Efficiently computing K-edge connected components

Team Number 11

Akshara P 181IT132 (+91 8310316175)	Adharsh Kamath 181IT202 (+91 8296678775)	Shidharth S 181IT243 (+91 9611304825)	Sriram Rao Udupi 181IT246 (+91 9480530464)

Connected components

Definitions:

- A **K-edge connected component** of a graph is a (maximal) connected subgraph which remains connected even after removing (any) k-1 edges from it
- A **cut** C, of a graph G = (V,E) is composed of two sets of vertices, S and T, where S and T are partitions of V
- The **value of a cut**, w(S,T), is the number of edges that go from a vertex in one of the vertex-sets to the other
- A **minimum *s-t* cut** of a graph G, is a cut, such that *s* and *t* are in different sets and the value of the cut is the least possible (of all *s-t* cuts)
- The **global min-cut** of a graph G is a cut of G, with the least value of all possible cuts of G

Maximum Adjacency Search (MAS)

The MAS algorithm is used to find the global min cut in a graph. Pseudocode:

```
MAS(Graph):
    while |V| > 1:
        A ← a
        while A ≠ V:
            add the most tightly connected vertex to A
            store the vCut and shrink G by merging the two vertices added last
    if the vCut is lighter than the current_minimum_cut
        store the vCut as the current_minimum_cut
```

(vCut is the cut of V that separates the vertex added last from the rest of the graph)

MAS Algorithm Complexity

- In MAS, Extract-max is called $|V|$ times (while finding the most tightly connected vertex) and Increase-key is called $|E|$ times (while merging two vertices)
- In a Fibonacci heap, Extract-max takes $O(\log|V|)$ amortized time and Increase key takes $O(1)$ time
- Total time complexity of MAS implemented using Fibonacci heap is $O(|E| + |V| \log |V|)$
- MAS is called $|V|-1$ times to find the global min-cut

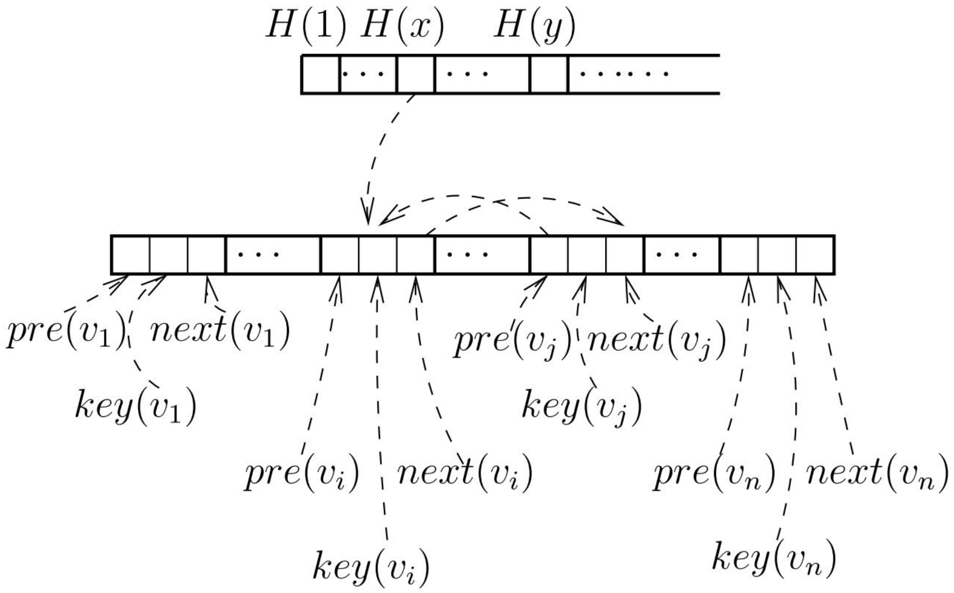
Total time complexity due to MAS: $O(|V||E| + |V|^2 \log |V|)$

Linear Heap ADT

- **Needs to support 2 operations on keys**, where a key of a vertex v is the sum of the weights of the edges connecting it to the current state of MAS queue: **update-key() and extract-max().**

- Other operations:
  init()
  insert()
  remove()

- **Doubly linked lists coupled with a head table**. Doubly linked list implemented as 3 arrays of size |V| each to store vertex keys, vertex predecessors, and successors. H(x) stores the first vertex in the list whose key value equals to x.



- **init():** Creates the LinearHeap. Initialize head table with max value = n. Set ***po*** (max key value) = 0.
- **insert():** Insert (vertex id, key) pairs into the list. Update head table. Keep track of max key value.
- **remove():** Removes a (vertex id, key) pair from the list. Updates pre, next, head arrays.
- **compress():** Utility function to keep track of the min and max key values in the head table.

Main ADT Operations

Update-key()

Update-key(v)**:**

remove v from the doubly linked list $H(key(v))$;
update $key(v)$ and insert v into $H(key(v))$;
$po \leftarrow key(v)$ if $key(v) > po$;

Time Complexity: O(1)

Extract-max()

Extract-max(u)**:**

while $H(po) = nil$ **do**
 $po \leftarrow po - 1$;
$u \leftarrow H(po)$, and remove u from $H(po)$;

Time Complexity: O(|E|)

MAS-Linear: Complexity Analysis

If In(i) and De(i) denote the increase and decrease value of *po* in the ith iteration of MAS, time complexity is bounded by **O(|E|+ ΣDe(j))**.

But ΣIn(i) < |E|, and ΣDe(i) < ΣIn(i) and hence,

Time Complexity = O(|E|)

Thus, we can find a minimum cut for an arbitrary vertex pair s and t in time **O(|E|)** using the Linear Heap ADT. This is used while finding the mincut in the decompose() procedure.

Graph Decomposition

- Iteratively decompose a non k-connected subgraph into several connected subgraphs by removing edges in all cuts of G with values less than k.
- The connected subgraphs and intermediate subgraphs can be represented as a tree structure.
- The root the input graph and the leaf nodes represent the k-edge connected components of G.

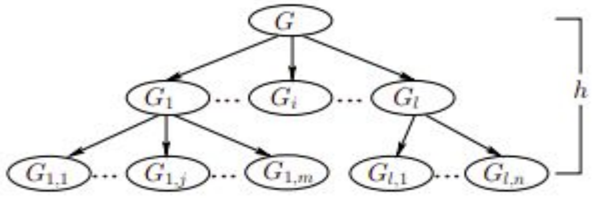


Figure 2: Graph decomposition tree

Graph Decomposition

- The paper also defines a Partition Graph(PG) along with two operators Split and Merge.
- Partition Graph (PG) - A meta graph where a node is related to at least one other node from the original graph.
- Merge operator
  - Merges 2 vertices u,v into a super vertex(V) and removes any existing edges between them.
  - Adds parallel edges such that indegree(V) = indegree(u)+indegree(v) in the PG.
- Split operator
  - Removes all edges of a cut C from a partition graph PG

Decompose Function

- We first construct the corresponding partition graph PG .
- Then, iteratively to update the partition graph until its edge set is empty, through the split operator and the merge operator.
- To choose which operator to be applied depends on the value of the cut found by MinCut (or MAS) which computes the minimum cut in the given partition graph.
- If cut value is less than k then split operation is performed else the merge operation is performed
- Return the decomposed graph.

Graph Decomposition

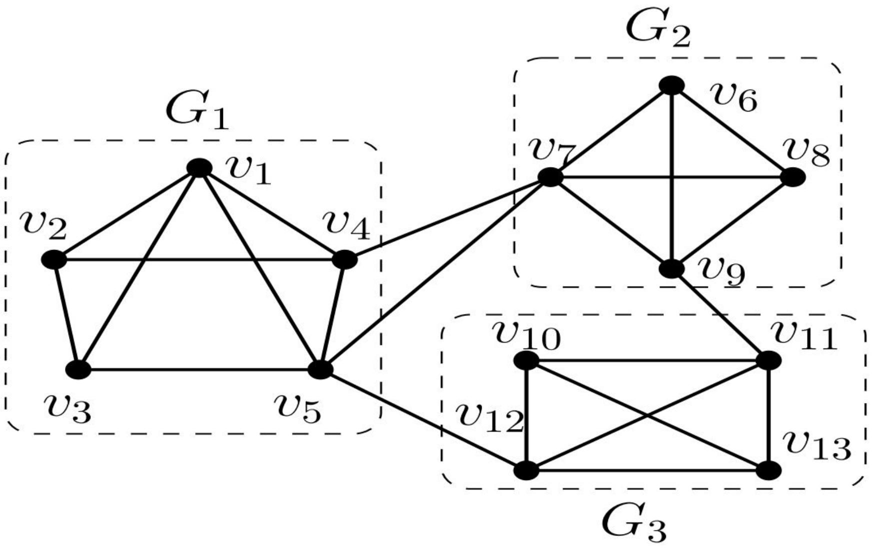
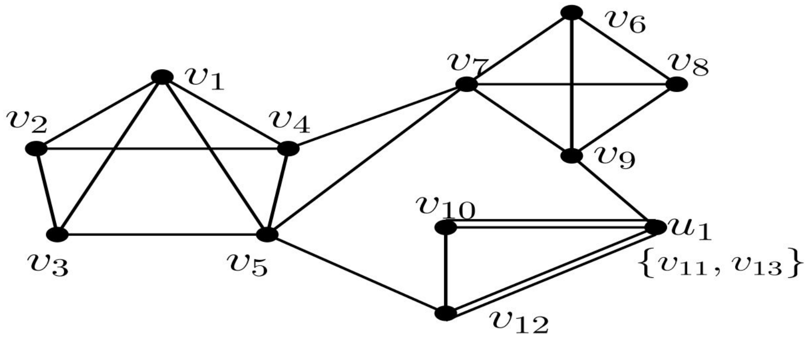
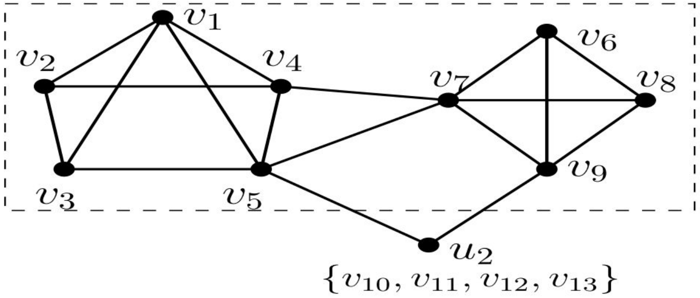


Figure 3: A graph and its 3-edge connected components

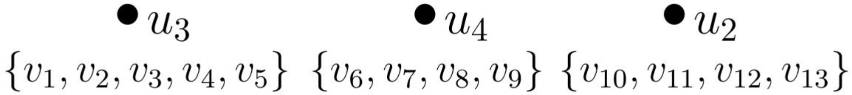
Graph Decomposition



(a) Result of merging v_{11}, v_{13}



(b) Result of merging v_{10}, v_{12}, u_1



(c) Final partition graph

Optimizations

Early merging: The process of merging vertices that happens in decompose after each iteration of MAS can be optimised. The algorithm MAS has the property that for any vertex v in the list at any time, if the sum of number of edges to the vertices in the list before it is greater than k, then the vertex v and the last vertex in the sub list are k connected.

With this optimisation, the vertices can be merged on the fly in each iteration of the MAS algorithm.

This brings the time complexity of the optimised decompose function is O(l*$|E|$), where l is the number of times the procedure MAS(linear) is called.

Optimised Decompose pseudocode

Input: A graph $G = (V, E)$ and an integer k.
Output: Subgraphs of G if $\lambda(G) < k$, and G otherwise.

Construct the corresponding partition graph PG of G, $PG_0 \leftarrow (G_0(\leftarrow G), D(\leftarrow V))$, $i \leftarrow 0$;
while The edge set of PG_i is non-empty **do**
    $PG_{i+1} \leftarrow$ Mas-LMS (PG_i, k);
    $i \leftarrow i + 1$;
return $\phi_k(PG_i)$;

procedure Mas-LMS (G, k)
$L \leftarrow$ {an arbitrary vertex u of V};
Initialize our data structure;
while $L \neq V$ **do**
    $u \leftarrow$ extract-max;
    Add u to L and remove u from the data structure;
    Initialize a queue Q with u;
    while $Q \neq \emptyset$ **do**
        $v \leftarrow Q.pop()$;
        for each $(v, s) \in E$ **with** $s \notin L$ **do**
            if the key of s increases to pass k **then**
                Add s to Q, remove s from the data structure;
            else
                Update-key for s;
        Merge u and v if $u \neq v$;
while $|L| > 1$ **and** the value of the cut implied by the last two vertices in L is less than k **do**
    Split the cut;
    Remove the last vertex from L;

Final algorithm and time complexity

Call the procedure decompose on subgraphs until there is no subgraph to split on

At each level, the complexity is O(I*|E|). Thus, if h is the height of the tree, the total time complexity is O(h*I*|E|)