Advanced Graph Algorithms Jan-Apr 2014

Lecture 2 — January 9, 2014

Lecturer: Saket Saurabh *Scribe: Anantha Padmanabha MS*

1 Overview

In the last lecture we saw algorithms for solving Maximum Independent Set problem for trees using 3 different Paradigms :

- Dynamic Programming
- Divide and Conquer
- Greedy

We also defined **Balanced Separator** for a graph.

In this lecture we try to solve Maximum Independent Set problem for graphs that are $t-$decomposible using Divide and Conquer and Dynamic Programming paradigms.

2 Approaches :

To begin with, the Problem of finding a Maximum Independent Set Problem for a given graph G, is to find $S \subseteq V(G)$ with maximum cardinality such that $\forall u, v \in S(u, v) \notin E(G)$.

This problem is known to be **NP-complete**.

2.1 Definitions

Balanced Seperator : Set $X \subseteq V(G)$ is called a **Balanced Separator** if the size of every connected compenent the graph induced by V(G)\X has atmost $\frac{V(G)}{2}$ number of vertices.

Claim : Every tree has a balanced separator of size 1.

Proof : Consider any $W \subseteq V(G)$. Place the token on an arbitrary vertex of the tree. Suppose this vertex is a Balanced separator for W then we are done. Suppose not, then there exists a component C which contains atleast $\frac{|W|}{2}$ vertices. Move the token to any vertex of this component. This is valid because the component C contains more than $\frac{|W|}{2}$ vertices so all other components put together, must have at most $\frac{|W|}{2}$ vertices. Since the number of vertices of the component C are finite, this process is sure to terminate.

$t-$decomposible Graph : A graph G is said to be $t-$ *decompisible*, if for every $W \subseteq V(G)$ induced graph $G[W]$ has atmost $t-$sized balanced separator.

ζ_t = { G | G is t-decomposible }

- $\zeta_1 \supseteq$ Class of Trees

2.2 Divide and Conquer approach :

Given a graph G which is known to be $t-$decomposible, we design an algorithm to find its Maximum Independent Set. It is assumed that there is an algorithm to obtain a balanced separator for a given graph.

A similar Divide and Conquer Algorithm used for the trees can be used here as well. The algorithm is as follows :

MIS(G)
Input : G which is $t-$ decomposible
Output : MIS of G
if *|E(G)| = 0* **then**
  return V(G)
else
  X ← Balanced Separator of G;
  G″ ← Subgraph Induced by V(G)\X;
  (A″,B″) ← partition of V(G″) such that none of the part has more than $\frac{3|V(G)|}{4}$ vertices;
  Q ← MIS(A″) ∪ MIS(B″);
  M ← Q;
  for *every $I \subseteq X$* **do**
    N(I) ← { x | x is a neighbour of some vertex in I };
    if *I is an Independent Set* **then**
      G′ ← Subgraph Induced by V(G)\(X∪N(I));
      (A′,B′) ← partition of V(G′) such that none of the part has more than $\frac{3|V(G)|}{4}$ vertices;
      P ← MIS(A′) ∪ MIS(B′);
      if *|M| < |P| + |I|* **then**
        M ← P ∪ I;
      end
    end
  end
  return M;
end

Algorithm 1: Divide and conquer algorithm for Max-Ind-Set for a $t-$decomposible graph

Running Time of the Algorithm : For a given graph, first the solution for the partition of $V \backslash X$ is obtained where the two partition takes $T(\frac{3n}{4}) + T(\frac{n}{4})$ time. Then for every possible subset I of X, the partition of $V \backslash (I \cup N(I))$ is obtained where the two partition takes $T(\frac{3n}{4}) + T(\frac{n}{4})$ time. Thus the Recursion of this Algorithm is given by :

$$T(n) \leq T(\frac{3n}{4}) + T(\frac{n}{4}) + 2^t \big(T(\frac{3n}{4}) + T(\frac{n}{4}))\big)$$

$$T(n) \leq 2^{t+2} T(\frac{3n}{4})$$

Solving this Recursion, we get :

$$T(n) = n^{O(t)}$$

For a fixed t, this is a polynomial time algorithm. However, it is desirable to get an algorithm which has a running time of the form $f(t) \cdot n$ (i.e, it is linearly dependent on n).

2.3 Dynamic Programming approach:

The Dynamic Programming approach for trees had a running time of $O(n)$. Thus a modified version of that algorithm can be used for a $t-$decomposible graph also. To do that, every graph G has to be associated with some tree.

An intuitive idea would be to map a graph to a tree such that every node of the tree would be associated with a subset of the vertices of the Graph such that union of all the associations of all the nodes of the tree would contain all the vertices of the graph. To limit the number of vertices associated with each node of the tree, we can use the $t-$decomposible property of the graph.

The following algorithm takes G which is $t-$decomposible, as input and returns (T, r, f) where the tree T is rooted at r and f is a mapping : $V(G) \rightarrow P(V(T))$ and the bag size of each node of the tree T is atmost t.

decompose(G,t)
Input : G, t
Output: (T,r,f) where T is a tree rooted at r and f : V(G) → P(V(T))
if $V(G) \leq t$ **then**
  Create a node v;
  forall the $w \in V(G)$ **do**
    f(w) ← { v };
  end
  return(v,v,f)
else
  $(X, C_1, C_2 ... C_l) \leftarrow$ Balanced Separator of G;
  for *i from 1 to n* **do**
    $G_i \leftarrow$ graph induced on G by the vertices in C_i;
    $(T_i, r_i, f_i) \leftarrow$ decompose(G_i,t);
  end
  Create a node v;
  T ← v;
  forall the $w \in X$ **do**
    f(w) ← $f(w) \cup$ { v };
  end
  for *i from 1 to n* **do**
    T ← T_i;
    E(T) ← (v, r_i);
    f ← f ∪ f_i
  end
  return (T,v,f);
end

Algorithm 2: To associate a graph G which is $t-$decomposible to a tree with bag size atmost t

Thus every graph G is associated with a rooted tree and a function mapping the nodes of the tree to a subset of vertices of the graph.

$$G \to \{T_r, f : V(T) \to 2^{V(G)}\}$$

Let G_v = graph induced graph of G by the vertices $\bigcup_{w \in V(T_v)} f(w)$ where T_v is the sub-tree rooted at v.
The properties of such decomposition are :

- $\bigcup_{v \in V(T)} f(v) = V(G)$
- if v is not an ancestor of u or u is not an ancestor of v in the tree, then there does not exist $a \in G_v$ and $b \in G_u$ such that $(a, b) \in E(G)$.

For Dynamic Programming, we need to maintain, $\forall\ X \subseteq f(v)$, $T_v[X] = MIS\ of\ G_v$ containing X and not containing $f(v) \backslash X$.

Filling these values for the leaves can be done by brute force (Since it contains atmost t elements).

To fill at an intermediate step for a vertex w, Let $X \subseteq f(w)$:

- $T_w[X] = 0$ if there is an edge in between any 2 vertices of X
- $T_w[X] = \sum_{i=1}^{l} \max_{Z \subseteq f(v_i) \setminus N(X)} \mathrm{T}_{v_i}[\mathrm{Z}]$

But this optimization function is valid only if there are no edges between non consecutive edges in the layers of the tree. Supose there are edges in the underlying graph between the non-consecutive layers of the tree, then this memoization cannot keep track of all the edges present. Thus the decomposition that is described earlier does not ensure this property and hence a modified way of decomposition is necessary to ensure this property. In the next class we will try to modify the definition of tree decomposition of a graph so that the desired properties are achieved.