IOE 691: Approximation Algorithms Date: 01/30/2017

Lecture Notes: Local Search (k-Median) and Dynamic Programming

Instructor: Viswanath Nagarajan Scribe: Gian-Gabriel Garcia

1 Local Search

1.1 k-Median

We now show that the 1-swap local search algorithm for k-Median (previous lecture) has a tight approximation ratio of 5. That is, there is some instance where the locally optimal solution costs at least 5 times the optimum. Consider the example in Figure 1. The black circles denote the "clients" whereas the white squares and circles denote possible centers. The bottom row is a local optimum and has cost equal to $2(k-1)+\frac{k+1}{2}$. It is easy to check that exchanging one center from the local optimum with another center, does not lead to any reduction in cost. The top row is the optimal solution with cost $\frac{k+1}{2}$. So the ratio of the algorithm's cost to the optimum is at least $5-o(1)$ as k grows large.

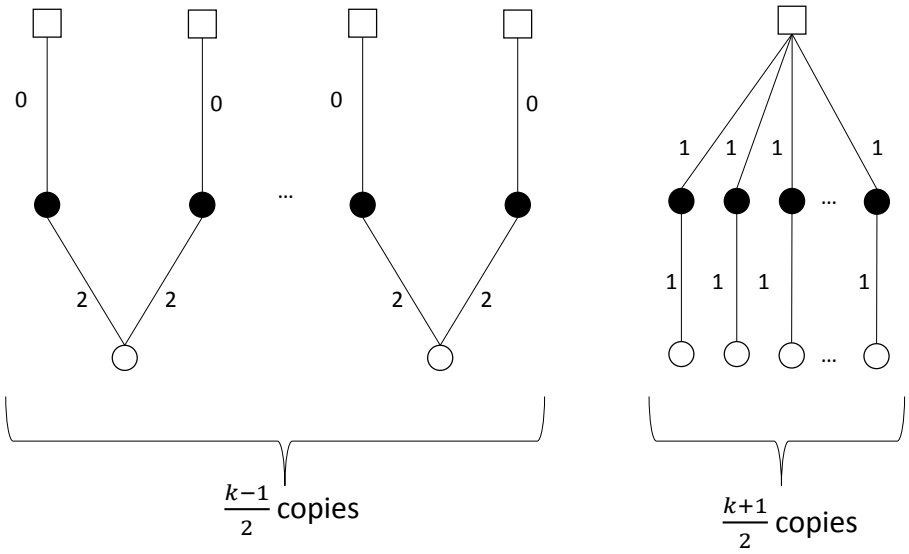

Figure 1: Tight example for 5-approximation of local search algorithm in k-median problem

The local search algorithm can also be modified to swap up to t centers at any step. The authors in [1] generalize their approximation ratio for the local search algorithm and show:

Theorem 1.1 *The t-swap local search for (kM) is a $\left(3+\frac{2}{t}\right)$-approximation.*

1.2 Max Cut

The input for the Max Cut problem is given by a graph $G=(V,E)$ and weight function $w: E \to \mathbb{R}_+$. For ease of notation, we can assume that if $e \notin E$, then $w_e = 0$. The goal is to find a subset S of V which maximizes the weight of all edges between S and $V \setminus S$. Formally, the Max Cut problem is given by

$$\max_{u \in S} \sum_{u \in S, v \in V \setminus S} w_{uv} \qquad \Longleftrightarrow \qquad \max_{e \in \delta(S)} \sum_{e \in \delta(S)} w_e \qquad (MC)$$

where $\delta(S) := \{(u,v) \in E : u \in S, v \notin S\}$.

The local move for (MC) at any solution $S \subseteq V$ is given by either moving some $v \in S$ to $\bar{S} = V \setminus S$ or the opposite so long as it increases the objective function value.

Let OPT denote the optimal value.

Observation 1.1 *OPT can be at most the total weight of all edges. That is, $\sum_{e \in E} w_e \geq OPT$.*

Theorem 1.2 *Any locally optimal solution for (MC) is a $\frac{1}{2}$-approximation.*

Proof: Let S be a local optimal solution obtained using the local search algorithm stated above. Then, it must be the case for all $u \in S$ that $\sum_{v \in \bar{S}} w_{uv} \geq \sum_{v \in S} w_{uv}$, or else there would be a local move which would increase the cost. Now, notice that by summing over all $u \in S$, we have

$$\begin{aligned} \sum_{u \in S, v \in \bar{S}} w_{uv} &\geq \sum_{u \in S, v \in S} w_{uv} \\ 2 \sum_{u \in S, v \in \bar{S}} w_{uv} &\geq \sum_{u \in S, v \in S} w_{uv} + \sum_{u \in S, v \in \bar{S}} w_{uv} \\ 2 \sum_{e \in \delta(S)} w_e &\geq \sum_{e \in E} w_e \\ &\geq OPT. \end{aligned} \qquad (1)$$

From (1), it is clear that S is a $\frac{1}{2}$-approximately optimal solution. ■

Each local move can clearly be implemented in polynomial time, but the number of iterations may not be polynomial (it can depend on the weights). A simple modification leads to a polynomial time algorithm: perform a local move only if the weight of the solution increases by a $1+\epsilon$ factor (for some $\epsilon \geq \frac{1}{n^2}$). As in the k-Median problem, we can now bound the number of iterations by $O(\frac{1}{\epsilon}\log(nw_{max}/w_{min})$ where w_{max} and w_{min} are the maximum/minimum positive weights. This affects Theorem 1.2 only by a small amount as it relies on adding the changes over n local moves, and we obtain a $\frac{1}{2+n\epsilon}$-approximation algorithm.

2 Dynamic Programming

2.1 Stable Set

The input to the stable (independent) set problem is given by a graph $G = (V, E)$ and weights $w : V \to \mathbb{R}_+$.

Definition 2.1 *A set of vertices S is called independent if no pair of vertices in S share an edge.*

The goal is to find an independent set S which maximizes the sum of the weights in S.

Here we show a simple dynamic program that solves the problem exactly on *trees*. Results for many graph problems on trees often extend to larger classes of graphs (eg. planar); so trees are a natural special class of graphs to consider. We consider the tree G to be rooted at some vertex. Let G_v be a subtree routed at vertex v and let $T[v, 0]$ be the maximum weight of the independent set in G_v such that v is not included in the independent set. Similarly, let $T[v, 1]$ be the maximum weight of the independent set in G_v such that v is included. The following dynamic programming algorithm can give an exact optimal solution:

Algorithm 1 Dynamic programming algorithm for Stable Set

Initialize by setting $T[u, 0] = 0$ and $T[u, 1] = w(u)$ for all $u \in V$ which are leaves.
Number vertices $i = 1, ..., n$ such that the index i is increasing from any leaf to the root. Let V_i denote all the children of vertex v_i.

1. For all non-leaf vertices $i = 1, ..., n$:
   Set $T[v_i, j] = \begin{cases} \sum_{k \in V_i} \max\{T[v_k, 0], T[v_k, 1]\} & j = 0 \\ w(v_i) + \sum_{k \in V_i} T[k, 0] & j = 1 \end{cases}$
2. The optimal solution is given by $\max\{T[v_n, 0], T[v_n, 1]\}$ at the root.

2.2 Longest Path in DAG

As another example of dynamic programming we consider the longest path problem on directed acyclic graphs (DAGs). The input is given by a DAG $G = (V, E)$ and two vertices $s, t \in V$. The objective is to find the $s - t$ path with the maximum number of nodes in it.

Since G is a DAG, a topological ordering for the vertices can be created as:

Algorithm 2 DAG Topological Ordering

1. Set $v_1 = s$. Remove s and all edges attached to it from G.
   Set $i \leftarrow 2$.
2. Do until any vertex remains:
   Find a vertex v such that v only has out-going edges.
   Set $v_i \leftarrow v$.
   Set $i \leftarrow i + 1$.
   Remove v and all any edge attached to it from G.

Assume that all nodes are ordered via Algorithm 2. Let $T[i]$ denote the longest path from s to node v_i. The following algorithm can be used to solve the longest path problem exactly. Assume that $v_r = t$.

Algorithm 3 Dynamic programming for longest path in DAG

1. For $i = 1, ..., r$:
   If $i = 1$, set $T[1] = 1$.
   Else, set $T[i] = \max_{j:(v_j,v_i)\in E}(1 + T[j])$.
2. Return $T[r]$.

2.3 Knapsack

The knapsack problem is concerned with picking a maximum-value subset, S, from n items, each with value v_i and size s_i, into a knapsack with capacity B (for sizes). We assume that all values/sizes are integer. In homework 1, we derived a polynomial time approximation scheme (PTAS). Using dynamic programming, it is possible to derive a fully polynomial time approximation scheme (FPTAS). That is, for any $\epsilon > 0$, we can get a $(1 - \epsilon)$ approximation in time polynomial in N and $\frac{1}{\epsilon}$, where $N = n \times \log \max_{i\in[n]}\{v_i, s_i\}$.

It will be shown that the knapsack problem can be solved exactly using dynamic programming. Consider the related problem where given a target u, we want to find the minimum size subset of total value at least u.

That is, given a value u, we want to solve the following optimization problem:

$$\begin{aligned} &\min_{S\subseteq[n]} \sum_{i\in S} s_i && (KS^*) \\ &\text{s.t.} \sum_{i\in S} v_i \geq u \end{aligned}$$

It turns out that (KS^*) can be solved exactly using dynamic programming. Let $T[i, w]$ denote the minimum size of subset $S \subseteq \{1, ..., i\}$ such that $\sum_{k\in S} v_k = w$.

Algorithm 4 Dynamic programming algorithm for (KS^*)

Let $v_{max} = \max_{i\in[n]} v_i$.

1. For $w = 0, ..., nv_{max}$:
   Set $T[1, w] = \begin{cases} 0 & w = 0 \\ s_1 & w = v_1 \\ \infty & \text{otherwise.} \end{cases}$
2. For $i = 2, ..., n$:
   For $w = 0, ..., nv_{max}$:
   $\rightarrow$ Set $T[i, w] = \min\{T[i - 1, w], T[i - 1, w - v_i] + s_i\}$.

Finally, to find the optimal solution to the knapsack problem, it suffices to find

$$\arg\max_w \{T[n, w] : T[n, w] \leq B\}.$$

Using Algorithm 4, we have shown the following result:

Theorem 2.1 *Assuming that all values are integer, there exists an exact algorithm for the knapsack problem in time that is polynomial in n and v_{max}.*

Note that the knapsack problem is NP-hard! So we cannot expect an exact algorithm in polynomial of n and $\log v_{max}$. We will use this DP to obtain an FPTAS in the next lecture.

References

[1] V. Arya, N. Garg, R. Khandekar, a. Meyerson, K. Munagala, and V. Pandit. Local search heuristic for k-median and facility location problems. *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, page 29, 2001.