INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH & DEVELOPMENT

ISSN: 2278 – 0211 (Online)

Dynamic Problems On Undirected Graphs

G.Srinivasu
Dept. of Mathematics, PBR VITS KAVALI, SPSR Nellore Dist., A.P., INDIA

D.Ramalinga Reddy
Dept. of Mathematics,Prakasam engineering college,kandukur,Prakasam Dist., A.P., INDIA

Abstract:

In many applications of graph algorithms, including communication networks, VLSI design, graphics, and assembly planning, graphs are subject to discrete changes, such as additions or deletions of edges or vertices. In the last two decades there has been a growing interest in such dynamically changing graphs, and a whole body of algorithms and data structures for dynamic graphs has been discovered. This paper presents two different data structures that maintain properties of dynamically changing trees: topology trees, ET trees. Next, in the course of the presentation, it was also highlighted how these techniques have been applied to solve the fully dynamic connectivity and/or minimum spanning tree problems, and which update and query bounds can be achieved when they are deployed.

1.Definitions

- An update on a graph is an operation that inserts or deletes edges or vertices of the graph or changes attributes associated with edges or vertices, such as cost or color.
- A dynamic graph is a graph that undergoes a sequence of updates.

2.Observation

- In a typical dynamic graph problem, one would like to answer queries on dynamic graphs, for instance, whether the graph is connected, or which is the shortest path between any two vertices.
- The goal of a dynamic graph algorithm is to update efficiently the solution of a problem after dynamic changes, rather than having to re compute it from scratch each time. Given their powerful versatility, it is not surprising that dynamic algorithms and dynamic data structures are often more difficult to design and to analyze than their static counterparts.

3.Definitions

We can classify dynamic graph problems according to the types of updates allowed.

- A dynamic graph problem is said to be fully dynamic if the update operations include unrestricted insertions and deletions of edges or vertices.
- A dynamic graph problem is said to be partially dynamic if only one type of update, either insertions or deletions, is allowed.
- A dynamic graph problem is said to be incremental if only insertions are allowed.
- A dynamic graph problem is said to be decremental if only deletions are allowed.

4.Observation

- This paper examines the dynamic problems on *undirected* graphs. To illustrate those techniques, we focus particularly on dynamic minimum spanning trees and on connectivity problems.

4.1.Dynamic Problems On Undirected Graphs

This part considers fully on dynamic algorithms for undirected graphs. These algorithms maintain efficiently some property of a graph that undergoes structural changes defined by insertion and deletion of edges, and/or updates of edge costs. To check the graph property throughout a sequence of these updates, the algorithms must be prepared to answer queries on the graph property efficiently.

5.Examples

- The fully dynamic minimum spanning tree problem consists of maintaining a minimum spanning forest of a graph during insertions of edges, deletions of edges, and edge cost changes.
- A fully dynamic connectivity algorithm must be able to insert edges, delete edges, and answer a query on whether the graph is connected, or whether two vertices are in the same connected component.

6.Observation

The goal of a dynamic algorithm is to minimize the amount of re computation required after each update.

All the dynamic algorithms described are able to maintain dynamically the graph property at a cost (per update operation) which is significantly smaller than the cost of re computing the graph property from scratch.

This part, presents general techniques and tools used in designing dynamic algorithms on undirected graphs, and then survey the fastest algorithms for solving two of the most fundamental graph problems: connectivity and minimum spanning trees. Many of the algorithms proposed in the literature use the same general techniques, and hence begin by describing these techniques. As a common theme, most of these techniques use some sort of graph decomposition, and they partition either the vertices or the edges of the graph to be maintained. Moreover, data structures that maintain properties of dynamically changing trees are often used as building blocks by many dynamic graph algorithms. The basic update operations are edge insertions and edge deletions. Many properties of dynamically changing trees have been considered in the literature.

7.Examples

- The basic query operation is tree membership: while the forest of trees is dynamically changing, the researcher likes to know at any time which tree contains a given vertex, or whether two vertices are in the same tree. Dynamic tree membership is a special case of dynamic connectivity in undirected graphs, and indeed in 1.2 and in 1.3 it is observed some of the data structures presented here for trees are useful for solving the more general problem on graphs.
- Other properties have also been considered: the parent of a vertex, the least common ancestor of two vertices, and the center or the diameter of a tree When costs are associated either to vertices or to edges, one could also ask what is the minimum or maximum cost in a given path.

8.Topology Trees

Topology trees have been introduced to maintain dynamic trees upon insertions and deletions of edges.

9.Definitions

- Given a tree T of a forest, a cluster is a connected sub graph of T.
- The cardinality of a cluster is the number of its vertices.
- The external degree of a cluster is the number of tree edges incident to it.
- A topology tree is a hierarchical representation of a tree T of the forest: each level of the topology tree partitions the vertices of T into clusters. Clusters at level 0 contain one vertex each. Clusters at level £' > 1 form a partition of the vertices of the tree T' obtained by shrinking each cluster at level £' - 1 into a single vertex. The basic partition must be suitably chosen so that the topology tree has depth O (log n) and, so that during edge insertions and deletions, each level of the topology tree can be updated by applying only a few local adjustments.

10.Assumption

In order to illustrate the solution proposed by Frederickson [7, 8], let us assume that the tree T has maximum vertex degree 3: this is without loss of generality, since a standard transformation can be applied if this is not the case [9].

11.Definition

- A restricted partition of a tree T is a partition of its vertex set V into clusters of external degree < 3 and cardinality < 2 such that:
  Each cluster of external degree 3 has cardinality 1.
  Each cluster of external degree < 3 has cardinality at most 2.
  No two adjacent clusters can be combined and still satisfy the above.
- An example of topology tree, together with the restricted partitions used to obtain its levels, is given in Figure1.

12.Observation

- There can be several restricted partitions for a given tree T, based upon different choices of the vertices to be unioned.
- From the above definition it is observed, the restricted partition implements a cluster-forming scheme according to a locally greedy heuristic, which does not always obtain the minimum number of clusters, but which has the advantage of requiring only local adjustments during updates.

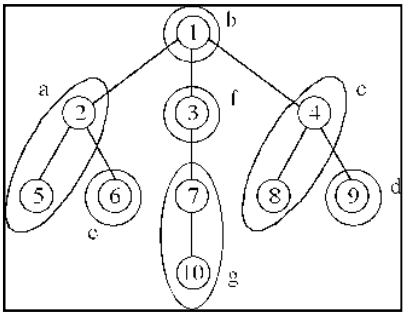
13.Approach

13.1.Edge Deletion

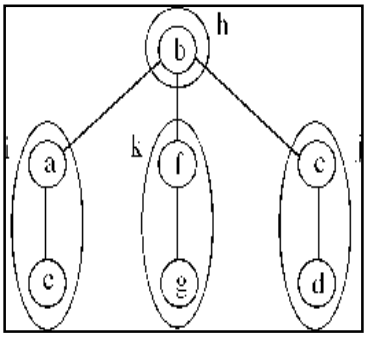
let us sketch how to update the clusters of a restricted partition when an edge e is deleted from a tree T. First, removing e splits T into two trees, say T_1 and T_2, which inherit all of the clusters of T, possibly with the following exceptions.

- If e is entirely contained in a cluster, this cluster is no longer connected and therefore must be split. After the split, we must check whether each of the two resulting clusters is adjacent to a cluster of tree degree at most 2, and if these two adjacent clusters together have cardinality < 2. If so, we combine these two clusters in order to maintain condition (3).

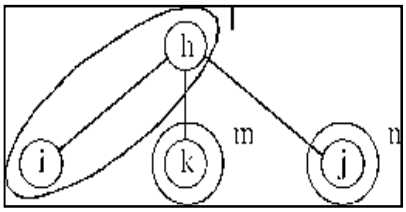
- If e is between two clusters, then no split is needed. However, since the tree degree of the clusters containing the endpoints of e has been decreased, we must check if each cluster should be combined with an adjacent cluster, again because of condition (3).



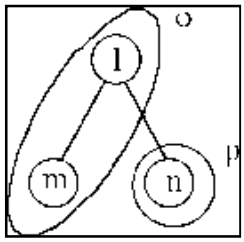
Tree T and clusters of level 1



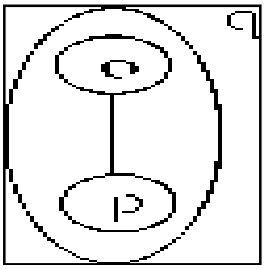
Clusters of level 2



Clusters of level 3



Clusters of level 4



Cluster of level 5

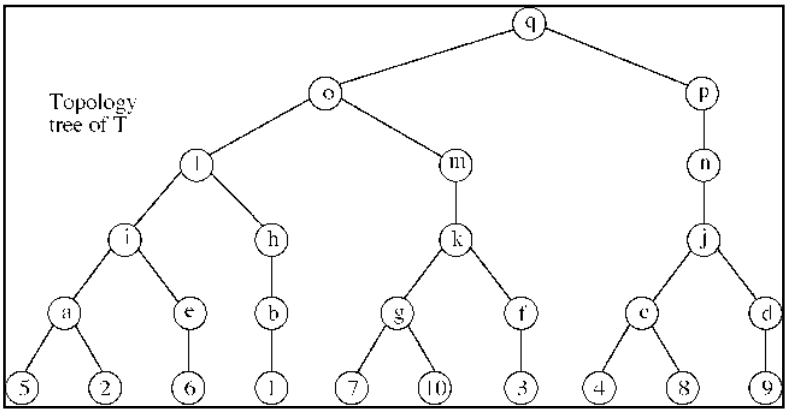


Figure 1: Restricted partitions and topology tree of a tree T

13.2.Edge Insertion

Similar local manipulations can be applied to restore invariants (1) - (3) of the definition of restricted partition, in case of edge insertions.

13.3.Construction Of The Topology Tree

The levels of the topology tree are built in a bottom up fashion by repeatedly applying the locally greedy heuristic.

13.4.Update Of The Topology Tree

Each level can be updated upon insertions and deletions of edges in tree T by applying few locally greedy adjustments similar to the ones described before. In particular, a constant number of basic clusters are examined: the changes in these basic clusters percolate up in the topology tree, possibly causing vertex clusters to be regrouped in different ways.

14.Inference

- The number of nodes at each level of the topology tree is a constant fraction of that at the previous level, and thus the number of levels is O(log n)
- The fact that only a constant amount of work has to be done on O (log n) topology tree nodes implies a logarithmic bound on the update time.
- The update of a topology tree because of an edge insertion or deletion can be supported in O (log n) time.

15.ET Trees

Euler Tour trees have been introduced by Henzinger and King [11] to work on dynamic forests whose vertices are associated with weighted or unweighted keys. Updates allow it to cut arbitrary edges, to insert edges linking different trees of the forest, and to add or remove the weighted key associated to a vertex. Supported queries are the following:

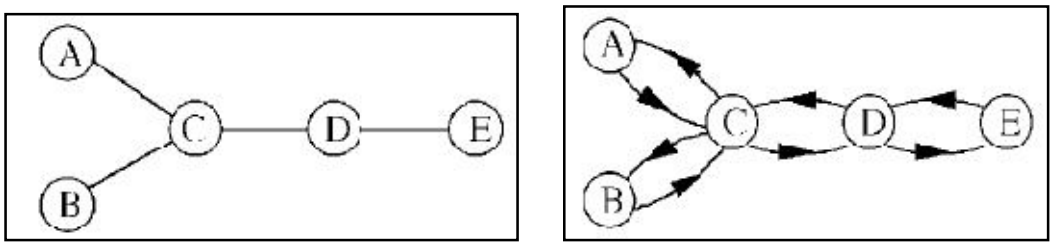
- Connected (u, v)*:* tells whether vertices u and *v* are in the same tree.
- Size (v): returns the number of vertices in the tree that contains v.
- Minkey(v): returns a key of minimum weight in the tree that contains v; if keys are unweighted, an arbitrary key is returned.

16.Definitions

- An Euler tour of a tree *T* is a maximal closed walk over the graph obtained by replacing each edge of T by two directed edges with opposite

direction. The walk traverses each edge exactly once; hence, if T has n vertices, the Euler tour has length (2n – 2) [Fig-2].

- An ET tree is a dynamic balanced binary tree (the number of nodes in the left and right sub trees of each node differs by at most one) over some Euler tour around T. Namely; leaves of the balanced binary tree are the nodes of the Euler Tour, in the same order in which they appear (Fig. 2).



Tree T Replacing edges of '1' with directed edges

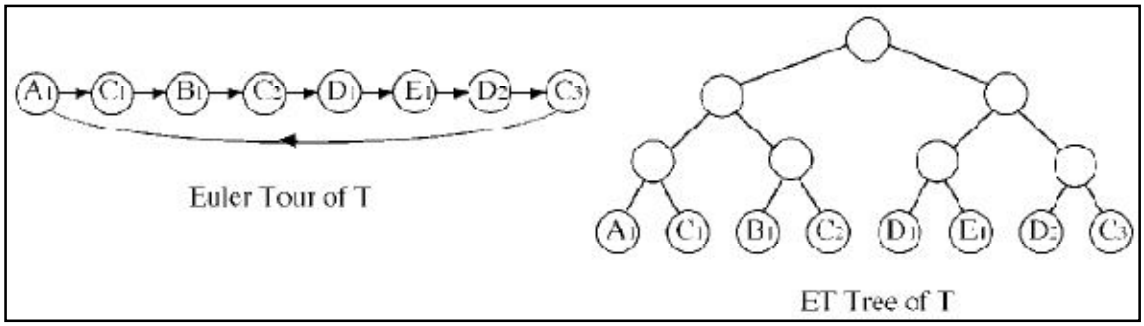


Figure 2: Dynamic balanced binary tree

17.Observation

Although each vertex of T may occur several times in the Euler tour (an arbitrary occurrence is marked as *representative* of the vertex), an ET tree has *0(n)* nodes.

18.Approach

18.1.Edge Insertion And Deletion

If trees in the forest are linked or cut, a constant number of splits and concatenations allow it to reconstruct the new Euler tour(s); the ET tree(s) can then be rebalanced by affecting only 0(log *n)* nodes.

18.2.Connectivity Queries

The query Connected (u, v) can be easily supported in 0(log *n)* time by finding the roots of the ET trees containing u and v and checking if they coincide.

18.3.Size And Minkey Queries

To support Size and Minkey queries, each node q of the ET tree maintains two additional values, the number s (q) of representatives below it and the minimum weight key k(q) attached to a representative below it. Such values can be maintained in 0(log n) time per update and allow it to answer queries of the form Size(v) and Minkey(v) in 0(log n) time for any vertex v of the forest: the root r of the ET tree containing v is found and values s(r) and k(r) are returned, respectively. See [11] for additional details of the method.

19.Inference

Both updates and queries can be supported in 0(log *n)* time using ET trees.

20.Reference

1. S. Alstrup, J. Holm, K. de Lichtenberg, and M. Thorup, Minimizing diameters of dynamic trees, Proc. 24th Int. Colloquium on Automata, Languages and Programming (ICALP 97) (1997), LNCS 1256, 270-280.
2. S. Alstrup, J. Holm, and M. Thorup, Maintaining center and median in dynamic trees, Proc. 7th Scandinavian Workshop on Algorithm Theory (SWAT 00) (2000), 46-56.
3. 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.
4. 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 Mathematical computing (FOCS'00) (2000), 381-389.
5. 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.
6. D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig, Sparsification - A technique for speeding up dynamic graph algorithms, J. Assoc. Comput. Mach., 44 (1997), 669-696.
7. G. N. Frederickson, Data structures for on-line updating of minimum spanning trees, SIAM J. Comput. 14 (1985), 781-798.
8. G. N. Frederickson, Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees, SIAM J. Comput. 26(2) (1997), 484-538.
9. F. Harary, Graph Theory, Addison-Wesley, 1969.
10. M. R. Henzinger and V. King, Maintaining minimum spanning trees in dy-namic graphs, Proc. 24th Int. Colloquium on Automata, Languages and Programming (ICALP 97) (1997) 594-604.
11. 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.
12. V. King, Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs, Proc. 40-th Symposium on Foundations of Mathematical computing (FOCS 99) (1999).