Maintaining chordal graphs dynamically: improved upper and lower bounds

Niranka Banerjee[1] Venkatesh Raman[1] Srinivasa Rao Satti[2]

[1]The Institute of Mathematical Sciences, HBNI, Chennai, India

[2]Seoul National University

CSR 2018, Moscow

- A graph is chordal if every cycle of size four or more in the graph has a chord.

- A graph is chordal if every cycle of size four or more in the graph has a chord.

- There are $O(m + n)$ time algorithms to detect whether a graph is chordal by computing what is called a *perfect elimination ordering* or *a clique tree decomposition* of the graph.

- A graph is chordal if every cycle of size four or more in the graph has a chord.
- There are $O(m+n)$ time algorithms to detect whether a graph is chordal by computing what is called a *perfect elimination ordering* or *a clique tree decomposition* of the graph.
- A perfect elimination ordering in a graph is an ordering of the vertices of the graph such that for each vertex x, x and the neighbors of x that occur after x in the ordering form a clique.

- A graph is chordal if every cycle of size four or more in the graph has a chord.
- There are $O(m+n)$ time algorithms to detect whether a graph is chordal by computing what is called a *perfect elimination ordering* or *a clique tree decomposition* of the graph.
- A perfect elimination ordering in a graph is an ordering of the vertices of the graph such that for each vertex x, x and the neighbors of x that occur after x in the ordering form a clique.
- A *clique tree* of a graph is a tree decomposition of the graph, where the bags in each node of the decomposition induce a maximal clique.

- Can we do better than $O(m+n)$ if a single edge is either deleted or added to an existing chordal graph?

- Can we do better than $O(m + n)$ if a single edge is either deleted or added to an existing chordal graph?
- Ibarra developed two fully dynamic algorithms for maintaining chordality.
  First one has a query and update time of $O(n)$,
  the other has a query time of $O(\sqrt{m})$ and the update time of $O(m + n)$
- Mezzini gave an algorithm which took $O(1)$ query and $O(n^2)$ update time
- Berry et al. improved upon Ibarra's insertion query from $O(n)$ to $O(1)$, while the other bounds remained the same

- Our results are in the decremental setting.
- Our first result is based on the maximum size k of a bag in the clique tree.
- In the worst case a delete query takes $O(1)$ time and update takes $O(n + k^2)$ time. Moreover, the update time is actually $O(n^2/\Delta + k^2)$ amortized over Δ edge deletions

- Our results are in the decremental setting.
- Our first result is based on the maximum size k of a bag in the clique tree.
- In the worst case a delete query takes $O(1)$ time and update takes $O(n + k^2)$ time. Moreover, the update time is actually $O(n^2/\Delta + k^2)$ amortized over Δ edge deletions
- Our second result is based on maintaining a perfect elimination ordering of the graph.
- We can detect chordality in $O(\min\{degree(u), degree(v)\})$ and update the resulting chordal graph in $O(degree(u) + degree(v))$ time.

- Our results are in the decremental setting.
- Our first result is based on the maximum size k of a bag in the clique tree.
- In the worst case a delete query takes $O(1)$ time and update takes $O(n + k^2)$ time. Moreover, the update time is actually $O(n^2/\Delta + k^2)$ amortized over Δ edge deletions
- Our second result is based on maintaining a perfect elimination ordering of the graph.
- We can detect chordality in $O(\min\{degree(u), degree(v)\})$ and update the resulting chordal graph in $O(degree(u) + degree(v))$ time.
- We show that any structure to maintain a chordal graph requires $\Omega(\log n)$ amortized time for a query or an update in the cell probe model.

Theorem

Let G be a chordal graph. Let k be the maximum size of a clique in G. We can construct a data structure such that given an edge (u, v) to be deleted from G, we can report in $O(1)$ time if $G \setminus (u, v)$ is chordal and if it is, we can update the structure in $O(n + k^2)$ time.

Theorem

Let G be a chordal graph. Let k be the maximum size of a clique in G. We can construct a data structure such that given an edge (u, v) to be deleted from G, we can report in $O(1)$ time if $G \setminus (u, v)$ is chordal and if it is, we can update the structure in $O(n + k^2)$ time.

First, we state a lemma given by Ibarra,

Lemma

Given a chordal graph G, and an edge $e = (u, v)$, $G \setminus e$ is chordal if and only if u and v are together present in exactly one maximal clique, and hence in only one bag of the clique tree.

Theorem

Let G be a chordal graph. Let k be the maximum size of a clique in G. We can construct a data structure such that given an edge (u, v) to be deleted from G, we can report in $O(1)$ time if $G \setminus (u, v)$ is chordal and if it is, we can update the structure in $O(n + k^2)$ time.

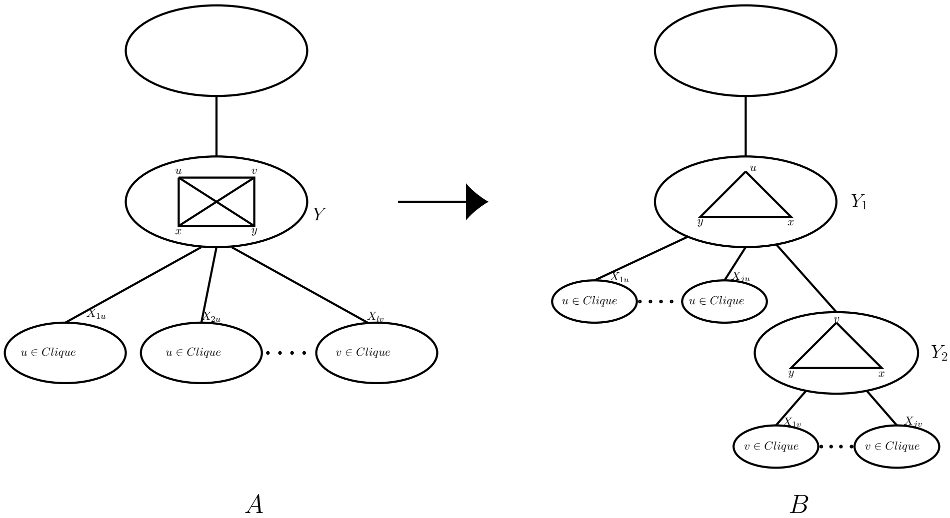

Figure: Figure A represents the clique tree with node Y, the only node containing the edge (u, v). Figure B represents the clique tree after edge (u, v) is deleted.

Theorem

Let G be a chordal graph. Let k be the maximum size of a clique in G. We can construct a data structure such that given an edge (u, v) to be deleted from G, we can report in $O(1)$ time if $G \setminus (u, v)$ is chordal and if it is, we can update the structure in $O(n + k^2)$ time.

Algorithm

1. Check if the given edge (u, v) is present in only one bag, if not report a negative answer, and if yes, then we need to update the clique tree.

Theorem

Let G be a chordal graph. Let k be the maximum size of a clique in G. We can construct a data structure such that given an edge (u, v) to be deleted from G, we can report in $O(1)$ time if $G \setminus (u, v)$ is chordal and if it is, we can update the structure in $O(n + k^2)$ time.

Algorithm

1. Check if the given edge (u, v) is present in only one bag, if not report a negative answer, and if yes, then we need to update the clique tree.

2. If Y is the unique bag containing the edge (u, v), Y is split into two nodes, Y_1 and Y_2. Y_1 contains $Y \setminus u$ and Y_2 contains $Y \setminus v$. From the neighbors of Y_1 remove all nodes which contain u and make them children of Y_2. The other children remain as children of Y_1.

Theorem

Let G be a chordal graph. Let k be the maximum size of a clique in G. We can construct a data structure such that given an edge (u, v) to be deleted from G, we can report in $O(1)$ time if $G \setminus (u, v)$ is chordal and if it is, we can update the structure in $O(n + k^2)$ time.

Algorithm

1. Check if the given edge (u, v) is present in only one bag, if not report a negative answer, and if yes, then we need to update the clique tree.

2. If Y is the unique bag containing the edge (u, v), Y is split into two nodes, Y_1 and Y_2. Y_1 contains $Y \setminus u$ and Y_2 contains $Y \setminus v$. From the neighbors of Y_1 remove all nodes which contain u and make them children of Y_2. The other children remain as children of Y_1.

3. Check whether the bags of any neighbor of these newly formed nodes is a superset of the node. If yes, we "absorb" these nodes into the corresponding neighbor.
   To check whether one node is a superset of the other, maintain the intersection size of two adjacent nodes X and Y. Let ℓ be the size of Y before splitting. If $|X \cap Y| = \ell - 1$ then X absorbs the new Y.

First, we build a clique tree from the given graph G. The clique tree can be represented by a pointer representation where each node points to its parent in the tree. Furthermore, we maintain the following structures.

- For each edge in the graph G we store

  a counter indicating the number of nodes of the clique tree to which the edge belongs. We can store this structure as an adjacency matrix.

First, we build a clique tree from the given graph G. The clique tree can be represented by a pointer representation where each node points to its parent in the tree. Furthermore, we maintain the following structures.

- For each edge in the graph G we store,

  a counter indicating the number of nodes of the clique tree to which the edge belongs. We can store this structure as an adjacency matrix.

- For each node X in the clique tree, we store

  the list of vertices sorted according to their labels,

First, we build a clique tree from the given graph G. The clique tree can be represented by a pointer representation where each node points to its parent in the tree. Furthermore, we maintain the following structures.

- For each edge in the graph G we store,

  a counter indicating the number of nodes of the clique tree to which the edge belongs. We can store this structure as an adjacency matrix.

- For each node X in the clique tree, we store

  the list of vertices sorted according to their labels,

- For each node Y in the clique tree which is a neighbor of X, we store $|X \cap Y|$ in non-increasing order of values in an array associated with the bag X

- For update, for each of the cases above it takes $O(k^2)$ time to update the counters and the pointers for all edges.

- For update, for each of the cases above it takes $O(k^2)$ time to update the counters and the pointers for all edges.
- We need to update the pointers of all neighbors of Y to point to Y_1 and Y_2. This takes $O(n)$ time.

Theorem

Let G be a chordal graph. We can construct a data structure such that given a sequence of Δ edge deletions, we can support deletion query in $O(1)$ time and deletion update in $O(n^2/\Delta + k^2)$ amortized time.

Theorem

Let G be a chordal graph. We can construct a data structure such that given a sequence of Δ edge deletions, we can support deletion query in $O(1)$ time and deletion update in $O(n^2/\Delta + k^2)$ amortized time.

- Updation of the structures involve the time to split a node in the clique tree and also to absorb the node into one of its neighbors and updating the clique tree.

- We deal with the total time taken to perform the split and absorb operations seperately.

- Let d be the degree of the node Y. Y gets split into multiple nodes. Let us denote these set of nodes to be Y_{split}.

- Let d be the degree of the node Y. Y gets split into multiple nodes. Let us denote these set of nodes to be Y_{split}.

- Whenever a node from Y_{split} splits into two the node size decreases by one and the total cost incurred is the degree of that node.

- Let d be the degree of the node Y. Y gets split into multiple nodes. Let us denote these set of nodes to be Y_{split}.
- Whenever a node from Y_{split} splits into two the node size decreases by one and the total cost incurred is the degree of that node.
- So the total time spent by Y is $O(kd)$. $k \sum d$ is at most $k(n-1)$ and hence we have the total time taken by the algorithm for splitting nodes is $O(kn)$.

- Let $Y's$ neighbor where it gets absorbed be Y_{nbr}. Let d be the degree of the node Y, and d_{nbr} be the degree of the node Y_{nbr} before absorption. The cost of absorption to update the pointers of Y_{nbr} is equal to d.

- Let $Y's$ neighbor where it gets absorbed be Y_{nbr}. Let d be the degree of the node Y, and d_{nbr} be the degree of the node Y_{nbr} before absorption. The cost of absorption to update the pointers of Y_{nbr} is equal to d.

- We associate a charge with every node to account for part of the work done during the absorption.

- Let $Y's$ neighbor where it gets absorbed be Y_{nbr}. Let d be the degree of the node Y, and d_{nbr} be the degree of the node Y_{nbr} before absorption. The cost of absorption to update the pointers of Y_{nbr} is equal to d.
- We associate a charge with every node to account for part of the work done during the absorption.
- If Y gets absorbed into Y_{nbr}, we add a charge of d to Y_{nbr}. So the new charge at Y_{nbr} is the old charge in that node plus the charge at Y plus d.

- Let $Y's$ neighbor where it gets absorbed be Y_{nbr}. Let d be the degree of the node Y, and d_{nbr} be the degree of the node Y_{nbr} before absorption. The cost of absorption to update the pointers of Y_{nbr} is equal to d.

- We associate a charge with every node to account for part of the work done during the absorption.

- If Y gets absorbed into Y_{nbr}, we add a charge of d to Y_{nbr}. So the new charge at Y_{nbr} is the old charge in that node plus the charge at Y plus d.

- The total charge accumulated at any node is at most d^2.

- Let $Y's$ neighbor where it gets absorbed be Y_{nbr}. Let d be the degree of the node Y, and d_{nbr} be the degree of the node Y_{nbr} before absorption. The cost of absorption to update the pointers of Y_{nbr} is equal to d.
- We associate a charge with every node to account for part of the work done during the absorption.
- If Y gets absorbed into Y_{nbr}, we add a charge of d to Y_{nbr}. So the new charge at Y_{nbr} is the old charge in that node plus the charge at Y plus d.
- The total charge accumulated at any node is at most d^2.
- The total charge on the existing nodes at any point of time is at most $4n^2$.

- Let $Y's$ neighbor where it gets absorbed be Y_{nbr}. Let d be the degree of the node Y, and d_{nbr} be the degree of the node Y_{nbr} before absorption. The cost of absorption to update the pointers of Y_{nbr} is equal to d.
- We associate a charge with every node to account for part of the work done during the absorption.
- If Y gets absorbed into Y_{nbr}, we add a charge of d to Y_{nbr}. So the new charge at Y_{nbr} is the old charge in that node plus the charge at Y plus d.
- The total charge accumulated at any node is at most d^2.
- The total charge on the existing nodes at any point of time is at most $4n^2$.
- We spend another $O(k^2)$ time for each update to update the nodes corresponding to every pair of vertices in the bag that got split.

We now give a decremental algorithm using perfect elimination ordering (PEO).

Theorem

Let G be a chordal graph represented by its adjacency list. Given a perfect elimination order of G whenever an edge (u, v) is deleted, we can determine if $G \setminus (u, v)$ is chordal in $O(\min\{degree(u), degree(v)\})$ time, and update the structures if it is the case, in $O(degree(u) + degree(v))$ time.

We now give a decremental algorithm using perfect elimination ordering (PEO).

Theorem

Let G be a chordal graph represented by its adjacency list and adjacency matrix. We can, in $O(m+n)$ time, construct a PEO of G, such that whenever an edge (u,v) is deleted, we can determine if $G \setminus (u,v)$ is chordal in $O(\min\{degree(u), degree(v)\})$ time, and update the structures if it is the case, in $O(degree(u) + degree(v))$ time.

Towards that we first state the following characterization.

Lemma

Let G be a chordal graph, and let $e = (u,v)$ be an edge. $G \setminus (u,v)$ is chordal if and only if all the common neighbors of u and v are adjacent to each other, i.e., they form a clique.

- Upon deletion of the edge (u, v), we scan and find the vertices that are common neighbors to both u and v.

- Upon deletion of the edge (u, v), we scan and find the vertices that are common neighbors to both u and v.
- Let the first vertex in the PEO be a_1. It is sufficient to check a_1's adjacency with everyone else.

- Upon deletion of the edge (u, v), we scan and find the vertices that are common neighbors to both u and v.

- Let the first vertex in the PEO be a_1. It is sufficient to check a_1's adjacency with everyone else.

- To update the PEO when the edge (u, v) is deleted, we first observe that we only need to worry about common neighbors of u and v that appear before u.

- Upon deletion of the edge (u, v), we scan and find the vertices that are common neighbors to both u and v.

- Let the first vertex in the PEO be a_1. It is sufficient to check a_1's adjacency with everyone else.

- To update the PEO when the edge (u, v) is deleted, we first observe that we only need to worry about common neighbors of u and v that appear before u.

- Let $a_1, a_2, \ldots, a_k$ be the set of all vertices which are common neighbors of u and v that appear before u in the PEO. To fix the PEO, we move all these vertices, in the same order, to the position immediately after u in the PEO.

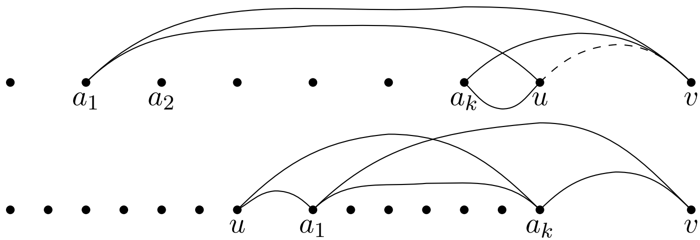

Figure: $a_1, a_2, ...a_k$ represent the common neighbors of u and v. The top figure shows the original PEO. The dotted edge $\{u, v\}$ is deleted from the graph. The bottom figure shows the new PEO after deletion of the edge $\{u, v\}$.

Theorem

Any dynamic structure that maintains a chordal graph under edge insertions and deletions requires $\Omega(\log n)$ *amortized time per update or query in the cell probe model of word size* $\log n$.

Theorem

Any dynamic structure that maintains a chordal graph under edge insertions and deletions requires $\Omega(\log n)$ amortized time per update or query in the cell probe model of word size $\log n$.

We reduce the following to our problem,

Theorem

Patrascu had shown that any dynamic data structure that performs a sequence of n edge insertions and deletions that maintains a forest starting from an edgeless graph. Suppose the structure also supports queries of the form whether a pair of vertices are in the same connected component. Then such a structure requires $\Omega(\log n)$ amortized time per query and update to support a sequence of n query and update operations in the cell probe model of word size $\log n$.

Theorem

Any dynamic structure that maintains a chordal graph under edge insertions and deletions requires $\Omega(\log n)$ *amortized time per update or query in the cell probe model of word size* $\log n$.

- The main idea is to ensure that when a query for a pair (u, v) comes, we add a new path of length three between u and v.

Theorem

Any dynamic structure that maintains a chordal graph under edge insertions and deletions requires $\Omega(\log n)$ *amortized time per update or query in the cell probe model of word size* $\log n$.

- The main idea is to ensure that when a query for a pair (u, v) comes, we add a new path of length three between u and v.
- Check whether the resulting graph is chordal.

Theorem

Any dynamic structure that maintains a chordal graph under edge insertions and deletions requires $\Omega(\log n)$ *amortized time per update or query in the cell probe model of word size* $\log n$*.*

- The main idea is to ensure that when a query for a pair (u, v) comes, we add a new path of length three between u and v.
- Check whether the resulting graph is chordal.
- If the pair of vertices are in different components, then the new additions don't add any cycle,

Theorem

Any dynamic structure that maintains a chordal graph under edge insertions and deletions requires $\Omega(\log n)$ amortized time per update or query in the cell probe model of word size $\log n$.

- The main idea is to ensure that when a query for a pair (u, v) comes, we add a new path of length three between u and v.
- Check whether the resulting graph is chordal.
- If the pair of vertices are in different components, then the new additions don't add any cycle,
- If they are in the same component, then new additions create a chordless cycle of length greater than three.

- An interesting open problem is to prove a super logarithmic lower bound for the query and update operations for maintenance of chordal graphs.
- Another problem would be to make our algorithms fully dynamic.

Thank You!