A Distributed Fully Dynamic Algorithm for Maintaining a Minimal Feedback Vertex Set in a Graph

Fouad Tigrine, *Laboratoire LE2I. Université de Bourgogne, Faculté Mirande, 21000 Dijon, France*
Hamamache Kheddouci, *Laboratoire LIESP. Université Lyon 1 - Bat Nautibus, 69622 Villeurbanne Cedex France*

Abstract

In the *feedback vertex set* problem, the aim is to minimize, in a connected graph $G = (V, E)$, the cardinality of the set $S \subset V(G)$, whose removal induces an acyclic subgraph. In this paper, we propose a distributed fully dynamic algorithm for maintaining a minimal feedback vertex set in $O(\Delta(n - \Delta))$ time, in an undirected connected graph of n vertices and maximum degree Δ.

Keywords: feedback vertex set, dynamic graph, distributed dynamic algorithm.

1. Introduction

By a graph is meant a finite, simple and undirected graph. A feedback vertex set of a graph $G(V, E)$ is a subset of vertices $S \subseteq V$ whose removal from G induces an acyclic graph $G'(V', E')$ with $V' = V \setminus S$ and $E' = \{\{u, v\} \in E \mid u, v \in V'\}$. If the cardinality of S is the minimal possible, we call it a minimum feedback vertex set of G. The cardinality of S is denoted by $F(G)$. S is a minimal FVS of G if no proper subset of S is a FVS of G.

The problem of finding a minimum feedback vertex set of a graph has been proved to be NP-hard [18], but polynomial time solutions have been found for particular classes of graphs, e.g., reducible graphs [16], Kronecker product of graphs [2] and the circulant graphs [3]. For general graphs, the best known approximation algorithm has approximation ratio 2 [4].

Dynamic problems on graphs have been extensively studied. Several algorithms have been proposed for maintaining fundamental structural information about dynamic graphs, such as connectivity [15, 11], transitive closure [14, 9], shortest paths [8, 13, 10], and minimum spanning trees [17, 12]. Dynamic planar graphs aise in communication networks, graphics, and VLSI design, and they occur in algorithms that build planar subdivisions such as Voronoi diagrams.

A minimal feedback vertex set of a graph on n vertices can be trivially found in time $O(2^n n)$ by trying all possible vertex subsets. Many polynomial time algorithms were known to construct a feedback vertex set of graphs. Schwikowski and al. [5] propose an algorithm for constructing a minimal FVS of a graph $G(V, E)$ with a polynomial delay of $O(|E||V|(|E| + |V|)$. Very recently, Tigrine and Kheddouci [1] proposed a self-stabilizing algorithm for constructing a minimal FVS on $O(n^2)$.

In this paper we propose a distributed fully dynamic algorithm for maintaining a minimal FVS in a dynamic graph which runs in $O(\Delta(n - \Delta))$ time, with Δ and n are respectively the maximum degree and the order of the graph.

The rest of this paper is organized as follows: Section 2, gives the necessary definitions and notations to introduce our problem. Section 3 presents the proposed distributed fully dynamic algorithm for maintaining a minimal feedback vertex set of dynamic graphs. Finally, Section 4 brings our remark concluding the paper.

2. Notation and terminology

We define a distributed network of order n as a connected undirected graph $G = (V, E)$ with vertex set V ($|V| = n$) and edge set E. Two vertices joined by an edge are said to be *neighbors*. We use $N(i)$ to denote the set of neighbors of vertex i and denote by d_i the degree of vertex i in G. The maximum degree of G is denoted by Δ. $G[V']$ is the subgraph of G induced by the set V', such that $V' \subset V$. Let S be a minimal vertex set of G. Let F be the vertex set of the induced acyclic subgraph of G given by removing S from G. A dynamic graph is a graph in which edges or vertices are inserted or deleted. We consider *updates* ζ of the following form:

- **insertion (resp. deletion) of a vertex into (resp. from)** V: let v be a new vertex, with $v \notin V$ (resp. $v \in V$) and E_v be a edge set connecting v to other vertices such that $G' = (V \cup \{x\}, E \cup E_v)$ (resp. $G' = (V \setminus \{x\}, E \setminus E_v)$) be the new graph given by update ζ;
- **insertion (resp. deletion) of an edge** (u, v) **into (resp. from)** G: let u and v be two vertices in V and (u, v) be a new edge, with $(u, v) \notin E$ (resp. $(u, v) \in E$, such that $G' = (V, E \cup \{(u, v)\})$ (resp. $G = (V, E \setminus \{(u, v)\})$) be a new graph given by update ζ.

An ***update on a graph*** is an operation that inserts or deletes edges or vertices on the graph. An ***dynamic graph*** is a graph that is undergoing a sequence of updates. The goal of dynamic algorithm is to update efficiently the solution of a problem after dynamic changes. 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.

3. A distributed fully dynamic self-stabilizing algorithm

This section considers a distributed fully dynamic algorithm for undirected graphs. This algorithm maintains efficiently a minimal FVS of graphs. We first present the data structure that maintains a minimal FVS of graphs.

3..1 Data structure for maintaing a minimal FVS

A minimal FVS of $G(V, E)$ is a subset $S \subset V$ which verifies the following conditions :

- **C1**: by removing S from G we induce an acyclic subgraph $G[(V \setminus S) = F]$.
- **C2**: each vertex of S creates at least one cycle in $G[F]$.

We consider an acyclic graph $G[F]$ as a forest which contains a set of trees. Each tree is represented by a connected component W_i, such that $V(W_i) = \{j : i \leq j \leq n$ and there exists a path between j and i in $G[F]\}$. Each vertex i of W_c maintains the following data structure:

$$\begin{cases} i.e \in \{f, e\} & \text{: state of } i \text{ } (i.e = f \text{ if } i \in F \text{ and } i.e = s \text{ if } i \in S). \\ i.c \in \{0, 1, 2, ..., n\} & \text{: index of the connected component which contains } i. \\ & \text{If } i \in S \text{ then } i.c = 0; \\ i.s & \text{: array which stores the indexes of the connected} \\ & \text{components which contain its neighbors.} \end{cases}$$

The proposed dynamic algorithm is decomposed into four main procedures. Each one maintains a minimal FVS after each update operation.

3..2 Update operations

In this subsection, we describe some update operations which are the kernel of our distributed fully dynamic algorithm for maintaining a minimal FVS.

We start with some observations which are common to all the update operations considered here. First of all, we define the four update operations for dynamic graphs:

1. *ADDedge(i,j)* which adds edge (i, j) to G. The new graph G becomes $G' = (V, E \cup \{(i, j)\})$.
2. *MOVedge(i,j)*which deletes edge (i, j) from G. So G' becomes $G' = (V, E \setminus \{(i, j)\})$.
3. *ADDvertex(i)*which adds vertex i to G. The updating graph becomes $G' = (V \cup \{i\}, E \cup E_i)$, with E_i be a edge set connecting i to other vertices of $V(G)$.
4. *MOVedge(i)*which deletes vertex i from G. Then the new graph becomes $G' = (V \setminus \{i\}, E \setminus E_i)$, with E_i be a edge set connecting i to other vertices of $V(G)$.

Each one is described as follows:

3..2.1 Edge deletion

We sketch how to update the connected components and a minimal FVS in G when an edge (i, j) is deleted from G. First, if i and j are not belong to the same connected component, then either i or j is in S. In this case, we update only the minimal FVS of G'.

1. *MOVedge(i,j)* .
{
If $i.c = j.c$ **then**
$i.c = i$;
For each element k of the array $i.s$ do $i.s[k] = i$;
$j.c = j$;
For each element k of the array $j.s$ do $j.s[k] = j$;
}

For the description of *MOVedge(i,j)* procedure, we distinguish two cases:

Case 1 If i and j belong to the same component of $G[F]$, then by removing edge (i, j) from G, the component $W_{i.c}$ must be split into two new components, and each one must have a different index. As $j \neq i$, so the first component which contains i becomes in W_i and the second component becomes in W_j

Case 2 If i and j are not belong to the same connected component, then removing edge in G does not create a cycle in the new graph G'. But if $i \in S$ (resp. $j \in S$), the removed edge produces a change of the state

of i (resp. j) from the state s to the state f. This problem is solved by applying *Update_Minimal_FVS*, which means that, if i (resp. j) does not create a cycle in $G[F]$, then i (resp. j) belongs to F).

Since, at least one vertex from S belongs to F, hence for all cases, the complexity of *Update_Minimal_FVS* is $O(|F|)$, because at most n vertices change their statut only once.

3..2.2 Edge insertion

2. *ADDedge(i,j)* /* *with* $d_i \geq d_j$

{
  If $(i.e = j.e = f)$ **then**
    If $i.c = j.c$ **then**
      $j.e = s$;
}

The edge insertion procedure starts with a sequence of split of a component W_c which contains the removed edge. Since, if i and j belong to the same component, then we must select arbitrary one vertex from $\{i, j\}$ to be removed (in our procedure, we chose j as a removed vertex). By removing j, W_c must splits into at least $\Delta - 2$, because two neighbors of j must belong to a same component. In this case, the indexes of the new components are given respectively by its neighbors of j.

3..2.3 Vertex insertion

3. *ADDvertex(i)*

{
  If i creates a cycle in $G[F]$ (*i.e.* i has two neighbors in the same component of $G[F]$) **then** $i.e = s$;
  Else
    $i.e = f$;
    $i.c = min\{i.s[k] : 1 \leq k \leq |N(i)|\}$;
    For each element k of the array $i.s$ do $i.s[k] = i.c$;
}

When we add a new vertex i in the graph G, then i is affected to S if i creates a cycle in $G[F]$, otherwise it is affected to F. Since this can hold $O(n)$ times, because if $i \in S$, then we obtain a new minimal FVS after zero updating. If $i \notin S$, then we obtain a new minimal FVS afer an updating of $G[f]$.

3..2.4 Vertex deletion

4. *MOVvertex_i(j)* /* with j is a vertex of F and i is the removed vertex.

{
  If $(\exists i \in (|F| \cap N(j)))$ and (i is removed) **then**
    $j.c = j$;
    For each element k of the array $j.s$ do $j.s[k] = j.c$;
}

To implement the remove vertex operation, we distinguish the two following cases:

1. If the removed vertex i belongs to S, then the new graph G' remains an acyclic graph and $S' = S \setminus \{i\}$ is a minimal FVS of G'.

2. If the removed vertex i belongs to F. In this case, the updating graph G' creates $\underset{G[F]}{d_i} - 1$ components, with $\underset{G[F]}{d_i}$ is the degree of i in $G[F]$. So, the index of each component is given respectively by its neighbor of i. As there are $\underset{G[F]-1}{d_i}$ new components, then at least $\underset{G[F]}{d_i} - 1$ vertices of S can belong to F. Therefore, for maintaining a minimal FVS of G', at least $\underset{G[F]}{d_i}(|F| - \underset{G[F]}{d_i}) \leq \Delta(n - \Delta)$ vertices must be updating.

When an update operation is done on a graph (by adding or moving a vertex or an edge), the data structures must be updating. In this case, both of the connected components and a minimal FVS, must be updating. So, we propose the following two update procedures :

1. ***Update_Components_Vertex(i)***. Each connected component can be update an edge or a vertex insertion/deletion to/from G, by applying a few locally greedy adjustments. *Update_Components_Vertex(i)* operation is executed by each vertex i of F, in order to update each local variable $i.c$ of i, to obtain the new connected components of the new graph after updating. The implementation of *Update_Components_Vertex(i)* operation starts by verifying the state of i according to the states of its neighbors. However, if there exists one neighbor of i in F which changes its connected component, then i also changes its connected component, and belongs to the same connected component of its neighbor.

If $i.e = f$ **then**
{
  If $\exists j \in N(i)$ and $i.s[j] \neq j.c$ **then**
  {
    $i.c = j.c$;
    For each element j of the array $i.s$ do $i.s[j] = i.c$;
  }
}

2. ***Update_FVS_Vertex(i)***

After each update operation (in particular, for edge insertion and vertex deletion opertaions[1]), some vertices of S need to update its status and belong to the new forest $G[F']$.

If $i.e = s$ **then**
  If i does not create a cycle in $G[F]$ **then**
  {
    $i.e = f$;
    $i.c = i$;
    For each element j of the array $i.s$ do $i.s[j] = i$;
  }

4. Conclusion

In this paper, we have proposed a distributed fully dynamic algorithm for maintaining a minimal FVS of dynamic graphs. In Section 3, we have shown that the complexity of the proposed distributed fully dynamic algorithm is $O(\Delta(n - \Delta)$.

[1]Because after this two update operations, the new graph can split into new connected components. As result, some vertices of S cannot create a cycle in the new graph $G[F']$, so it changes its status from s to f and belongs to F'.

References

[1] F. Tigrine and H. Kheddouci. A Self-Stabilizing Algorithm for Constructing a Minimal Feedback Vertex Set of Graphs. *Technical report. LIESP Laboratory, Lyon 1 University 2006.*

[2] F. Tigrine and H. Kheddouci. The Minimum Feedback Vertex Set for Kronecker Product of Graphs. *Utilitas Mathematica (to appear).*

[3] H. Kheddouci and O. Togni. Minimum Feedback Vertex Set in Distance Graphs and Circulant Graphs. *Submitted (2006).*

[4] V. Bafna, P. Berman and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Discrete Math. 12 3 (1999), pp. 289 − 297.*

[5] B. Schwikowskia and E. Speckenmeyer. On enumerating all minimal solutions of feedback problems. *Discrete Applied Mathematics 117 (2002) 253265.*

[6] L. Lamport. Solved pronlems, unsolved problems and non-problems in concurrency. *In Proc. 3rd ACM Symp. on Principles of Distributed Computing (1984) 1 − 11.*

[7] D. Eppstein, G.F. Italiano, R. Tamassia, R. E. Tarjan, J. Westbrook and M. Yung. Maintenance of a minimum spanning forest in a dynamic plane graph. *J. of Algorithms 13 (1992), 33 − 54.*

[8] G. Ausiello, G. F. Italiano, A. Marchetti Spaccamela and U. Nanni. Incremental algorithms for minimal length paths. *Proc. 1st ACM-SIAM Symp. on Discrete Algorithms (1989), to appear.*

[9] G. F. Italiano. Finding paths and deleting edges in directed acyclic graphs. *Inform. Process. Lett. 28 (1988), 5-11.*

[10] D. YeBin. A dynamic transitive closure algorithm. *Research Report, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY, 1988.*

[11] J. H. Reif. A topological approach to dynamic graph connectivity. *Inform. Process. Lett. 25 (1987), 65-70.*

[12] G. N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput. 14 (1985), 781-798.*

[13] H. Rohnert. UA dynamization of the ah pairs least cost path problem. *Proc. 2nd Annual Symp. on Theoretical Aspects of Computer Science (STACS 85), Lecture Notes in Computer Science, vol. 182, Springer-Verlag, Berlin, 1985, 279-286.*

[14] T. Ibaraki and N. Katoh. On-line computation of transitive closure for graphs. *Inform. Process. Lett. 16 (1983), 95-97.*

[15] S. Even and Y. Shiloach. An on-line edge deletion problem. *J. Assoc. Comput. Much. 28 (1981), 1-4.*

[16] A. Shamir. A linear time algorithm for finding minimum cutsets in reducible graphs. *SIAM J. Comput. 8 4 (1979), pp. 645 − 655.*

[17] F. Chin and D. Houck. Algorithms for updating minimum spanning trees. *J. Comput. System Sci. 16 (1978), pp. 333 − 344.*

[18] M.R. Garey and D.S. Johnson. Computers and Intractability. *Freeman, San Francisco, CA (1979).*