ISSN 2278 – 0211 (Online)

Dynamic Transitive Closure Problems on Directed Graphs

Dasari Vemkata Lakshmi
Department of Mathematics, Bapatla Women's Engineering College Bapatla, Guntur District, A. P., India
Marella Sunitha Bharathi
Department of Mathematics, Vasireddy Venkatadri Institute of Technology, Nambur, Guntur District, A. P., India
G. Srinivasu
Department of Mathematics, R.S.R. Engineering College, Kadanuthala, SPSR Nellore District, A. P., India

Abstract:
In this paper we survey the newest results for dynamic problems on directed graphs. In particular, we focus on the most fundamental problem, transitive closure. These problems play a crucial role in many applications, including network optimization and routing, traffic information systems, data bases, compilers.

Keywords: *Transitive closure, directed graphs*

1. Introduction
We first present general techniques and tools used in designing dynamic path problems on directed graphs and then we address the newest results for dynamic transitive closures. Here the goal is to maintain reach ability information in a directed graph subject to insertions and deletions of edges. The fastest known algorithms support graph updates in quadratic or near-quadratic time and reach ability queries in constant time.

2. General Techniques for Directed graphs
In this subsection we discuss the main techniques used to solve dynamic path problems on directed graphs. We first address combinatorial and algebraic properties and then we consider some efficient data structures, which are used as building blocks in designing dynamic algorithms for transitive closure and shortest paths.

2.1. Path Problems and Kleene Closures
Path problems such as transitive closure and shortest paths are tightly related to matrix sum and matrix multiplication over a closed semi ring [2].
NOTATION: The usual sum and multiplication operations over Boolean matrices are denoted by + and • respectively.
NOTATION: Given two real-valued matrices A and B, C = A ® B is the matrix product such that C[x, y] = $\min_{1<z<n}\{A[x, z] + B[z, y]\}$ and D = A+B is the matrix sum such that D[x, y] = min {A[x, y], B[x, y]}.
NOTATION: We also denote by AB the product A ® B and by AB[x y] ,entry (x, y) of matrix AB.

2.2. Inference
- Let G = (V, E) be a directed graph and let TC (G) be the (reflexive) transitive closure of G. If X is the Boolean adjacency matrix of G, then the Boolean adjacency matrix of TC(G) is the Kleene closure of X on the {+, •, 0, 1} Boolean semi ring :
$$X^* = \sum_{i=0}^{n-1} X_i$$
- Let G = (V, E) be a weighted directed graph with no negative-length cycles. If X is a weight matrix such that X [x, y] is the weight of edge (x, y) in G, then the distance matrix of G is the Kleene closure of X on the {+, ®, R} semi ring:
$$X^* = \bigoplus_{i=0}^{n-1} X_i$$
- The next two facts recall two well-known methods for computing the Kleene closure X* of an n x n matrix X.
- Logarithmic Decomposition. A simple method to compute X*, based on repeated squaring, requires O (n^{μ}. log n) worst-case time, where O(n^{μ}) is the time required for computing the product of two matrices over a closed semi ring.

- This method performs $\log_e n$ sums and products of the form $X_i = X_i + X_z$ where $X = X_0$ and $X^* = X_i \log n^2$
- Recursive Decomposition. Another method, due to Munro [7], is based on a Divide and Conquer strategy and computes X* in $O(n^{\mu})$ worst-case time.
- Munro observed that, if we partition a matrix X into four sub matrices A, B, D, C of size n/2 x n/2 ,considered in clockwise order, and the closure X* similarly into four sub matrices E, F, H, G of size n/2 x n/2, then X* is definable recursively according to the following equations:
  E= (A+BD*C)*= EBD*= D*CE= D* + D*CEBD*
- Surprisingly, using this decomposition the cost of computing X* starting from X is asymptotically the same as the cost of multiplying two matrices over a closed semi ring.

2.3. Reach Ability Trees
A special tree data structure has been widely used to solve dynamic path problems on directed graphs. The first appearance of this tool dates back to 1981, when Even and Shiloach showed how to maintain a breadth-first tree of an undirected graph under any sequence of edge deletions [4], they used this as a kernel for decremental connectivity on undirected graphs. Later on Henzinger and King [5] showed how to adapt this data structure to fully dynamic transitive closure in directed graphs.

2.4. Problem
In the unweighed directed version, the goal is to maintain information about breadth-first search on a directed graph G undergoing deletions of edges. In particular, in the context of dynamic path problems, we are interested in maintaining BFS trees of depth up to d, with $d < n$. Given a directed graph G = (V, E) and a vertex $r \in V$, we would like to support any intermixed sequence of the following operations:
Delete(x, y): delete edge (x, y) from G.
Level (u): return the level of vertex a in the BFS tree of depth d rooted at r (return +oo if a is not reachable from r within distance d).

2.5. Inference
- King Maintaining BFS levels up to depth d from a given root requires O(md) time in the worst case throughout any sequence of edge deletions in a directed graph with m initial edges.

2.6. Observation
- This means that maintaining BFS levels requires d times the time needed for constructing them. Since $d < n$, we obtain a total bound of O(mn) if there are no limits on the depth of the BFS levels.
- As it was shown in it is possible to extend the BFS data structure presented in this section to deal with weighted directed graphs. In this case, a shortest path tree is maintained in place of BFS levels, after each edge deletion or edge weight increase, the tree is reconnected by essentially mimicking Dijkstra's algorithm rather than BFS.

3. Dynamic Transitive Closure
In this subsection we survey the best known algorithms for fully dynamic transitive closure. Given a directed graph G with n vertices and m edges, the problem consists of supporting any intermixed sequence of operations of the following kind.
Insert (u, v): insert edge (u, v) in G;
Delete (u, v): delete edge (u, v) from G;
Query(x, y): answer a reach ability query by returning "yes" if there is a path from vertex x to vertex y in G, and "no" otherwise;

3.1. Inference
- A simple-minded solution to this problem consists of maintaining the graph under insertions and deletions, searching if y is reachable from x at any query operation. This yields O(1) time per update (Insert and Delete), and O(m) time per query, where m is the current number of edges in the maintained graph.
- Another simple-minded solution would be to maintain the Kleene closure of the adjacency matrix of the graph, rebuilding it from scratch after each update operation. Using the recursive decomposition of Munro [7] discussed in Path Problems and Kleene Closures and fast matrix multiplication [8], this takes constant time per reach ability query and $O(n^w)$ time per update, where $w < 2.38$ is the current best exponent for matrix multiplication.

3.2. Observation
- Despite many years of research in this topic, no better solution to this problem was known until 1995, when Henzinger and King [5] proposed a randomized Monte Carlo algorithm with one-sided error supporting a query time of $O(n/\log n)$ and an amortized update time of $O(nm^0\ 58 \log^2 n)$, where m is the average number of edges in the graph throughout the whole update sequence. Since m can be as high as $O(n^2)$, their update time is $O(n^{216} \log^2 n)$.

- Khanna, Motwani and Wilson [12] proved that, when a look ahead of O (n^0 18) in the updates is permitted, a deterministic update bound of o(n') can be achieved.
- King and Sagert showed how to support queries in O(1) time and updates in O(n^{226}) times for general directed graphs and O(n^2) time for directed acyclic graphs; their algorithm is randomized with one-sided error. These bounds were further improved by King [6], who exhibited a deterministic algorithm on general digraphs with O(1) query time and O(n^2logn) amortized time per update operations, where updates are insertions of a set of edges incident to the same vertex and deletions of an arbitrary subset of edges.
- Using a different framework, Demetrescu and Italiano [10] obtained a deterministic fully dynamic algorithm that achieves O(n^2) amortized time per update for general directed graphs.
- We note that each update might change a portion of the transitive closure as large as O (n^2). Thus, if the transitive closure has to be maintained explicitly after each update so that queries can be answered with one lookup, O (n^2) is the best update bound one could hope for.
- If one is willing to pay more for queries, Demetrescu and Italiano [8] showed how to break the O (n^2) barrier on the single-operation complexity of fully dynamic transitive closure, building on a previous path counting technique introduced by King and Sagert [11], they devised a randomized algorithm with one-sided error for directed acyclic graphs that achieves O(n^1 58) worst-case time per update and O(n^{058}) worst-case time per query.
- Other recent results for dynamic transitive closure appear in [7].

3.3. King's O (n^2 log n) Update Algorithm
King [6] devised the first deterministic near-quadratic update algorithm for fully dynamic transitive closure. The algorithm is based on the tree data structure considered in Reachability Trees and on the logarithmic decomposition discussed in Path Problems and Kleene Closures. It maintains explicitly the transitive closure of a graph G in O (n^2 log n) amortized time per update, and supports inserting and deleting several edges of the graph with just one operation. Insertion of a bunch of edges incident to a vertex and deletion of any subset of edges in the graph require asymptotically the same time of inserting or deleting just one edge.

3.4. Approach
The algorithm maintains log n + 1 levels. level i, 0 < i < log n, maintains a graph GZ whose edges represent paths of length up to 2' in the original graph G. Thus, Go = G and $G_{\log n}$ is the transitive closure of G.

3.5. Inference
- Each level i is built on top of the previous level i - l by keeping two trees of depth < 2 rooted at each vertex v of G: an out-tree OUT_Z (v) maintaining vertices reachable from v by traversing at most two edges in G_{Z_1}, and an in-tree IN_Z(v) maintaining vertices that reach v by traversing at most two edges in G_{Z_1}. An edge (x, y) will be in Gz if and only if x ∈ INz (v) and y E OUT_T (v) for some v.
- The 2 log n trees INz (v) and OUT_T (v) are maintained with instances of the BFS tree data structure considered in Reachability Trees.
- To update the levels after an insertion of edges around a vertex v in G, the algorithm simply rebuilds INz(v) and OUT_T (v) for each i, 1 < i < log n, while other trees are not touched. This means that some trees might not be up to date after an insertion operation. Nevertheless, any path in G is represented in at least the in/out trees rooted at the latest updated vertex in the path, so the reachability information is correctly maintained. This idea is the key ingredient of King's algorithm.
- When an edge is deleted from GZ, it is also deleted from any data structures INz(v) and OUT_T (v) that contain it [6].

4. Demetrescu and Italiano's O(n^2) Update Algorithm
The algorithm by Demetrescu and Italiano [8] is based on the matrix data structure considered in Matrix Data Structures and on the recursive decomposition discussed in Kleene Closures. It maintains explicitly the transitive closure of a graph in O(n^2) amortized time per update, supporting the same generalized update operations of King's algorithm, i.e., insertion of a bunch of edges incident to a vertex and deletion of any subset of edges in the graph with just one operation. This is the best known update bound for fully dynamic transitive closure with constant query time.

4.1. Approach
The algorithm maintains the Kleene closure X* of the n x n adjacency matrix X of the graph as the sum of two matrices X_1 and X_2.

4.2. Notation
Let V_1 be the subset of vertices of the graph corresponding to the first half of indices of X, and let V_2 contain the remaining vertices.

4.3. Inference

- Both matrices X_I and X_2 are defined according to Munro's equations of Path Problems and Kleene Closures, but in such a way that paths appearing due to an insertion of edges around a vertex in V_1 are correctly recorded in X_1, while paths that appear due to an insertion of edges around a vertex in V_2 are correctly recorded in X_2.Thus, neither X_1 nor X_2 encode complete information about X*, but their sum does.
- In more detail, assuming that X is decomposed in sub-matrices A, B, C, Das explained in Path Problems and Kleene Closures, and that X_1, and X_2 are similarly decomposed in sub-matrices E_1, F_1, G_1, H_1 and E_2, F_2, G_2, H_2, the algorithm maintains X_1 and X_2 with the following 8 polynomials using the data structure discussed in Matrix Data Structures

$$Q=A+BP^2C \qquad E_2=E_1BH_2^2CE_1$$
$$F_1=E_1^2BP \qquad F_2=E_1BH_2^2$$
$$G_1=PCE_1^2 \qquad G_2=H_2^2CE_1$$
$$H_1=PCE_1^2BP \qquad R=D+CE_1^2B$$

where $P=D^*$, $E_1=Q^*$, and $H_2=R^*$ are Kleene closures maintained recursively as smaller instances of the problem of size $\frac{n}{2}\times\frac{n}{2}$.

- To support an insertion of edges around a vertex in V_1, strict updates are per-formed on polynomials Q, F_1, G_1, and H_1 using SetRow and SetCol, while E_2, F_2, G_2, and R are updated with LazySet**.**
- Insertions around V_2 are performed symmetrically, while deletions are supported via Reset operations on each polynomial in the recursive decomposition
- Finally, P, E_1, and H_2 are updated recursively. The low-level details of the method appear in [10].

5. References

1. C. Demetrescu and G.F. Italiano, A new approach to dynamic all pairs shortest paths, Proc. 35th Symp. on Theory of Computing (STOC'03), San Diego, CA (2003), 159-166.
2. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, Second Edition, MIT Press, 2001.
3. D. H. Greene and D.E. Knuth, Mathematics for the analysis of algorithms, Birkhauser, 1982.
4. S. Even and Y. Shiloach, An on-line edge deletion problem, J. Assoc. Comput. Mach. 28 (1981), 1-4.
5. M. R. Henzinger and V. King, Randomized fully dynamic graph algorithms with polylogarithmic time per operation, J. Assoc. Comput. Mach. 46(4) (1999), 502-536.
6. V. King, Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs, Proc. 40-th Symposium on Foundations of Computer Science (FOCS 99) (1999).
7. I. Munro, Efficient determination of the transitive closure of a directed graph, Information Processing Letters 1(2) (1971), 56-58.
8. D. Coppersmith and S. Winograd, Matrix multiplication via arithmetic progressions, J. of Symbolic Computation 9 (1990).
9. C. Demetrescu and G. F. Italiano, Fully dynamic all pairs shortest paths with real edge weights, Proc. of the 42nd IEEE Annual Symposium on Foundations of Computer Science (FOCS'01), Las Vegas, Nevada (2001), 260-267.
10. C. Demetrescu and G. F. Italiano, Fully dynamic transitive closure: Breaking through the $0(n^2)$ barrier, Proc. of the 41st IEEE Annual Symposium on Foundations of Computer Science (FOCS'00) (2000), 381-389.
11. G. Ausiello, G. F. Italiano, A. Marchetti-Spaccamela, and U. Nanni, Incremental algorithms for minimal length paths, J. of Algorithms 12(4) (1991), 615-638.
12. S. Khanna, R. Motwani, and R. H. Wilson, On certificates and lookahead on dynamic graph problems, Algorithmica 21(4) (1998), 377-394.