Approximation Algorithms: Supplementary notes for Set Cover and Knapsack

Kristoffer Arnsfelt Hansen

May 14, 2012

1 The set covering problem

Recall that the set covering problem is the following optimization problem.

Set Cover
Instance: Universe X, $n = \|X\|$. Family $\mathcal{F}$ of subsets of X, such that $\cup_{S \in \mathcal{F}} S = X$. **Objective:** Find $\mathcal{C} \subseteq \mathcal{F}$ that minimizes $\|\mathcal{C}\|$ and satisfies $\cup_{S \in \mathcal{C}} S = X$.

This problem is a generalization of the node covering problem.

Node Cover
Instance: Graph $G = (V, E)$, $n = \|V\|$. **Objective:** Find $\mathcal{C} \subseteq V$ that minimizes $\|\mathcal{C}\|$ and satisfies that for all $(u, v) \in E$ either $u \in \mathcal{C}$ or $v \in \mathcal{C}$.

To see this, given an instance $G = (V, E)$ of the Node Cover problem, we can define a corresponding Set Cover instance as follows.

- $X := E$.
- $\mathcal{F} := \{S_w \mid w \in V\}$, where $S_w = \{(u, v) \in E \mid u = w\}$.

1.1 Greedy approximation algorithm

We will study the following approximation algorithm for the set covering problem, that build a set cover by greedily choosing the next set to add that will cover the most new elements.

Input: Universe X, Family $\mathcal{F}$ of subsets of X.
Output: Set cover $\mathcal{C}$.

$U \leftarrow X$
$\mathcal{C} \leftarrow \emptyset$
while $U \neq \emptyset$ **do**
  pick $S \in \mathcal{F}$ that maximizes $|S \cap U|$
  $U \leftarrow U \setminus S$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{S\}$
end while
return $\mathcal{C}$

Algorithm 1: Approximation algorithm for Set Cover.

1.1.1 Quick analysis

Theorem 1 *Algorithm 1 is a polynomial time approximation algorithm for* Set Cover *with approximation ratio* $\ln(n)$.

Proof Clearly the algorithm is a polynomial time algorithm, returning a valid set cover; We next give a proof of the approximation ratio.

Let $\mathcal{C}$ be the set cover returned by the algorithm, and let $\mathcal{C}^*$ be an optimal set cover, $m = |\mathcal{C}^*|$. We will prove that after at most $m\ln(n)$ iterations, the algorithm terminates. Since each iteration adds exactly one set to the cover we will have $|\mathcal{C}| \leq m\ln(n)$, which establishes the claimed ratio.

Note that after each iteration i, since $\mathcal{C}^*$ is a set cover of the elements in U, one of the sets in $\mathcal{C}^*$ covers at least a fraction $1/m$ of the elements in U. Since the algorithm picks the set S_i that cover the most new elements, picking S_i must also cover at least a fraction $1/m$ of the elements in U.

We can conclude that after (at most) $m\ln(n)$ iterations, the number of remaining elements is less than

$$n\left(1-\frac{1}{m}\right)^{m\ln(n)} < n\left(e^{-\frac{1}{m}}\right)^{m\ln(n)} = ne^{-\ln(n)} = 1 \ ,$$

using Lemma 7. Thus in fact all elements are covered. □

1.1.2 Refined analysis

Definition 2 *The kth Harmonic number H_k is defined as*

$$H_k = 1 + \frac{1}{2} + \cdots + \frac{1}{k} \ .$$

We remark that for all k we have $\ln(k) \leq H_k \leq \ln(k) + 1$, see Lemma 6.

Theorem 3 *Algorithm 1 is a polynomial time approximation algorithm for* Set Cover *with approximation ratio H_k, where $k = \max_{S\in\mathcal{F}} |S|$.*

Proof As noted in the previous proof, it is clear that the algorithm is a polynomial time algorithm, returning a valid set cover; We next give a proof of the approximation ratio.

Let $\mathcal{C}$ be the set cover returned by the algorithm. Let $S_1, S_2, \ldots, S_m$ be the sequence of sets added to $\mathcal{C}$ by the algorithm. We will distribute the cost of adding a new set S_i to the cover $\mathcal{C}$ evenly over all new-covered elements.

Assume that element $x \in U$ is covered for the first time in iteration i of the algorithm. We then define the cost of adding x,

$$c_x = \frac{1}{|S_i \setminus (S_1 \cup \cdots \cup S_{i-1})|} \ .$$

Since at every iteration of the algorithm 1 unit of cost is distributed we have

$$|\mathcal{C}| = \sum_{x \in X} c_x \ .$$

Now, let $\mathcal{C}^*$ be an optimal set cover. Since $\mathcal{C}^*$ is a set cover, i.e $\cup_{S \in \mathcal{C}^*} S = X$, we have

$$\sum_{x \in X} c_x \leq \sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x \ ,$$

and by combining these we have

$$|C| \leq \sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x \ .$$

Let $T \in \mathcal{F}$ be any set of the family $\mathcal{F}$. Write the elements

$$T = \{y_1, y_2, \ldots, y_k\}$$

such that $(y_1, y_2, \ldots, y_k)$ is the *reverse* order of when the elements $y_1, \ldots, y_k$ are covered by the algorithm.

The central observation is the following: at the moment y_j is in fact covered by the algorithm by set S_i, the set T has at least j elements of X that are not yet covered! Since the algorithm picks the set S_i that cover the most new elements, picking S_i must cover at least j elements (as otherwise the algorithm would have picked T over S_i). In other words we have

$$|S_i \setminus (S_1 \cup \cdots \cup S_{i-1})| \geq j \ ,$$

and it follows

$$c_{y_j} = \frac{1}{|S_i \setminus (S_1 \cup \cdots \cup S_{i-1})|} \leq \frac{1}{j} \ .$$

Hence

$$\sum_{x \in T} c_x \leq \sum_{j=1}^{k} \frac{1}{j} = H_k \ .$$

We can now conclude

$$|C| \leq \sum_{S \in \mathcal{C}^*} \left(\sum_{x \in S} c_x \right) \leq \sum_{S \in \mathcal{C}^*} H_{|S|} \leq |\mathcal{C}^*| \cdot \max_{S \in \mathcal{C}^*} H_{|S|} \leq |\mathcal{C}^*| \cdot \max_{S \in \mathcal{F}} H_{|S|} \ ,$$

and therefore

$$\frac{|C|}{|C^*|} \leq \max_{S \in \mathcal{F}} H_{|S|}$$

□

As we noted, for all k we have $\ln(k) \leq H_k \leq \ln(k)+1$. In particular, since $\max_{S \in \mathcal{F}} |S| \leq n$ the algorithm guarantees approximation ratio $\ln(n)+1$, by Lemma 6. (Actually, if $\max_{S \in \mathcal{F}} |S| = n$, the algorithm finds the optimal solution. Hence we can assume $\max_{S \in \mathcal{F}} |S| < n$, and the refined analysis thus shows an approximation ratio $\ln(n)$ like the quick analysis).

Observation 4 *Specializing to* VERTEX COVER *gives*

$$\frac{|C|}{|C^*|} \leq \max_{v \in V} H_{\deg(\text{v})} \ .$$

When the degree of G is at most 3 we have an approximation algorithm with approximation ratio $H_3 = \frac{11}{6} < 2$.

2 The knapsack problem

Recall that the knapsack problem is the following optimization problem.

KNAPSACK

Instance: Weights $w_1, \ldots, w_n$, values $v_1, \ldots, v_n$ and weight limit W.
Objective: Find $S \subseteq \{1, \ldots, n\}$ that maximizes $\sum_{i \in S} v_i$ and satisfies $\sum_{i \in S} w_i \leq W$.

Let $V = \max v_i$. In the tutorials we have seen a pseudo-polynomial time algorithm for KNAPSACK using dynamic programming, running in time $O(n^2 V)$.

2.1 A fully polynomial time approximation scheme

We will study the following approximation algorithm for the knapsack problem, that given an additional input $\epsilon > 0$ *"rounds"* the values and computes an approximate solutioin by solving the rounded instance to optimality.

Input: Weight $w_1, \ldots, w_n$, values $v_1, \ldots, v_n$, weight limit W and parameter $\epsilon > 0$.
Output: Selection of items, $S' \subseteq \{1, \ldots, n\}$.

1: $V \leftarrow \max\{v_i \mid w_i \leq W\}$
2: $B \leftarrow \epsilon V / n$
3: $v'_i \leftarrow \lfloor v_i / B \rfloor$
4: Compute optimal solution S' using weights $w_1, \ldots, w_n$, new values $v'_1, \ldots, v'_n$ and weight limit W, by the dynamic programming algorithm.
5: **return** S'

Algorithm 2: Approximation algorithm for KNAPSACK.

Theorem 5 *Algorithm 2 computes in time $O(n^3/\epsilon)$ a solution that obtains a solution with value at least a $(1 - \epsilon)$ fraction of the optimum value.*

Proof Since S' is computed for the same weights and weight limit as the input instance, it is a valid solution. The running time is $O(n^2 V (n/\epsilon V)) = O(n^3/\epsilon)$ as stated.

Let S' be the solution returned by the algorithm and let S be an optimal solution. Thus the value of the optimal solution S is $\sum_{i \in S} v_i$ while the value of the approximate solution is $\sum_{i \in S'} v_i$. We next establish a number of inequalities.

Since we are rounding down after dividing, we have

$$\sum_{i \in S'} v_i \geq B \sum_{i \in S'} v'_i \ .$$

Since S' is an optimal solution for values $v'_1, \ldots, v'_n$, we have

$$B \sum_{i \in S'} v'_i \geq B \sum_{i \in S} v'_i \ .$$

Since rounding a rational down to an integer descreases the number by at most 1, we have

$$B\sum_{i\in S} v_i' \geq B\sum_{i\in S}(v_i/B-1) = \sum_{i\in S}(v_i - B) \geq (\sum_{i\in S} v_i) - nB \ .$$

Furthermore, since $V \leq \sum_{i\in S} v_i$ and $nB = \epsilon V$ we have

$$nB \leq \epsilon \sum_{i\in S} v_i \ .$$

Combining these four inequalities we obtain

$$\sum_{i\in S'} v_i \geq (1-\epsilon)\sum_{i\in S} v_i \ ,$$

as claimed. □

Note: As we have defined it, the approximation ratio the above analysis guarantees is $\frac{1}{1-\epsilon}$. Thus to obtain an approximation ratio of $1+\epsilon'$ for a given $\epsilon' > 0$, we can give $\epsilon := \epsilon'/(1+\epsilon')$ as input to algorithm 2. This also gives $\frac{1}{\epsilon} = (1+\epsilon')/\epsilon' = O(\frac{1}{\epsilon'})$. Thus algorithm 2 is also an approximation algorithm with approximation ratio $1+\epsilon'$ running in time $O(n^3/\epsilon')$.

A Mathematical Preliminaries

Remark: The proofs of the statements that follow are not part of the curriculum.

A.1 Harmonic numbers

We have the following simple bound on the Harmonic numbers.

Lemma 6

$$\ln(k) \leq H_k \leq \ln(k) + 1 \ .$$

Proof Compare with the integral $\int_1^x \frac{1}{y}\,dy = \ln(x)$, see Figure 1. □

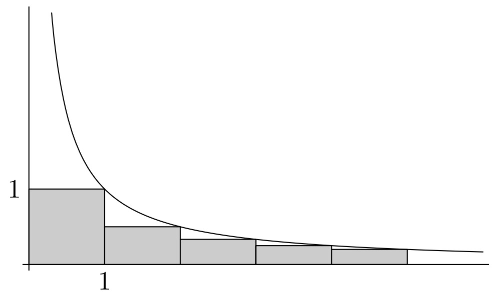

Figure 1: Graph of the function $f(y) = \frac{1}{y}$.

A.2 An inequality for the exponential function

Lemma 7 *For all $x \neq 0$ it holds that $1 + x < e^x$.*

Proof Define $f(x) = e^x - (1+x)$. Then $f'(x) = e^x - 1$ and $f''(x) = e^x$. Since $f''(x) > 0$ for all x, the function f is strictly convex, and hence assumes its minimum at its only critical point, $x = 0$. Since $f(0) = 0$, $f(x) > 0$ for all $x \neq 0$. □