Answering reachability queries on streaming graphs

Gulay Unel, Florian Fischer and Barry Bishop

Semantic Technology Institute (STI) Innsbruck,
University of Innsbruck, Austria firstname.lastname@sti2.at

Abstract. Graph reachability is a fundamental problem in many applications, such as reasoning in lightweight formalisms, geographic navigation, XML/RDF/OWL query processing, etc. Many real world scenarios involve huge graphs and require fast algorithms to test for reachability between nodes. The problem becomes even more challenging when the graph is rapidly changing and received as a real-time stream of nodes and arcs. In this paper we will review current graph reachability algorithms and focus on how they can be adapted to the streaming setting. We will also outline a new algorithm for answering reachability queries on huge, rapidly changing graphs.

1 Introduction

The graph reachability problem is defined as: given two vertices u and v on a directed graph find out whether there is a path from u to v. The problem has been explored in depth in several research fields [1–6] and becomes more challenging when the reachability query is performed on rapidly changing graphs.

Some typical applications involve geographic navigation, traffic control, click streams, etc. The problem is also a fundamental step in many reasoning tasks based on various logical formalisms such as OWL, WSML, DL. The reason for this is that for a directed graph its reachability relation is also its transitive closure, which in turn can be considered a classical example of monotonic reasoning and used as a building block to facilitate more complex reasoning.

As a more specific use case we can consider Urban Computing [7], which is the application of pervasive computing to urban environments. The data in this application can be modeled as streams representing real objects such as cars, trains, crowds, etc. monitored at given locations. Reasoning on such streams can be very costly if we consider the amount of dynamically changing data. For instance, if we want to answer queries such as: list all the cars that traveled between location a and b and return the results periodically then we need efficient reachability query answering capabilities on rapidly changing graphs representing the movement of the traffic. The applicability of the problem is not limited to these applications since almost any structured data can be represented using graph structures, e.g. data on the Web, computer networks, ontologies, physical models, neural networks, etc. Many graph models have edges of a dynamic nature and these can be represented using streaming graphs.

The outlined recent applications of the problem where large, rapidly changing graphs are involved rekindled the interest in the graph reachability problem. The solutions proposed for the problem clearly show the trade-off between time and space requirements: 1)Additional information about the graph (i.e the transitive closure) needs to be stored and maintained for fast query answering, 2)The query time becomes linearly proportional to the size of the graph if no additional information is kept. In this paper, we will review various graph reachability algorithms, comment on their applicability to the streaming setting, and outline a new algorithm designed for rapidly changing graphs.

The outline of the paper is as follows: in Section 2 we will review the current graph reachability algorithms and focus on how they can be adapted to the streaming setting. In Section 3 we will outline a new algorithm for streaming graph reachability queries. Finally we will present our conclusions and outline the future work in Section 4.

2 Overview of the Existing Algorithms

Given a graph $G = (V, E)$ where V is the set of vertices, E is the set of edges and $|V| = n$, $|E| = m$, there are two naive approaches for answering reachability queries. One is to use the shortest path algorithm with $O(m)$ query time. Another naive approach to this problem is to pre-compute reachability between every pair of vertices in a graph so that reachability queries over this graph can be answered in constant time and require $O(n^2)$ space. As can be seen from their time and space requirements, these approaches are impractical for large graphs, even if they are static. If we consider the streaming setting we also need to consider real-time updates to the graph and the ability to continuously evaluate a reachability query. The query time of the first approach and update time of the second approach clearly show the infeasibility of their use for streaming graphs.

Efficient solutions to this problem on large sparse graphs involve reachability labeling methods. Several approaches have been proposed to encode graph reachability information using labeling schemes [1, 3, 5, 8, 6]. A labeling scheme assigns labels to vertices of the graph and answers a reachability query over two vertices by comparing the labels of the vertices. Interval-based labeling is used for tree structures that can answer reachability queries in constant time. However, the time complexity of this method is $O(m)$ for graphs. Cohen et al [3] proposed a 2-hop labeling scheme which uses $O(nm^{1/2})$ storage and $O(m^{1/2})$ time. Indexing (labeling) time for this method is $O(n^4)$ which is then reduced to $O(n^3)$ by the HOPI algorithm proposed by Schenkel et. Al [5, 8]. As it can be seen from the complexity results it is challenging to adapt these algorithms for streaming huge graphs. The space requirement of the Interval algorithm is $O(n^2)$ and the index time for HOPI is $O(n^3)$ which are quite high for huge graphs especially if they are rapidly changing.

The last method is called dual labeling by Wang et al. [6], which represents a graph using two components: a spanning tree and a set of t non-tree edges. For sparse, tree-like graphs, it is assumed that $t << n$. The two components together

	Query Time	Index Time	Index Size
Shortest Path	$O(m)$	0	0
Transitive Closure	$O(1)$	$O(n^3)$	$O(n^2)$
Interval	$O(n)$	$O(n)$	$O(n^2)$
2-Hop	$O(m^{1/2})$	$O(n^4)$	$O(nm^{1/2})$
HOPI	$O(m^{1/2})$	$O(n^3)$	$O(nm^{1/2})$
Dual-I	$O(1)$	$O(n+m+t^3)$	$O(n+t^2)$
Dual-II	$O(logt)$	$O(n+m+t^3)$	$O(n+t^2)$
Fluid Path	$O(t)$	$O(n+m+t)$	$O(n+t)$

Fig. 1. Complexity Comparison

contain the complete information needed to answer a reachability query over the original graph. The dual labeling method integrates interval-based labeling, which encodes reachability in the spanning tree and non-tree labeling to complete the reachability information of the graph. This method consists of two schemes Dual-I and Dual-II. The Dual-I scheme has constant query time, whereas it is $O(logt)$ for Dual-II. Both schemes have $O(n + t^2)$ space complexity, however Dual-II uses less space in practice. These algorithms are more promising for the streaming graph reachability problem, especially Dual-I with its constant query time. However the dynamically changing nature of the graph will impose further requirements on the efficiency of storing this index structure composed of two components. For each update we need to regenerate the index which is very costly for huge streaming graphs. Figure 1 summarizes the complexity results for the methods mentioned in this section and in the next section.

3 An Algorithm for Reachability on Streaming Graphs

In this section, we outline an algorithm for answering reachability queries on rapidly changing graphs which we call 'fluid path'. Our algorithm uses interval-based labeling and is comparable to dual labeling in terms of the trade off between time and space complexities which depend on the size of the set of non tree edges in the graph.

The input is a directed graph $G = (V, E)$ where $|V| = n$, $|E| = m$. We assume that G is acyclic and if not it can be transformed to an equivalent acyclic one in terms of reachability information in $O(n + m)$ time [9]. The next step is to find a spanning tree in the graph and and assign interval-based labels to all the nodes. The non-tree edges must be detected and added to a set T where $|T| = t$. Then we convert the directed graph to a tree with size $O(n + t)$ using this information, where non-tree edges are converted to tree edges by duplicating the target node as shown in Figure 2. The interval based label of the duplicate nodes are inherited from the parent of the duplicate node and the original node. Hence if there is a non-tree edge (u, v) and v is duplicated to $(v, 1)$ and $(v, 2)$, where there is an edge $(u, (v, 2))$ in the tree transformation, then the label of the

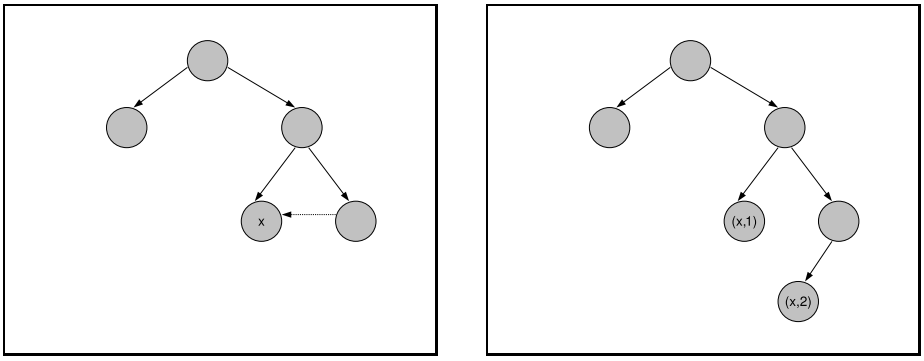

Fig. 2. Input Graph and its tree transformation

node $(v, 2)$ is (l_1, l_2), where l_1 is the label of u and l_2 is the label of v. All the remaining nodes are also assigned a timestamp which is 1, the initial timestamp.

In the streaming setting, we assume that the graph information is received as this tree transformation since time-based labels can be added to the nodes as they are received, so a node is a pair (n_1, t_1) in this case where t_1 is the timestamp assigned to the node.

The last step of the algorithm is to check whether a node v is reachable from u in the original graph using the labeling information. For this we need to check whether $(v, 1)$ is reachable from $(u, 1)$ in the tree transformation. Assume that the label of $(v, 1)$ is l_1 and the label of $(u, 1)$ is l_2 then there are two cases. First if the interval represented by l_1 is in the interval represented by l_2 then v is reachable from u and this step takes constant time. Second if the interval represented by l_1 is not in the interval represented by l_2 then we also need to check the other copies of v and determine if a copy of v is reachable from a duplicated node that is reachable from $(u, 1)$. For instance if the label of $(v, 2)$ is $l_3 = (l_4, l_1)$ to determine if $(v, 2)$ is reachable from $(u, 1)$ we check if at least one of the intervals represented by l_4 or l_1 is in the interval represented by l_2 and return 'reachable' if so, otherwise we check all the duplicated nodes reachable from $(u, 1)$ and determine whether a copy of v is reachable from them. Since the size of the set of the duplicated nodes is $O(t)$, the time complexity of this step is $O(t)$.

4 Conclusions and Future Work

Graph reachability is a well studied problem which has applications in many fields. The problem attracted more attention with the huge increase in data where graph structures play an important role in representing the connections and the dynamic nature of it. In this paper, we reviewed the various graph reachability algorithms and their applicability for rapidly changing graphs. We also outlined an algorithm designed exclusively for these types of graphs.

As future work we plan to extend our survey on the literature, provide a detailed algorithm and analyze how it performs in real applications that involve

rapidly changing graphs. We also plan to propose methods for a reasoner component that uses different (and possibly hybrid) reachability algorithms depending on the structure of the input graph and time/space requirements.

References

1. Agrawal, R., Borgida, A., Jagadish, H.V.: Efficient management of transitive relationships in large data and knowledge bases. SIGMOD Rec. **18**(2) (1989) 253–262
2. Abiteboul, S., Kaplan, H., Milo, T.: Compact labeling schemes for ancestor queries. In: SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics (2001) 547–556
3. Cohen, E., Halperin, E., Kaplan, H., Zwick, U.: Reachability and distance queries via 2-hop labels. SIAM J. Comput. **32**(5) (2003) 1338–1355
4. Roditty, L., Zwick, U.: A fully dynamic reachability algorithm for directed graphs with an almost linear update time. In: STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, New York, NY, USA, ACM (2004) 184–191
5. Schenkel, R., Theobald, A., Weikum, G.: HOPI: An efficient connection index for complex XML document collections. In Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K., Ferrari, E., eds.: Advances in database technology, EDBT 2004 : 9th International Conference on Extending Database Technology. Volume 2992 of Lecture Notes in Computer Science., Heraklion, Crete, Greece, Springer (2004) 237–255 Acceptance ratio 1:7.
6. Wang, H., He, H., Yang, J., Yu, P.S., Yu, J.X.: Dual labeling: answering graph reachability queries in constant time. In: in Proc. 22nd International Conference on Data Engineering, IEEE Computer Society (2006) 75
7. Kindberg, T., Chalmers, M., Paulos, E.: Guest editors' introduction: Urban computing. IEEE Pervasive Computing **6**(3) (2007) 18–20
8. Schenkel, R., Theobald, A., Weikum, G.: Efficient creation and incremental maintenance of the hopi index for complex xml document collections. In: ICDE '05: Proceedings of the 21st International Conference on Data Engineering, Washington, DC, USA, IEEE Computer Society (2005) 360–371
9. Paige, R., Tarjan, R.E.: Three partition refinement algorithms. SIAM J. Comput. **16**(6) (1987) 973–989