternative method will work as follows: Given graph $G = (V, E, L)$, the proposed All Label Combinations (ALC) method will construct multiple graphs of all possible combinations of labels. This leads to the construction of $2^l - 1$ unlabeled subgraphs. For example, if the total set of labels for a

graph is $L = \{a, b, c\}$ then the ALC algorithm will first take all possible label combinations $\{a\}$,$\{b\}$,$\{c\}$,$\{a, b\}$,$\{a, c\}$,$\{b, c\}$ $\{a, b, c\}$, creating graphs based exclusively on edges in the label subsets. Then, a given query (s, t, L'), the pre computed subgraph of G corresponding to the the subset of labels specified in L' is first brought forward. BFS is then computed over this subgraph in order to establish whether a path exists between source node s and destination t.

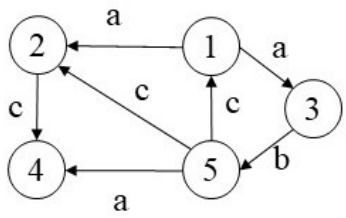

Figure 2: Example graph with 5 vertices, 7 edges and labels $L = \{a, b, c\}$. Figure 2 shows decomposed unlabeld subgraphs of this graph.

3.1.2 Decomposing Algorithm

We present an easily implementable space-efficient algorithm to construct separate unlabeled subgraphs of the given edge-labeled graph for any possible combination of labels. The algorithm builds combinations of the label set in a naive recursive manner. We define the stored combinations of the label set as $CombinationSet[0...2^l - 1]$. Each entry in the $CombinationSet$ saves one of the $2^l - 1$ label sets. Let the label set, subgraph pair $ALC < CombinationSet\ ls$, $subgraph\ G' >$ denote the final result of the algorithm which keeps an unlabeled subgraph corresponding to its related label combination. These $ALC < ls$, $G' >$ pairs are constructed as follows. For each $ls \in CombinationSet$, each edge in graph G is iterated over. For any edge E encountered during the edge iteration, if the label for $E \in ls$, this E is assigned to the corresponding $subgraph\ G'$. In practice, to reduce construction time, for any G' with ls of length greater than 1, the graph G' is created by combining each G' corresonding to every single label $l \in ls$. To illustrate, for the graph in figure 2, if the corresponding label set is $\{a, b\}$, the graph G' will be formed by joining G' from both $ALC < \{a\}, G' >$, and $ALC < \{b\}, G' >$. Algorithm 1 sketches the procedure which constructs the ultimate separate unlabeled subgraphs.

3.1.3 Query processing

The query algorithm (Algorithm 2) for ALC is straightforward and resembles the simple BFS approach. Apart from the additional step of choosing a suitable subgraph, every step is the same as the basic BFS. The query algorithm initiates with given reachability query (s, t, L'),

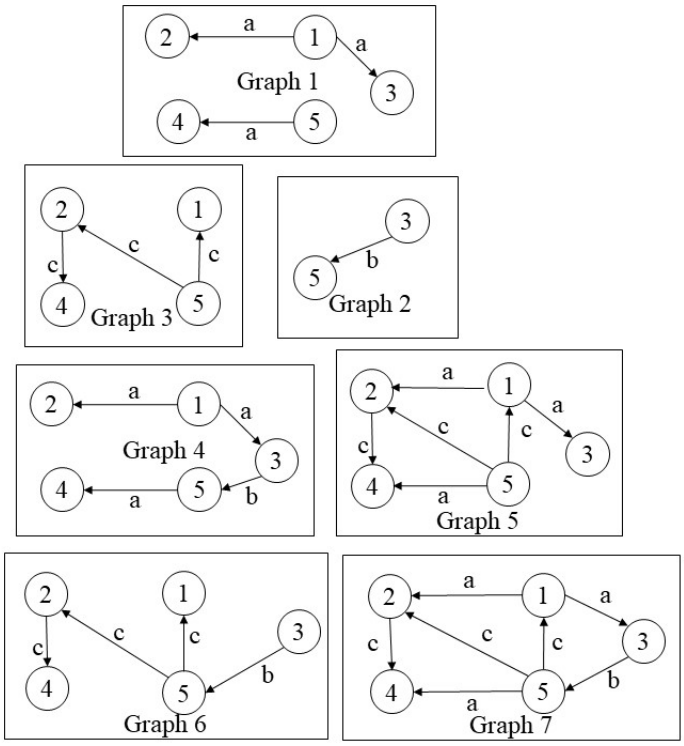

Figure 3: Graphs generated based on label combinations. Each individual block represents a single graph. The edge labels are not stored as part of these graphs.

and attempts to find the corresponding graph with regards to given L'. After finding the corresponding unlabeled G' subgraph, the query traverses through the graph by using the BFS algorithm. The algorithm determines whether the given target is reachable from the given start vertex. If the target is reachable, the query returns $true$, otherwise it returns $false$.

3.1.4 Time for Space Trade-off

The preprocessing construction time for the ALC algorithm is governed by the time taken to construct the 2^l unlabeled subgraphs. Constructing each of these subgraphs takes $O((n+m)\log l)$ time. Since the number of all labels combinations is $O(2^l)$, the overall construction time is $O(2^l(n+m)\log l)$. The query time is at most linear to the size of the subgraph corresponding to the query label set, which is at worst $O(n+m)$. The current work attempts to minimize space usage while achieving reasonable query time. The overall space usage of the algorithm is $O(2^l(n+m))$.

Algorithm 1 All Label Combinations

Input: A given Edge-Labeled Graph G and all label selections $CombinationSet[0...2^l - 1]$
Output: $2^l - 1$ Number of Separate Unlabeled Subgraphs Stored in Map Data Structure ALC $< CombinationSet\, ls,\ SubgraphG' >$

```
1: for each labelset ls ∈ CombinationSet do
2:     Initialize subgraph G';
3:     for Each edge e ∈ G.E do
4:         if e.label == ls then
5:             Add e to G'
6:         end if
7:     end for
8:     Add < ls, G' > to ALC
9: end for
```

3.2 Synthesis of Strongly Connected Components and All Labels Combinations

In this section, Section 3.2.1 illustrates the extended version of ALC termed ALC-SCC. Section 3.2.2, Section 3.2.3, and Section 3.2.4 present the Strongly Connected Components-ALC (ALC-SCC) algorithm, associated query algorithm, and space usage and expected time complexity respectively.

3.2.1 General Strongly Connected Components Notion

The use of strongly connected components (SCC) [12] to compute label-constraint reachability queries was first introduced by Zou et al[19]. The proposed method first decomposes the underlying directed edge-labeled graph into several strongly connected components, then a bipartite graph B_i is generated by replacements of maximal connected components. B_i includes all $in - portal$ and $out - portal$ vertices in C_i. With regards to this suggested definition, $in-portal$ vertices must have at least one incoming edge from a vertex outside of C_i and in contrast, $out-portal$ must have at least one outgoing edge to a vertex outside of C_i.

Definition 4.1: A subset of vertices C in G is called strongly connected as long as there is a path between all pairs of vertices u and $v \in C$. For all pairs of vertices u, v in the set of strongly connected vertices C there exists both $u \to v$ and $u \to v$. As such, all pairs of vertices u, v are reachable from each other.

The main drawback of the ALC approach is that query response time for both *true* queries and $false$ queries are long in comparison to its competitors. We attempt to overcome this issue by integrating Strongly Connected Components to the ALC algorithm. For any given edge-labeled graph, we first build separate unlabeled subgraphs with the help of ALC. Next, for every single subgraph, our algorithm finds all strongly connected components in the

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**.

```
1: Find graph G' corresponding to ls form ALC < ls, G' >
2: for each v ∈ G'.V do
3:     v.status= false
4: end for
5: Queue q is an empty queue
6: q.push(s)
7: while q is not empty do
8:     u = q.pop()
9:     u.status= true
10:    if u == t then
11:        return true
12:    end if
13:    for each w ∈ G'.Adj[u] do
14:        if w.status == false then
15:            q.push(v)
16:        end if
17:    end for
18: end while
19: return false
```

graph. We can simply replace all strongly connected components by single SCC vertices since there are no labels in the graphs post ALC. This means the transformation does not lead to missing edge labels in each strongly connected component. The final result of applying SCC on each unlabeled graph is a directed graph where each vertex is a strongly connected component. Figure 3 shows an example of the terminal result of a graph after applying both ALC and SCC. Figure 3(a) represents a synthetic edge-labeled graph G. Figure 3(b) is the result of the ALC method for the combination $\{b\}$. Figure 3(c) presents the outcome of employed SCC on the corresponding unlabeled graph relating to the combination $\{b\}$.

3.2.2 Strongly Connected Component ALC Algorithm

The pseudocode of the integrated ALC and SCC algorithm is shown in Algorithm 3. The algorithm takes the unlabeled graph G' from the result of ALC as input and finds all maximal strongly connected components in G'. The algorithm for finding SCC's is mainly based on DFS [6]. Algorithm 3 performs a depth-first search (DFS) two times. The first DFS is done to find the "finish time" for all vertices in the graph, where higher finishing times correspond to deeper nodes. Then DFS is performed on the transpose graph of G^T, with DFS being performed on nodes in decreasing order of finishing time. Since both G and G^T have exactly the same Strongly Connected Components [6], and DFS was performed in order of decreasing

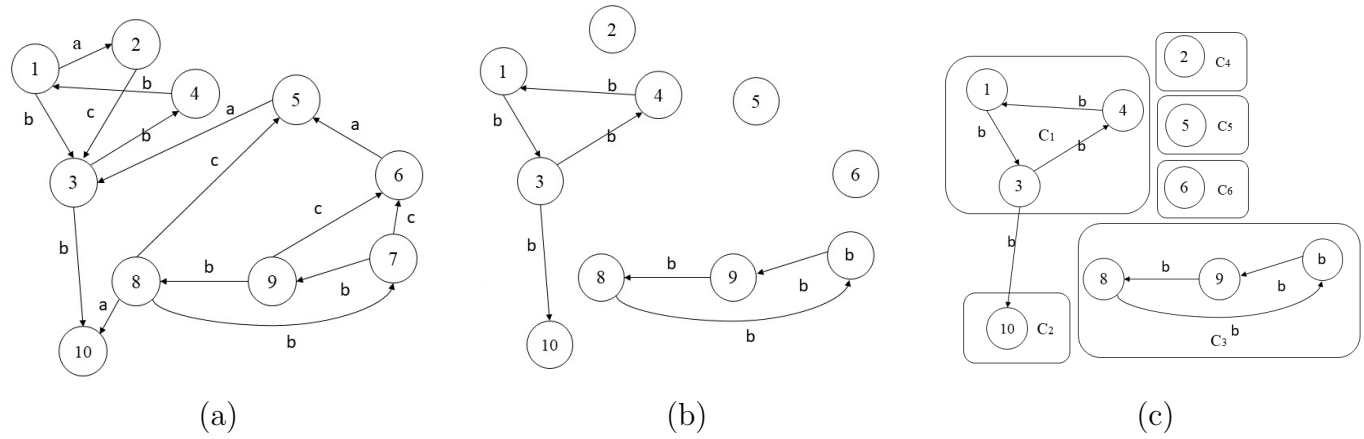

Figure 4: (a) A synthetic edge-Labeled graph (b) The result of ALC for selection $\{b\}$ (c) determined the corresponding graph SCC, C_is shows strongly connected components.

finishing time, the SCC's found from G^T, represent the SCC of maximum possible size in G. The result will be a Directed Acyclical Graph, either in the form of a single tree or a forest, where each node represents a SCC. Provided that all vertices are reachable from DFS starting vertex, the result will be a single tree. Otherwise, the result is a forest. First, the algorithm decomposes the graph into various strongly connected components, and eventually combines the final result according to the connections among components. In order to speed up query processing, the algorithm stores additional data for each strongly connected component to check whether the component is isolated or has an $out-portal$ edge. This data is stored in a boolean value, $outPortal$.

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 $< LabelSet\,ls,\ SubgraphG' >$

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

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

3.2.3 Query Algorithm

Algorithm 4 shows pseudo-code for LCR evaluation over two vertices s as a source and t as a target under the given constrained labelset $L' \subseteq L$ using the SCC-ALC algorithm. The algorithm will return $true$ if it can find a target t in its travel started form s, otherwise it will return $false$. When the algorithm triggers, first, it finds the corresponding subgraph G' to the labelset l in StronglyConnectedComponentsALC. Next, the algorithm compares the given source s and target t's SCC component ID's, which have been stored as $SCCid$ in Algorithm 3. If they belong to the same strongly connected components the algorithm returns $true$. If they are not in the identical strongly connected components the algorithm performs BFS over the unlabeled graph G' until it finds a vertex with the same $SCCid$ as target vertex t. In case the graph is undirected, the algorithm just runs lines 2-4. If the strongly connected component has an $out-portal$ edge the algorithm continues. Otherwise, the algorithm returns $false$. The algorithm store a boolean variable $outPortal$ for each component.

3.2.4 Expected Space and Time Complexity

The combined approach here is an extension of the ALC approach which is described in the 3.1 section. The Strongly connected component is called once for each individual $2^l - 1$ graphs. The algorithm performs normal DFS twice which is linear to the number of nodes and edges of each graph that are in the worse case n and m. Owing to the fact that the data structure used to store graph is an adjacency list, performing used algorithm (Algorithm 3) for each graph takes $O(n + m)$ time. Therefore, the construction time of the proposed algorithm is $O((2^l(n + m) \log l))$. SCC algorithm stores the SCCs of $2^l - 1$ graphs, each with size at most $O(n + m)$ which are the results of Algorithm 1. Consequently, the query algorithm manages to perform the BFS for an individual graph in $O((n + m))$.

4 Experiments

In this section, we talk about our experiments, datasets used in experiments, set up, query generation, and details regarding each different experiment.

4.1 Combined Method Versus Landmark Indexing

In this section, we evaluate our approaches compared to Landmark Indexing. In our experiments, we used both synthetic graphs and real-world graphs. In order to generate synthetic graphs to test both algorithms against, we used SNAP [10, 9]. The purpose is to judge the performance of our approach in three distinct criteria:

- *Response Time*: Time of response in milliseconds (ms).
- *Memory Usage*: Amount of memory space required to service queries in Kilobyte (Kb).

Algorithm 4 Query Algorithm for Synthesis of SCC and ALC Approach

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

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

```
1: Find graph G'' corresponding to l form StronglyConnectedComponentsALC < ls, G' >
2: if s.SSCid == t.SSCid then
3:     return true
4: end if
5: if s.SSCid.oudPortal == false then
6:     return False
7: end if
8: for each v ∈ G''.V do
9:     v.status= false
10: end for
11: Queue q is an empty queue
12: q.push(s)
13: while q is not empty do
14:     u = q.pop()
15:     u.status= true
16:     if u.SSCid == t.SSCid then
17:         return true
18:     end if
19:     if s.SSCid.oudPortal == false then
20:         return False
21:     end if
22:     for each w ∈ G''.Adj[u] do
23:         if w.status == false then
24:             q.push(v)
25:         end if
26:     end for
27: end while
28: return false
```

- *Construction Time*: Length of pre-processing time required to respond to queries in second (s).

Our two approaches, ALC and ALC-SCC were implemented using C++.

4.1.1 Setup

The experimental settings were a Windows with Intel i7-6700 processor and 32 GB memory. Experiments were all single-threaded. The number of landmarks for response time experiments was set to be 50 percent of the number of vertices.

4.1.2 Datasets

Our experiments used two types of datasets. We utilize both synthetic and real data sets. Synthetic graphs are generated by using SNAP with Scale-Free Model (SF) graph generation. The direction and labels are augmented to the synthetic graphs randomly. For synthetic graphs, the size of label sets is either 4 or 8. The reason why we choose preferential attachment over other graph generation models is that the Scale-Free Model follows the power-law connectivity distribution [2] which means they are likely to be near to real-world graphs. The power-law connectivity distribution parameter α is set to between 2.0 and 3.0 to simulate real-world networks [1]. For graph generation, The parameter is fixed to be α= 2.5. In addition, we also employ nine different real datasets in our experiments. Data sets are chosen from a wide variety of sources, including those graphs which come with SNAP [10]. During the testing process, we augment labels to several datasets with the same principle used for synthetic graph generation. Label size is set to $\ell = 8$ with distribution parameter α=2.5. **BioGrid**, **StringsFC**, and **StringsHS** were originally undirected due to the structure of the graph representing protein relations. Table 2 shows all information regarding data sets used in our experiments.

4.1.3 Queries

Four types of query sets were used in these experiments, two $'True'$ query sets, and two $'False'$ query set. The query set contained either 1000 $True$-queries or 1000 $False$-queries. For both $'True'$ and $'False'$-queries sets, we considered two different label sizes $\ell/4$ and $\ell - 2$. For query generation, we followed the same methodology as the authors of Landmark Indexing. The query generation procedure starts by selecting the number of queries to generate in a set (in this case, $t = 1000$) and selecting a random vertex $v \in V$. Then a random number r is generated between $50 + \log n$ and $50 + n/50$. After that, the procedure begins a loop for $t/100$ times. In each round, the procedure chooses another random $u \in V$ and generates 10 label sets$L_1, ..., L_{10}$. For each label set, the procedure runs BFS to check that label exists between v and u. In the BFS traverse, the procedure counts the number of seen vertices and stores this information as r'. r and r' are designed to check the steps. Therefore, if $r' < r$, the procedure runs the loop from the beginning and diregards the query. The query $q = (v, u, L_i)$ stores in $True$-queries if and only if there is a path between v

Table 2: 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[1]	1400	2900	4	No	Directed
Advogato[8]	14010	70472	4	No	Directed
webGoogle [10]	875K	5.1M	8	Yes	Directed
webStanford [10]	281K	2.3M	8	Yes	Directed
WebBerkstan [10]	658K	7.6M	8	Yes	Directed
Youtube[18]	15K	10.7M	5	No	Directed
StringFC[14]	15K	2M	7	No	Undirected
BioGrid[11]	64k	1.5M	7	No	Undirected
StringHS[13]	16K	1.2M	7	No	Undirected

and u with L_i, otherwise, the query is stored as a $False$-query. The procedure disregards duplicates and runs until we find all t $True$-queries and t $False$-queries. The main purpose of performing the procedure in this way is to assure that we do not have the same queries with the same vertices. This way we consider as many vertices as possible and also attempt to find vertices from different parts of the graph that are not restricted to small parts of the graph.

4.2 Performance of Combined Method

In this section, we compare the performance of the ALC method, Synthesis of Strongly Connected Components and All Labels Combinations method (ALC+SCC) with the Landmark Indexing (LI) method on both real and synthetic datasets mentioned on Table 2. Comprehensive reports regarding query response time, space usage, and construction time are provided.

4.2.1 Response Time

The first experiment is designed to measure the response time of each approach. Figure 5-8 depict response times for each approach for $True$-queries and $False$-queries with different label sizes.

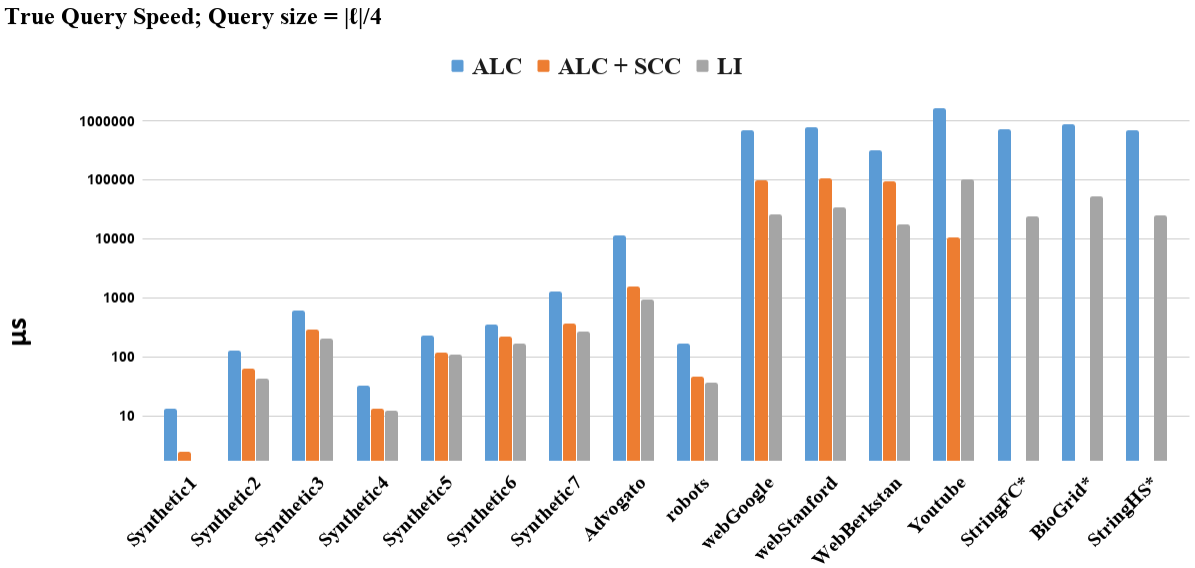

Figure 5: *True*-query times for ALC, ALC+SCC and LI algorithms, measured in μs. Here ℓ refers to size of label set, and $\ell/4$ refers to number of labels in the query.

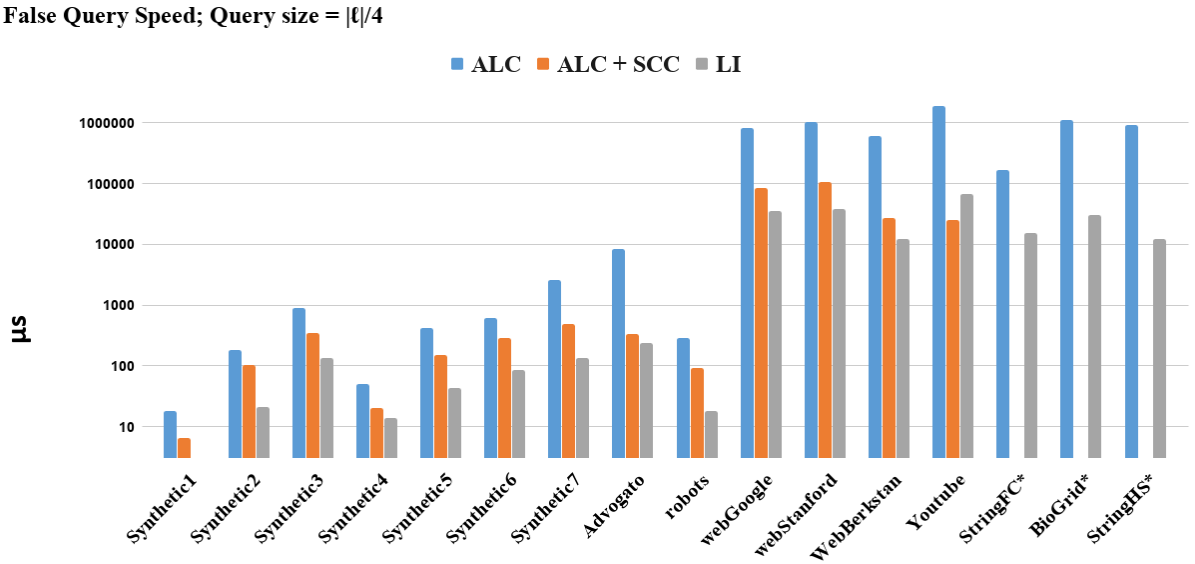

Figure 6: *False*-query times for ALC, ALC+SCC and LI algorithms, measured in μs. Here ℓ refers to size of label set, and $\ell/4$ refers to number of labels in the query.

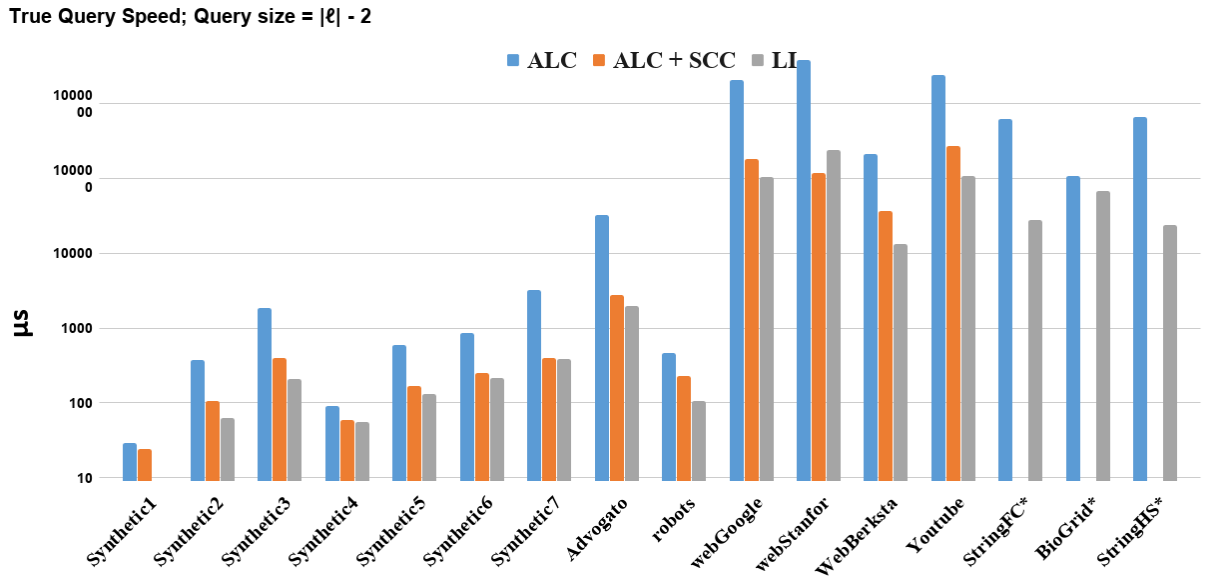

Figure 7: *True*-query times for ALC, ALC+SCC,and LI algorithms, measured in μs. Here ℓ refers to size of label set, and $\ell - 2$ refers to number of labels in the query.

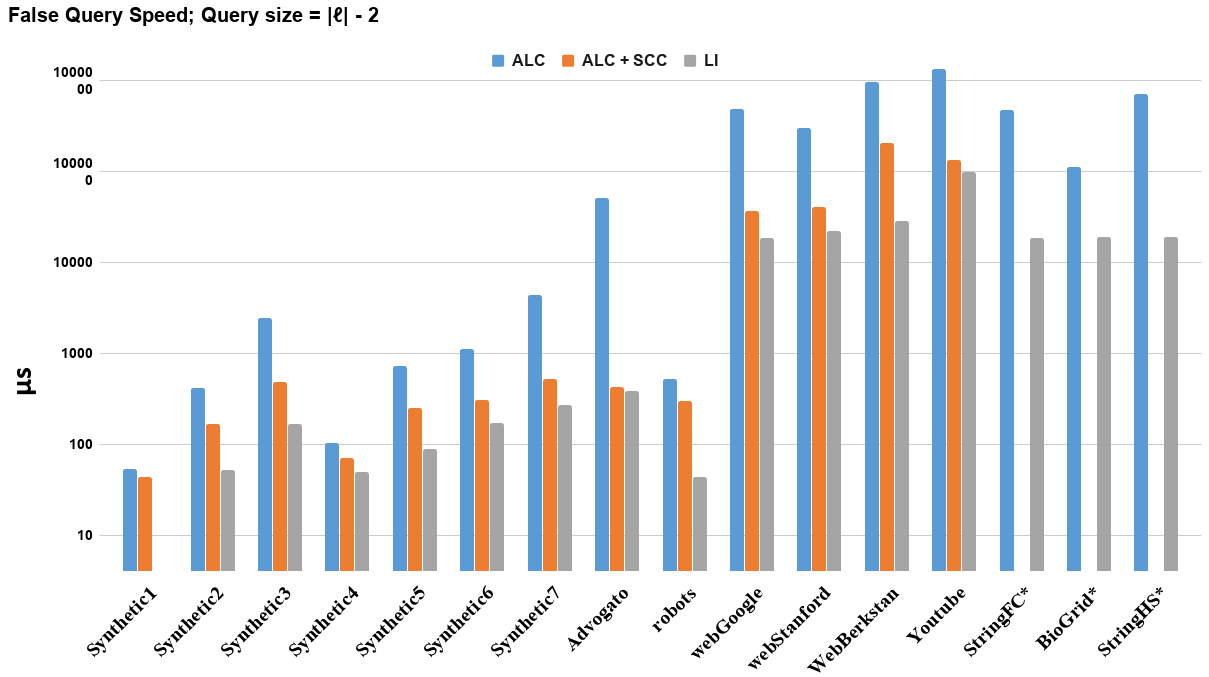

Figure 8: *False*-query times for ALC, ALC+SCC,and LI algorithms, measured in μs. Here ℓ refers to size of label set, and $\ell - 2$ refers to number of labels in the query.

The LI algorithm consistently outperformed the ALC algorithm in both *true* and $false$ queries. Here, the Landmark Index algorithm was implemented such that k vertices in the graph were landmarked, where $k > \frac{V}{2}$. As can be seen in Figure 5-8, the proposed ALC method is between 4-24 times slower than Landmark indexing for $True$ queries and 6-134 times slower for $False$ queries. The ALC algorithm performed consistently notably worse for $False$ queries when benchmarked in comparison to LI. This makes sense, as in LI as soon as any landmarked node is reached, the algorithm can immediately return $False$ if the target node t is not reachable from the landmarked node. In contrast, the entire unlabeled subgraph G' must be traversed in ALC before the algorithm can return $False$. It is worth mentioning that the query response time in ALC does not depend upon constraint label size while the constraint label size has an adverse impact on LI. Figure 5-8 depicts the response time of each approach individually. The y-axis in the figure is log normalized in order to make the small numbers visible in contrast to large numbers for larger graphs.

To improve the response time, the ALC + SCC method was designed. The algorithm sacrifices construction time to improve query time. The ALC +SCC approach performs well for undirected graphs. The approach can return the query response in constant time for both $True$ and $False$ queries on undirected graphs due to the fact that after employing Algorithm 3 for undirected graphs, the outcome would be distinct SCC subgraphs. Each subgraph is strongly connected inside the local subgraph and completely disconnected from the rest of the graph. Consequently, ALC+SCC is strictly faster than LI for undirected graphs. For directed graphs, the query response time largely depends on the number of strongly connected components in the graph. Therefore, the response time is relatively comparable to LI. Experimentation found that ALC+SCC performed worse on non-dense graphs like **WebStanford**. On this graph, ALC+SCC was 5 times slower than LI for $True$ queries and 6 times slower for $False$ queries. The performance difference between $False$ and $True$ queries, which was substantial for ALC, are improved in ALC+SCC by storing extra information about out-portal vertices for each SCC destination. If not out-portal exists in an SCC, the graph can return $False$. For ALC+SCC, note that the constraint label size in the query does not have a destructive effect on query response time.

4.2.2 Memory Usage

In the second experiment, we aim to investigate the memory usage of each method. Figure 9-10 show the individual space usage of the 4 approaches. The storage usage in LI relies on the number of landmarks. The details of memory usage are described as follows:

The proposed ALC algorithm required significantly less memory space when responding to LCR queries than the alternative Landmark Indexing method. As can be seen in Figure 8, the ALC method used between 1.5 and 46 times less memory space when compared to the LI algorithm. The difference in memory space usage depends on the number of landmarks k used by the algorithm, the size of the graph, and the total number of labels in the graph.

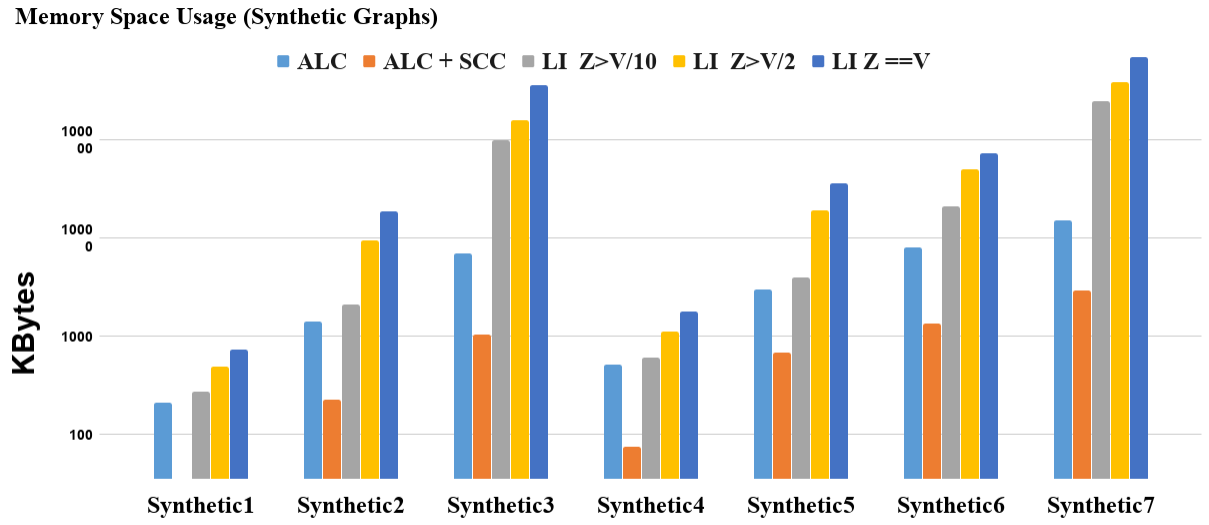

Figure 9: Memory usage for both ACL and LI algorithms, measured in Kb on synthectic graphs created through SNAP.

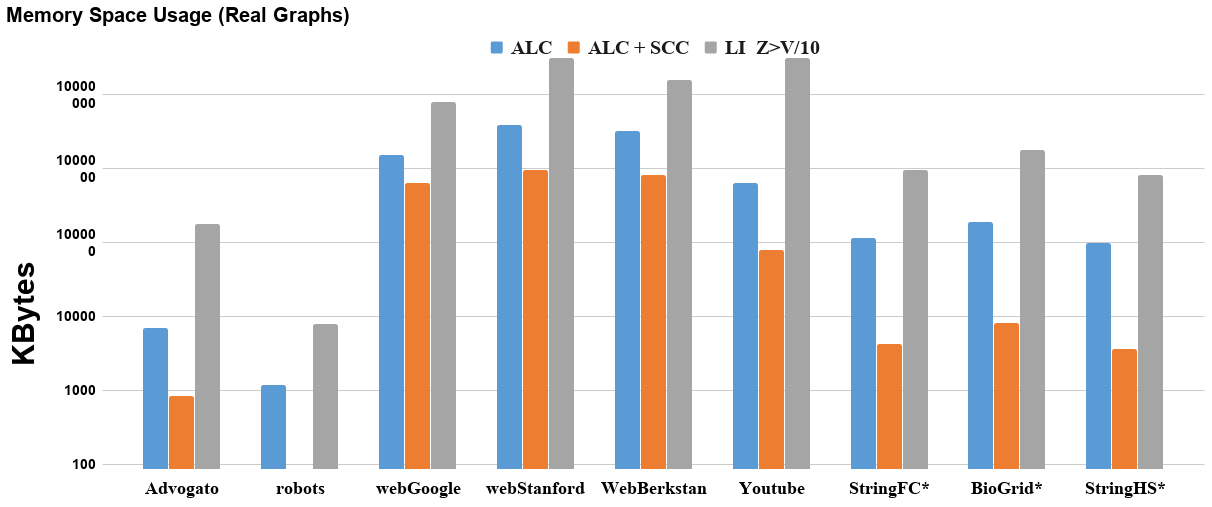

Figure 10: Memory usage for both ACL and LI algorithms, measured in Kb on real world graphs.

The ALC algorithm had the least notable memory usage benefits over LI for small graphs **Synthetic 1**. The ALC algorithm had the most notable memory usage benefits over LI for **WebBerkstan**. When compared to the version of LI with $k > \frac{V}{2}$, ALC consistently used significantly less memory space in all experiments.

ALC+SCC was intended to improve response time in a manner that could preserve the memory space usage benefits associated with ALC. Strongly Connected Components are perfect for this purpose, as they both reduce the amount of vertices that must be checked in BFS while also compressing the graph. The ALC+SCC approach reduces space usage to the extent that in the **Youtube** dataset, it used 392 times less memory space than LI with $k > \frac{V}{10}$. In the **StringFC** test case, the ALC+SCC implementation can store all necessary information for answering the query with 228 times less memory space usage than LI. However in the case of graphs like **WebBerkstan**, due to the fact that the data set is not dense and does not have a lot of strongly connected components, memory space usage is only 20 times better than LI with $k > \frac{V}{2}$. The extent to which ALC+SCC can improve performance is determined by label size ℓ and the number of strongly connected components in the graph.

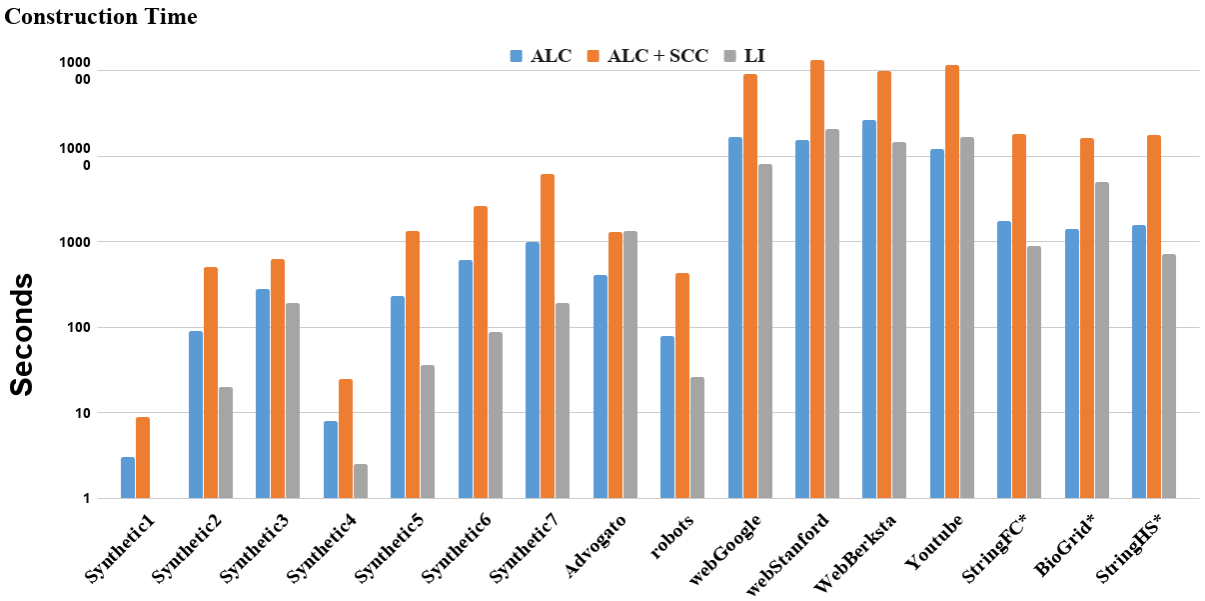

Figure 11: Construction time for prepossessing of both ALC and LI algorithms, measured in s. Here ℓ refers to the number of graph labels. LI graph constructed with landmarks $Z > \frac{V}{10}$.

4.2.3 Construction Time

In the third experiment, we report construction time for all approaches. Note that the construction times that are mentioned for LI are based on using 10% of all nodes in the graph as landmarks. Figure 11 depicts all construction times. The y-axis is log normalized.

The proposed ALC algorithm takes between 3- 7 times longer construction time when compared to the LI algorithm with 10% of Vertices designated as landmarks (See Figure 6). ALC benchmarked best in comparison to LI on **Youtube**. ALC benchmarked worst in comparison to LI on **Synthetic 7**. The reason why **Youtube** construction was fast is that the label size of the graph is small in comparison to other real graphs. ALC construction time is particularly slow for graph **Synthetic 7** when compared to LI. This is because **Synthetic 7** graph is dense and LI can perform well on dense graphs.

As intuition would indicate, applying the Strongly Connected Components technique adds more overhead in construction time over ALC. This additional construction time is a tradeoff to gain both space usage advantages and response time performance benefits. In some cases, ALC+SCC construction time is relatively comparable to LI. However on large real world graphs, ALC+SCC can be very high when compared to LI, due to the size of the network and also the size of the label set. This can be seen on the **StringFC** graph, where construction time was up to 20 times higher when compared to LI. **StringFC** has fewer strongly connected components than other graphs and an $\ell = 7$. As such, construction time for ALC+SCC is strongly influenced by the size of the label set, the size and density of the graph, and the number of strongly connected components in the graph.

5 Conclusions

In this paper, we introduce a simple approach to answer label constrained reachability query.

5.0.1 Innovations of the Present Work

- The paper proposes two new algorithms, called the All Labels Combined algorithm and the All Labels Combined- Strongly Connected Component Algorithm for solving Label Constrained Reachability queries.
- The new proposed ALC algorithm is highly optimized for memory efficiency. This algorithm is at least 16 times more memory efficient than the Landmark Indexing Algorithm, even when only 10% of nodes are designated as landmarks.
- Integrating Strongly Connected Components with ALC further reduces memory space usage, leading to the ALC+SCC method requiring up to 392 times less memory space usage when compared to LI on large real world graphs, even when only 10% of nodes are designated as landmarks.

- The proposed ALC+SCC algorithm achieves these memory space usage improvements, while also being able to provide query response times which are comparable to LI.
- Application of ALC could be used for evaluation of practical query languages such as openCypher and SPARQL1.1.

The novel ALC+SCC approach optimizes memory consumption, while also providing query speed which is comparable to the LI algorithm. Experimental results show that this approach uses much smaller amounts of memory space than the landmark indexing method, while providing query response speeds which are roughly equivalent to LI, even on very large real world graphs. Furthermore, ALC+SCC can respond to LCR queries on undirected graphs in constant time. Future work could investigate integrating LI with our proposed ALC+SCC approach in order to further enhance performance. In terms of construction time, both the proposed ALC and ALC+SCC method take considerably longer than the LI algorithm. As such, the proposed methods may not be suitable for quickly changing graphs. However, this is a price we are willing to pay in order to create a memory efficient algorithm capable of responding to LRC queries in a reasonable response time. Future work could also be directed towards improving construction time for ALC+SCC.

References

[1] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[2] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[3] Scott Beamer, Krste Asanović, and David Patterson. Direction-optimizing breadth-first search. *Scientific Programming*, 21(3-4):137–148, 2013.

[4] Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Antti Ukkonen. Distance oracles in edge-labeled graphs. In *EDBT*, pages 547–558, 2014.

[5] Panagiotis Bouros, Theodore Dalamagas, Spiros Skiadopoulos, and Timos Sellis. Evaluating "find a path" reachability queries. In *Proceedings of the European Conference on Artificial Intelligence (ECAI) Workshop on Spatial and Temporal Reasoning (Patras, Greece)*, 2008.

[6] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[7] Ruoming Jin, Hui Hong, Haixun Wang, Ning Ruan, and Yang Xiang. Computing label-constraint reachability in graph databases. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 123–134. ACM, 2010.

[8] Jérôme Kunegis. Konect: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1343–1350. ACM, 2013.

[9] Jure Leskovec and Rok Sosič. A general purpose network analysis and graph mining library, 2014.

[10] Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1, 2016.

[11] Rose Oughtred, Andrew Chatr-aryamontri, Bobby-Joe Breitkreutz, Christie S Chang, Jennifer M Rust, Chandra L Theesfeld, Sven Heinicke, Ashton Breitkreutz, Daici Chen, Jodi Hirschman, et al. Biogrid: a resource for studying biological interactions in yeast. *Cold Spring Harbor Protocols*, 2016(1):pdb–top080754, 2016.

[12] Micha Sharir. A strong-connectivity algorithm and its applications in data flow analysis. *Computers & Mathematics with Applications*, 7(1):67–72, 1981.

[13] Damian Szklarczyk, Andrea Franceschini, Stefan Wyder, Kristoffer Forslund, Davide Heller, Jaime Huerta-Cepas, Milan Simonovic, Alexander Roth, Alberto Santos, Kalliopi P Tsafou, et al. String v10: protein–protein interaction networks, integrated over the tree of life. *Nucleic acids research*, 43(D1):D447–D452, 2014.

[14] Damian Szklarczyk, John H Morris, Helen Cook, Michael Kuhn, Stefan Wyder, Milan Simonovic, Alberto Santos, Nadezhda T Doncheva, Alexander Roth, Peer Bork, et al. The string database in 2017: quality-controlled protein–protein association networks, made broadly accessible. *Nucleic acids research*, page gkw937, 2016.

[15] Lucien DJ Valstar, George HL Fletcher, and Yuichi Yoshida. Landmark indexing for evaluation of label-constrained reachability queries. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 345–358. ACM, 2017.

[16] Sarisht Wadhwa, Anagh Prasad, Sayan Ranu, Amitabha Bagchi, and Srikanta Bedathur. Efficiently answering regular simple path queries on large labeled networks. *Age*, 3:v7, 2019.

[17] Jeffrey Xu Yu and Jiefeng Cheng. *Graph Reachability Queries: A Survey*, pages 181–215. Springer US, Boston, MA, 2010.

[18] R. Zafarani and H. Liu. Social computing data repository at ASU, 2009.

[19] Xu K. Yu J.X. Chen L. Xiao Y. Zou, L. and D." Zhao. Efficient processing of label-constraint reachability queries in large graphs. *Information Systems*, 40:47–66, 2014.

Contents