Advanced Graph Algorithms Jan-Apr 2014

Lecture 1 — January 8, 2014

Lecturer: Saket Sourabh *Scribe: Prafullkumar P Tale*

1 Introduction

In this lecture we introduce the Maximum Independent Set Problem (MAX-IND) and Non-Deterministic Polynomial Class(**NP**) along with **NP**–*completeness*. MAX-IND will be used as benchmark to develop further theory.

For a given graph G, an independent set is a set of vertices in a graph, no two of which are adjacent. A maximum independent set is an independent set of largest possible size. Problem R is in class **NP** if it can be solved on *Non Deterministic Turing Machine* in polynomial time. Furthermore it is in class **NP**–*Complete* if it is as hard as any other problem in **NP**, in other words, if every other problem in **NP** can be transformed into R in polynomial time. The books by Hopcroft, Motwani and Ullman [HMU01] and by Papadimitriou, Christos [Pap03] contain introduction to this subject.

In 1972, Richard M. Karp in his ground breaking paper proved that MAX-IND problem, along with 21 other combinatorial problems, is NP complete [Kar72]. This means it is unlikely that we will have efficient (polynomial time) algorithm to solve it for *any* given graph.

Exercise

- Show that MAX-IND is NP-complete by reducing 3SAT problem it to.
- Prove that for any given graph $G(V, E)$, min Vertex Cover + max Independent Set = $|V|$

2 Attacking the Independent Set Problem

Stephen A. Cook first precisely formulated *P vs NP* problem in his paper published in 1971 [Coo71]. Since then it have been one of the major unsolved problem in contemporary mathematics and theoretical computer science. It has central importance in academics and industry, and hence researcher worked on various other methods to attack **NP** problem. Few of them are mentioned below.

2.1 Solving for Special Cases

Instead of trying to find polynomial time algorithm to solve **NP** problems, which may not exists at all, we can try to solve the same problem for a particular class of input. There might be a polynomial time algorithm to solve problem for such special cases owing to their properties. For example, although it has been proved that we can't solve MAX-IND problem for *any* graph in polynomial time, we can do it for bipartite graphs.

Another example of such class is Split Graphs. Split Graph is a graph in which the vertices can be partitioned into a clique and an independent set. Once we decide that given graph belongs to this category, it is very easy to solve MAX-IND problem for this class.

For Trees also there is polynomial time algorithm to find maximum independent set. We will study it in details in following section.

Exercise

- For given graph $G(V, E)$, give an $O(|V| + |E|)$ algorithm to check whether given graph is Split Graph or not.
- For given bipartite graph $G(V_1 \cup V_2, E)$, give an algorithm to find min vertex cover.

2.2 Relaxing optimality condition

Theoretically, the optimal solution for **NP** problems can not be achieved in reasonable time, but practically it will be sufficient to have close to best possible results. We study approximation algorithms, to find approximation solutions to problems for which we know that we don't have efficient polynomial time algorithms. Book by Vijay Vazirani [Vaz01] has detailed description of this subject.

Exercise

- Find $log(n)$ factor approximation algorithm for vertex cover.

2.3 Exact Time Algorithms

For some NP complete problems it is possible to design algorithm that are significantly faster than exhaustive search, though still not in polynomial time. In recent year, there has been growing interesting in design and analysis of such algorithms. This survey by Gerhard Woeginger outlines the development in this area [Woe03].

3 Finding Independent set on Trees

Here on, we will concentrate on one specific NP hard problem (MAX-IND) and will try to solve it for specific class of graphs (Trees). We will see algorithm in three major paradigm to find MAX-IND set in any given tree.

3.1 Divide and Conquer

Before proceeding to algorithms, let us derive the following theorem.

Theorem 1. *For given Tree T, $\exists v \in V(T)$ such that every connected component of $T\backslash\{v\}$ has at most $|V(T)|/2$ vertices.*

Proof. Select any vertex $u \in V(T)$. Let $\{C_1, C_2, \ldots, C_l\}$ be the connected components of $V(T)\backslash\{v\}$. If $\forall i\ |C_i| \leq |V(T)|/2$, then u is the desired vertex. If not, there exists a *oversized* component, say C_i, such that $|C_i| > |V(T)|/2$. In this case, consider the child of u in C_i. All the other components $\{C_1, C_2, \ldots, C_{i-1}, C_{i+1}, \ldots, C_l\}$ along with u have less than $|V(T)|/2$ vertices. Hence, child of u or one of its descendant is desired vertex. We will repeat the procedure until there is no *oversized* component. In every step, we are reducing the size of *oversized* components by one, making sure that all other components don't become *oversize*. Hence this process terminates in finite steps at node v such that all the components in $V(T)\backslash\{v\}$ are of size less than $|V(T)|/2$. □

Lemma 2. *Let $\sum_{i=1}^{l} n_i = n$ and $n_i \leq n/2$ then we can partition $\{n_1, n_2, \ldots, n_l\}$ into subset A and B such that*

$$\frac{n}{4} \leq \sum_{n_i \in A} n_i \leq \frac{3n}{4} \tag{1}$$

Proof. Without loss of generality (wlog), let us assume $n_1 \leq n_2 \leq \cdots \leq n_l$.

$\exists j$ such that $\sum_{i=1}^{j} n_i < n/4$ and $\sum_{i=1}^{j+1} n_i \geq n/4$.

If $\sum_{i=1}^{j+1} n_i = n/4$, then $A = \{n_1, n_2, \ldots, n_{j+1}\}$ and $B = \{n_{j+2}, \ldots, n_l\}$.

If $\sum_{i=1}^{j+1} n_i > n/4$, then $n/4 < \sum_{i=1}^{j+1} n_i < 3n/4$ since $n_{j+1} \leq n/2$ and $\sum_{i=1}^{j} n_i < n/4$. In this case $A = \{n_1, n_2, \ldots, n_{j+1}\}$ and $B = \{n_{j+2}, \ldots, n_l\}$.

Hence proved.

□

Following is the algorithm which returns the Maximum independent set in given tree. We define neighbour of vertex v as set $\{u|(v,u) \in E(T)\}$ and denote it by $N(u)$.

Input:$T(V, E)$
Output: Maximum Independent Set I
if $V(T) = \phi$ **then**
  return ϕ
end
Find $v \in V(T)$ such that $V(T)\backslash\{v\}$ can be partitioned into A and B such that $|V(T)|/4 \leq |A|, |B| \leq 3|V(T)|/4$;
$I_1 \leftarrow$ Maximum Independent Set in $T\backslash\{v\}$;
$I_2 \leftarrow$ Maximum Independent Set in $T\backslash N(v)$;
return $\max\{I_1, \{v\} \cup I_2\}$

Algorithm 1: Finding MIS in Tree using Divide and Conquer

To find the running time of this algorithm, we will use the following lemma 2.

$$T(n) = 2 * T(n/4) + 2 * T(3n/4) + O(n)$$

$$T(n) \leq 4 * T(3n/4) + O(n)$$

Hence $T(n) = n^{log_{4/3}(4)}$.

3.2 Dynamic Programming

We will follow bottom up approach to build the maximum independent set in Tree. For every vertex define –

$A[v] \leftarrow$ Maximum Independent Set of subtree rooted at v and containing v.
$B[v] \leftarrow$ Maximum Independent Set of subtree rooted at v and not containing v.

With this notation, we define the following algorithm.

Input: $T(V, E)$ rooted at r
Output: Maximum Independent Set
for u *in* $V(T)$ **do**
  $A[u] \leftarrow 0$;
  $B[u] \leftarrow 0$;
end
$Leaf(T) \leftarrow$ All leaf nodes of T';
for u *in* $Leaf(T)$ **do**
  $A[u] \leftarrow 1$;
end
$V' \leftarrow$ Vertex in $V(T)$ in the ascending order of distance from root
for u *in* V' **do**
  forall the w *which are children of* u **do**
    $A[u] \leftarrow A[u] + B[w]$;
    $B[u] \leftarrow B[u] + \max\{A[w], B[w]\}$;
  end
end
return $\max\{A[r], B[r]\}$;

Algorithm 2: Find Maximum Independent Set in Tree using Dynamic Programming

Since every vertex is visited only once the running time of this algorithm is $O(n)$.

3.3 Greedy Algorithms

It is more convenient to find Vertex Cover using greedy approach and then find Independent set by taking compliment of it in vertex set. Following is the algorithm to find Vertex Cover in Tree using greedy approach.

Input: $T(V, E)$
Output: Vertex Cover VC
$VC \leftarrow \Phi$;
while $V(T)$ *is not empty* **do**
  Remove all isolated vertex; Find vertex u of degree 1;
  $VC \leftarrow VC \cup N(u)$;
  $T \leftarrow T \backslash N(u)$
end
return VC

Algorithm 3: Finding Vertex Cover in Tree using Greedy approach

Above algorithm outputs minimum vertex cover. Maximum independent set is obtained by taking the complement of this vertex cover in $V(T)$.

4 Balance Separator

We end this lecture by introducing the balance separator.

Definition 3. ***Balance Separator:*** *For given graph G, a set $X \subseteq V(G)$ is called balance separator if all the connected components of $G \backslash X$ are of size $\leq |V(G)|/2$.*

References

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[HMU01] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation, 2nd, 2001.

[Kar72] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[Pap03] Christos H Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[Vaz01] Vijay V Vazirani. *Approximation algorithms*. springer, 2001.

[Woe03] Gerhard J Woeginger. Exact algorithms for np-hard problems: A survey. In *Combinatorial Optimization-Eureka, You Shrink!*, pages 185–207. Springer, 2003.