EDAN55 Kurssammanfattning

Contents

1 Set Cover (with weights)

Find the collection such that the union of its sets are equal to all elements in U. In weighted version, every set S_i has an associated weight $w_i \geq 0$.

Example:
U = $\{S_1, S_2, S_3, S_4\}$ = {{1,2}, {2}, {2,3}, {3,4,5}}
$S_1 \cup S_4 = \{1, 2, 3, 4, 5\}$ = All elements of U = Set Cover of U.

Goal: Find set cover C such that $\sum_{S_i \in C} w_i$ is minimised.

Maintain a set R of all remaining uncovered elements.

$\frac{w_i}{|S_i \cap R|}$ = cost for covering remaining elements in S_i.

Algorithm: (Greedy-Set-Cover)
R = U
While $R \neq \emptyset$
Select S_i which minimises $\frac{w_i}{|S_i \cap R|}$.
Delete all $s \in S_i$ from R.
End
Return selected sets

Example: (bad instance)
$S_1 = \{1, 2, 3, 4\}, w_1 = 1 + \epsilon$ (ϵ small number > 0)
$S_2 = \{5, 6, 7, 8\}, w_2 = 1 + \epsilon$
$S_3 = \{3, 4, 7, 8\}, w_3 = 1$
$S_4 = \{2, 6\}, w_4 = 1$
$S_5 = \{1\}, w_5 = 1$
$S_6 = \{5\}, w_6 = 1$

Optimal solution: $2 + 2\epsilon$ $\{S_1, S_2\}$.

1. Picks S_3 instead of S_1 or S_2 since $\frac{1}{4} < \frac{1+\epsilon}{4}$.
2. Picks S_4 instead of S_1 or S_2 since $\frac{1}{2} < \frac{1+\epsilon}{2}$
2. Picks S_5 or S_6 instead of S_1 or S_2 since $\frac{1}{1} < \frac{1+\epsilon}{1}$
Finds solution: $1 + 1 + \frac{1}{2} + \frac{1}{4} = 2.75 > 2 + 2\epsilon$.
This example can be expanded in the same way to construct an arbitrarily large instance that will perform just as bad.

Record cost:
$c_s := \frac{w_i}{S_i \cap R}$ for every $s \in S_i \cap R$
Does not change the algorithm, used only for analyse.

Example:
$S_1 = \{1, 3\}, w_1 = 1$
$S_2 = \{2, 3, 4\}, w_1 = 1$
1. Pick S_2 of cost $\frac{1}{3} \Rightarrow c_2 = c_3 = c_4 = \frac{1}{3}$
2. Pick S_1 of cost $\frac{1}{1} \Rightarrow c_1 = 1, (c_3 = \frac{1}{3}$, unchanged)

The costs completely account for the total weight of the set cover.

(11.9) If C is the set cover obtained by the Greedy-Set-Cover, then $\sum_{S_i \in C} W_i = \sum_{s \in U} c_s$.

Example: (continuation of previous)
$\sum_{S_i \in C} W_i = w_1 + w_2 = 1 + 1 = 2$
$\sum_{s \in U} c_s = c_1 + c_2 + c_3 + c_4 = \frac{1}{3} + \frac{1}{3} + \frac{1}{3} + 1 = 2$
$\Rightarrow \sum_{S_i \in C} W_i = \sum_{s \in U} c_s$

Q: How much cost can any single set S_k account for, including sets not picked by the algorithm? Find upper bound for the ratio $\frac{\sum_{s \in S_k} c_s}{w_k}$.

The optimum solution must cover the full cost $\sum_{s \in U} c_s$ via the sets it selects so this bound will establish that it needs to use at least a certain amount of weight (= lower bound, what we want).

Harmonic function:

$H(n) = \sum_{i=1}^{n} \frac{1}{i}$

Sum approximates the area under the curve $y = \frac{1}{x}$.

Naturally bounded above by $1 + \int_1^n \frac{1}{x} dx = 1 + ln(n)$ and

below by $\int_1^{n+1} \frac{1}{x} dx = ln(n+1)$.

Thus $H(n) = \Theta(ln(n))$

(11.10) For every set S_k, the sum $\sum_{s \in S_k} c_s$ is at most $H(|S_k|) * w_k$.

Proof. Simplify notation by assume that the elements of S_k are the first $d = |S_k|$ elements of the set U ($S_k = \{s_1, \ldots, s_d\}$).

Example:

$U = \{a, b, c, d, e, f, g\}$

$S_1 = \{a, b\}, d = |S_1| = 2$

$S_2 = \{a, b, c, d, e\}, d = |S_2| = 5$

Assume that the elements are labeled in the order in which they are assigned a cost c_{s_j} by the greedy algorithm (ties broken arbitrarily). No loss of generality, only involves renaming elements in U.

Consider the iteration when s_j is covered by the greedy algorithm for some $j \le d$. When the iteration begins, $s_j, s_{j+1}, \ldots, s_d \in R$, according to our naming convention (elements have not been selected). This implies that $|S_k \cap R| \ge d - j + 1$ (there are $d - j + 1$ not already selected elements in S_k). The average cost of the set S_k is at most $\frac{w_k}{|S_k \cap R|} \le \frac{w_k}{d-j+1}$.

It is not necessarily an equality because in the same iteration as s_j is covered by the greedy algorithm, some other elements $s_{j'}$ for $j' < j$ may be covered as well.

In this iteration, the greedy algorithm selects a set S_i of a minimum average cost so that the set S_i has an average cost at most that of S_k. The average cost of S_i gets assigned to s_j, so

$c_{s_j} = \frac{w_i}{|S_i \cap R|} \le \frac{w_k}{|S_k \cap R|} \le \frac{w_k}{d-j+1}$.

Add up all inequalities for all elements $s \in S_k$:

$\sum_{s \in S_k} c_s = \sum_{j=1}^{d} c_{s_j} \le \sum_{j=1}^{d} \frac{w_k}{d-j+1} = \frac{w_k}{d} + \frac{w_k}{d-1} + \ldots + \frac{w_k}{1} = H(d) * w_k$

Let $d* = max_i |S_i|$ denote the maximum size of any set.

(11.11) The set cover C selected by the Greedy-Set-Cover has weight at most H(d*) times the optimal weight w*.

Proof. Let C* denote the optimum weight set cover, so that $w* = \sum_{S_i \in C*} w_i$. For each of the sets in C*, (11.10) implies

$w_i \ge \frac{1}{H(d*)} \sum_{s \in S_i} c_s$

(The cost has to be paid by the weight). Because these sets form a set cover, we have

$\sum_{S_i \in C*} \sum_{s \in S_i} c_s \ge \sum_{s \in U} c_s$

(The same element can be present several times in C* but not in U, there for no equality). Combining these with (11.9) gives the desired bound:

$w* = \sum_{S_i \in C*} w_i \ge \sum_{S_i \in C*} \frac{1}{H(d*)} \sum_{s \in S_i} c_s \ge \frac{1}{H(d*)} \sum_{s \in U} c_s = \frac{1}{H(d*)} \sum_{S_i \in C} w_i$

The greedy algorithm finds a solution within a factor $O(log(d*)$ of the optimal. Since the maximum set size $d*$ can be a constant fraction of the total number of elements n, this is a worst-case upper bound of $O(log(n))$. By expressing the bounds in terms of $d*$ shows us that we're doing much better if the largest set is small.

It has been shown that no polynomial-time approximation algorithm can achieve an approximation bound much better than $H(n)$, unless P = NP.

2 The Pricing Method: Vertex Cover

Vertex cover in a graph $G = (V, E)$ is a set $S \subseteq V$ so that each edge has at least one end in S. In this version of the problem, each vertex $i \in V$ has a weight $w_i \ge 0$, with the weight of a set S of vertices denoted $w(S) = \sum_{i \in S} w_i$.

Goal: find a vertex cover S for which w(S) is minimised.

(When all weights are equal to 1, deciding if there is a vertex cover of weight at most k is the standard decision version of Vertex Cover).

Vertex Cover $\le_P$ Set Cover

If we had a polynomial-time algorithm that solves the Set Cover Proble, then we could use this algorithm to solve the Vertex Cover Problem in polynomial time.

(11.12) The Set Cover approximation algorithm can be used to give an $H(d)$-approximation algorithm for the weight Vertex Cover Problem, where d is the maximum degree of the graph. (The degree of the graph is the maximum number edges attached to a vertex).