# Lab11-Approximation

Algorithm: Analysis and Theory (X033533), Xiaofeng Gao, Spring 2018.

∗ If there is any problem, please contact TA Mingding Liao.

*

1. **Dominating Set Problem:** Given a graph $G = (V, E)$, find the minimal dominating set. Dominating set is a subset $D$ of $V$ such that every vertex not in $D$ is adjacent to at least one member of $D$.

   **Steiner Minimum Tree Problem:** Given a graph $G = (V, E)$ with a distance function on $E$, and a subset $S \subseteq V$, find a shortest tree interconnecting the vertices in $S$.

   What are the fourtuple $(I, Sol, m, goal)$ in Dominating Set Problem and Steiner Minimum Tree Problem?

   **Solution:**

   Dominating Set Problem:

   Given a graph $G = (V, E) | G$ is a graph
   sol(G) $= D \subset V | \forall_v \notin D, \exists_u \in D, (u, v) \in E$
   $m(G, D) = |D| = min$
   Hence, $goal = min$

   Steiner Minimum Tree Problem:

   I=Graph G=(V,E)
   cost of the edges $c :\rightarrow R^+$, $S \subseteq V$
   sol(G) = A
   $T = (V_T, E_T)$ in G, such that $S \subseteq V_T \subseteq V$, $E_T \subseteq E$
   $m(G, T) = \sigma_{e \in E_T} c(e) = min$
   Hence, $goal = min$

   Reference

   NP Optimization Problem from Lecture Slides

   http://www.nada.kth.se/~viggo/problemlist/compendium.html

   https://en.wikipedia.org/wiki/Optimization_problem

2. The TA of algorithm course is required to solve $n$ problems but he cannot solve any one. Luckily, the TA has $m$ genius friends and every friends can solve some of these problems (different friends may be able to solve different problems), so the TA wants to ask the fewest friends for help to solve these $n$ problems. The TA is foolish, so he enumerates all of the possible cases of friend subsets and find the best one (minimal friend subset). One of TA's roommate (called Z) does not attend the algorithm course, so he gives TA a naive suggestion: ask for all of $m$ friends for help. Another roommate (called X) learns algorithms well, and he finds that the friend selection problem can be solved by an approximation algorithm introduced in the class.

   The questions are:

   (a) Please describe the approximation algorithm mentioned by X.

   (b) You should analyze the solutions of TA and his roommates and discuss whose solution is better in aspects of time complexity and APX class.

**Solution:**
Let U is the set of problems N, Let S is a set of m subsets of U which also covers U, Here each subset denotes the set of problems that a friend can help in solving the problem N. So $S_i$ subset will be $c(S_i)$.
Hence by Set Cover Problem, A subcollection $S^{'} \subseteq S$ covers all the elements of U, Where U is the set of problems N so, U= $N_1, ...., N_n$. So the total cost of the chosen subcollection can be $\sigma_{S_i \in S'} c(S)$.
So the greedy Set cover algorithm will be

---

**Algorithm 1:** Greedy Set Cover Algorithm

---

**INPUT:** U with n item; S with m subsets; cost function c(Si)

**OUTPUT:** A subset $S^{'} \subseteq S$ Such that $U_{e_i \in S_k \in S'}\ e_i = \cup$

C $\leftarrow \emptyset$

While C **IS NOT** U do

Find the most cost effective Set S, Set C $\leftarrow$ C $\cup$ S

$\forall_e \in \frac{S}{C}$, set price (e)=$\frac{c(S)}{|S-C|}$

**end While**

Output: Selected S

---

The cost-effectiveness of a set S is the average cost at which it covers new elements; The price of an element e is the average cost when e is covered.
The time complexity of the Z algorithm is very simple as the Teaching Assistant has no work to do other than calling his friends for help to solve the assignments. However by using set cover problem, we can assume that O(mn) run-time is the run-time complexity of the algorithm. As There are at most O(minm, n) iterations to select the subcollection. Within each iteration to find the minimum cost-effectiveness, it requires O(m) times.
There are totally n elements, and each ei, the price modification will perform at most O(m) times, each with linear operations. Totally the price updating procedure requires O(mn) time. Thus the total running time is $O(minm, n)O(m) + O(mn) = O(mn)$
Reference
Slides From the lecture. Approximation

3. Let $G = (V, E)$ be an undirected graph and $S \subseteq V$. We define $E(S)$ to be the edges induced by $S$, says, $E(S) = \{(u, v) \in E \mid u \in S, v \in S\}$. Then we define the density $f(S)$ as following:

$$f(S) = \frac{|E(S)|}{|S|}$$

Let $f(G)$ be the maximal density, says, $f(G) = \max_{S \subseteq V}\{f(S)\}$. You should design a 2-approximation algorithm by greedy strategy to find $f(G)$ and prove the approximation ratio of your algorithm.

Hint: you could try to remove the node with the minimal degree.

**Solution:**

Let $G = (V, E)$ be an undirected graph and $S \subseteq V$. We define $E(S)$ to be the edges induced by $S$, says, $E(S) = \{(u, v) \in E \mid u \in S, v \in S\}$.

Consider the set S that maximizes f (S). Now, each edge in E(S) must be assigned to a vertex in S. Thus

$|E(S)| \leq |S|.d^{max}$

$f(S) = \frac{|E(S)|}{|S|} \leq d^{max}$

This concludes the proof. Now, the assignment of edges to one of the end points is constructed as the algorithm executes. Initially, all edges are unassigned. When the minimum degree vertex is deleted from S, the vertex is assigned all edges that go from the vertex to the rest of the vertices in S. We maintain the invariant that all edges between two vertices in the current set S are unassigned; all other edges are assigned. At the end of the execution of the algorithm, all edges are assigned.

Let $d^{max}$ be defined as before for the specific assignment constructed corresponding to the execution of the greedy algorithm. The following lemma relates the value of the solution constructed by the greedy algorithm to $d^{max}$.

Let v be the maximum value of f (S) for all sets S obtained during the execution of the greedy algorithm. Then $d^{max} \leq 2_v$

Consider a single iteration of the greedy algorithm. Since $i_{min}$ is selected to be the minimum degree vertex in S, its degree is at most $2\frac{|E(S)|}{|S|} \leq 2_v$

Note that a particular vertex gets assigned edges to it only at the point when it is removed from S. This proves that $d^{max} \leq 2_v$.

Reference:

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.1184 (1.1) (Lemma 3 and 4)

4. Given a directed graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}^+$, the maximum weight acyclic subgraph problem is determining a subset $E' \subseteq E$ of maximum total weight $w(E') = \sum_{e \in E'} w(e)$ such that the subgraph $G' = (V, E')$ induced by $E'$ is acyclic. You should find a 2-approximation algorithm to solve it and prove the approximation ratio of your algorithm. Hint: you could select the edges in the arbitrary order if you know how to find a acyclic subgraph based on the node ID.

   **Solution:**

   We prove that the above algorithm is a 2-approximation for the maximum weight acyclic

---

**Algorithm 2:** 2 Approximation Algorithm

---

**INPUT:** A directed graph G = (V,E) with edge weights $w : E \to \mathbb{R}^+$

**OUTPUT:** A subset $E' \subseteq E$ Such that $G' = (V, E')$ is acyclic

Fix an arbitrary order $u_1, ..., u_n$ on the nodes of G and define $r(u_i) = i$.

Call an edge $(u, v) \in E$ a forward edge if $r(u) < r(v)$ and a backward edge otherwise. Let F and B be the set of all forward and backward edges, respectively.

Let $E' = F$ if $w(F) > w(B)$ and $E' = B$ otherwise.

**return** $E'$

---

subgraph problem.
The algorithm has polynomial running time: Defining r(u) for every node u 2 V takes O(n) time and identifying all forward and backward edges needs O(m) time. Altogether the algorithm thus needs O(n+m) time.
The algorithm computes a feasible solution: Suppose for the sake of a contradiction that the subgraph $G = (V, E)$ contains a cycle $C = (v_1, v_2, ..., v_k = v_1)$
Because C consists only of forward (or backward) edges, we must have $r(v_1) < r(v_2) < \ < r(vk) = r(v_1)(orr(v_1) > r(v_2) > \ > r(v_k) = r(v_1))$, which is a contradiction
The algorithm computes a 2-approximate solution: Note that F and B partition the edge set E, i.e.,
$E = F \cup B$ and $F \cap B = \emptyset$. Also note that $OPT \geq w(E)$ Because we choose the set of larger weight among F and B, we have
$2w(E^{'}) \geq w(F) + w(B) = w(E) \geq OPT$
That is, $w(E^{'} \geq \frac{1}{2}OPT)$
The approximation ratio of is tight as the following example shows: Let $V = u_1, u_2, u_3$ and assume that this is the order fixed in Step 3 of the algorithm.
let E = $(u_1, u_3), (u_3, u_2)$ and $w(e) = 1$ for both edges.
Note that the entire graph is acyclic. Thus the edge set E is the optimal solution with weight $OPT = w(E) = 2$.
The algorithm, however, will output the edge set $E\prime = (u_3, u_2)$ with weight $w(E\prime) = 1$.

Reference: http://wwwhome.math.utwente.nl/~kernw/lectures/DO/Re-Exam-Sol.pdf (Problem 5)