

CSE 203A — Advanced Algorithms — Scribe Notes
UCSD — Winter Quarter 2013
Instructor: Russell Impagliazzo
Notes by: David Lrant¹
Thursday, January 17, 2013

Set Cover

Recall from last week the Set Cover problem in which you have a universe of n elements U , and m sets of size at most k containing elements from U . The goal is to pick as few S_j as possible such that $\bigcup_{picked} S_j = U$. We came up with the following simple greedy algorithm: Find the set S_i that covers the most new elements (let this number be r). Pick this set, and assign all the newly covered elements weight $1/r$

Lemma 1. $W = \sum w(x) = |SetCover|$

Lemma 2. For every set S_i , $\sum_{x \in S_i} w(x) \leq \sum_{i=1}^k 1/i = H_k$ ²

This is because there are at most r nodes in S_i of weight $1/r$.

Lemma 3. r only decreases throughout the Algorithm.

So all elements in S_i that eventually get weight $\geq 1/r$ must still be uncovered when the maximum # of uncovered elements reaches r . This is because if there were $> r$ at this point, S_i would have $> r$ uncovered elements which contradicts Lemma 2.

In particular, if we let OPT be the optimal set cover, and let $l = |OPT|$

$$H_k \cdot l \geq \sum_{S \in OPT} \sum_{x \in S} w(x) \geq \sum w(x) = |\text{our set cover}|$$

so our algorithm is within H_k factor of optimal.

¹Based on handwritten class notes by Michael Goldberg

²Note this is also true for sets not in the cover

Why Weights?

Let y_j be indicator variables, valued 1 if S_j is in the set cover. Then for set elements $x_1, \dots, x_n$, that element is covered when $w_i(\sum_{x_i \in S_j} y_j) \geq 1$, and we want to minimize y_i .

We want values to be either 0 or 1, but the equation make sense for any real number (in particular between 0 and 1).

$$\begin{array}{rcccccccl} \text{Ex:} & y_1 & + & y_2 & + & y_3 & & \geq & 1 \\ & & & y_2 & + & y_3 & + & y_4 & \geq & 1 \\ + & y_1 & & & & & + & y_4 & \geq & 1 \\ \hline & 2y_1 & + & 2y_2 & + & 2y_3 & + & 2y_4 & \geq & 3 \end{array}$$

Therefore $\sum y_i \geq 1.5 \Rightarrow \sum y_i \geq 2$ since y_i are integers.

Now consider relaxation of linear system in which variables are not required to be integer. This analysis will imply that H_k is actually the ratio between our solution and the optimal non-integer solution. I.e. H_k is the price we pay for requiring integer solutions.

Transformation Method

For Optimality show $\text{Cost}(\text{opt}) \geq \text{cost}(\text{greedy})$.

Consider the solution as a sequence of moves, each of which contributes some additive local cost For example

$$\text{Sol}'n = \text{Move}_1 + \text{Move}_2 + \dots + \text{Move}_n \text{ and } \text{Cost}(\text{Sol}'n) = \text{Cost}_1 + \text{Cost}_2 + \dots + \text{Cost}_n$$

Now say that

$$\text{Opt}_0 = \text{MO}_1 + \text{MO}_2 + \dots + \text{MO}_k \text{ and } \text{Greedy} = \text{MG}_1 + \text{MG}_2 + \dots + \text{MG}_j$$

Create a new optimal solution Opt_1 where

$$\text{Opt}_1 = \text{MG}_1 + \text{MO}'_2 + \text{MO}'_3 + \dots + \text{MO}'_{k'}$$

such that $\Delta_1 = \text{Cost}(\text{Opt}_1) - \text{Cost}(\text{Opt}_0) \leq 0$ From that create a new optimal solution Opt_2 where

$$\text{Opt}_2 = \text{MG}_1 + \text{MG}_2 + \text{MO}''_3 + \dots + \text{MO}''_{k''}$$

such that $\Delta_2 = \text{Cost}(\text{Opt}_2) - \text{Cost}(\text{Opt}_1) \leq 0$ and so on.

–To convert this proof schema to studying approximation algorithms, we relax the constraint on the Δ_i . In other words $\Delta_i \leq \rho \cdot \text{Cost}(MO_i)$ This will give you a $(1 + \rho)$ -approximation.

This is similar (but different!) for maximization problems, as we will see now.

Interval Scheduling

Instance:

- n jobs $(s_i, f_i) = (\text{start time}, \text{finish time})$
- m identical machines

Solution:

- An assignment of (some) jobs to machines satisfying constraint: ≤ 1 job on any machine at any time.

Maximize:

- The number of assigned jobs (unweighted)

Greedy Heuristics:

- (1) Schedule the smallest duration job first, inn the first machine where it fits
- (2) Schedule earliest finish time first on the first machine where it fits. (Note this is optimal on the 1 machine case)

Consider heuristic (1) first. and consider schedule as choosing one job (move) at a time.

Let's compare the greedy and optimal solution by transforming the optimal solution to the greedy and keeping track of how much the size changed.

Since there can be at most two jobs conflicting on a single machine, the worst cases are subtract 2 add 1, or subtract 3 add 1 if we are moving a job that currently placed elsewhere.

Lemma 4. *Heuristic (1) gives you a $1/3$ approximation*

Proof. Let $j_1, \dots, j_i$ be the first i jobs scheduled by smallest duration. Then $\exists$ a schedule OPT_i that contains $j_i, \dots, j_i$ on the same machines as greedy does such that $|OPT_i| \geq |OPT| - 2i$. In particular, let l be the total # of jobs schedules by Shortest Duration. $l \geq |OPT| - 2l$, so $l \geq 1/3|OPT|$. Therefore this is a $1/3$ approximation. $\square$

Lemma 5. *Heuristic (2) gives you a $1/2$ approximation*

Proof. Consider the same setup (converting OPT to Greedy).

If Greedy job conflicts with > 1 job, then at least one ends before the greedy job (but we still have to account for potentially moving one) The argument is identical, but with $|opt_i| \geq |OPT| - i$ which gives an approximation factor of $1/2$. $\square$