Advanced Algorithms Course. Lecture Notes. Part 2

Weighted Set Cover

This is a very fundamental problem abstracted from a variety of applications. Given a set U of n elements, and m subsets $S_i \subseteq U$ with positive weights w_i, find a set cover with minimum total weight. A **set cover** is a selection from the given sets S_i whose union is still the whole of U.

The Set Cover problem is NP-complete, as it generalizes the Vertex Cover problem. You should be able to give a polynomial-time reduction from Vertex Cover.

A natural greedy rule is to successively add sets S_i to the solution, that cover as many *new* elements as possible per unit of weight. More formally: Let R denote the set of yet uncovered elements, initially $R := U$. In every step we put some S_i with minimal $w_i/|S_i \cap R|$ in the solution, and we update R by $R := R \setminus S_i$. This is repeated until $R = \emptyset$.

It may be good to reflect why this greedy rule is proposed, rather than simpler rules: Why don't we choose the cheapest set each time? Why don't we choose the largest set each time? (What could go wrong?) Once you see the reasons why the ratio of weight and "coverage" is a much better criterion, you could also be wondering: Why don't we choose some S_i with minimal $w_i/|S_i|$ each time? That is, why is R also taken into account?

After these heuristic questions you may feel comfortable with the proposed algorithm, and we can go on and analyze its approximation ratio.

In the following analysis we will need the so-called Harmonic sum. It is defined as $H(n) := \sum_{i=1}^{n} 1/i$. Asymptotically it behaves roughly as $\ln n$. (But note that $H(n)$ is quite different from $\ln n$ for small n.) The reason for using $H(n)$ is not obvious at this point; it is just more convenient to define it already now, so we can focus on the actual analysis.

Let C be the greedy solution, and C^* an optimal set cover, with total weight w^*. The natural the algorithm is, deriving a good bound for its

approximation ratio is not so trivial. The concern is: How should we be able to compare the costs of two (possibly) totally different set families?

The key idea of the analysis is that both solutions must cover the same elements, namely all elements of U. Therefore we "charge" the elements for being covered, and then we can compare their "payments" element-wise! We only have to make sure that the prices correspond to the real costs. In detail:

Whenever the elgorithm selects a set S_i, every element $s \in S_i \cap R$ pays an amount $c_s := w_i/|S_i \cap R|$. In words: The cost w_i for the step is shared equally among all newly covered elements.

The weight of the greedy solution C obviously equals the sum of these costs: $\sum_{s\in U} c_s = \sum_{S_i\in C} w_i$. Now the somewhat tricky part of the analysis begins.

Consider any set $S_k = \{s_1, \ldots, s_d\}$, where the elements of S_k are sorted in the order they are covered by the greedy algorithm. The index d is just the size of this set. We do not know whether S_k will be chosen in a solution, but regardless of that we may study how much is paid by the *elements* of S_k. Only later we will use this information for comparing the solutions.

Next consider any element $s_j \in S_k$. Just before s_j gets covered by the greedy solution, we have $|S_k \cap R| \geq d - j + 1$, simply by the definition of R. Hence $w_k/|S_k \cap R| \leq w_k/(d - j + 1)$. (Remember that our prices look like this, therefore this step.) Let S_i be the set that covers this element s_j in the greedy solution. Since the algorithm always picks a set S_i with minimum weight-per-element ratio, this means

$$w_i/|S_i \cap R| \leq w_k/|S_k \cap R| \leq w_k/(d - j + 1).$$

Summation of all element costs in S_k now yields $\sum_{s\in S_k} c_s \leq H(|S_k|)w_k$. Note carefully how the Harmonic sum comes in: recall what c_s was, and pay attention to the summation index.

So far we have studied any single set S_k of size d. Now we re-define d to be the *maximum* size of all sets S_i. Then the previous inequality is still true: $H(d)w_k \geq \sum_{s\in S_k} c_s$ for each k. We also have $\sum_{S_k\in C^*} \sum_{s\in S_k} c_s \geq \sum_{s\in U} c_s$, simply because C^* is a set cover. Finally we can put things together:

$$H(d)w^* = H(d) \sum_{S_k\in C^*} w_k \geq \sum_{S_k\in C^*} \sum_{s\in S_k} c_s \geq \sum_{s\in U} c_s = \sum_{S_k\in C} w_k.$$

This shows that the algorithm has an approximation ratio $H(d) \approx \ln d$.

It might be disappointing that the approximation ratio is not constant but grows with d. However, it grows only logarithmically, and it is constant when the size d is fixed (a frequent case in applications). The ratio $H(d)$ is also the best possible one for any polynomial-time Set Cover algorithm. The latter fact is very hard to prove. Such hardness-of-approximation results are far beyond the reach of this course. We only mention this fact for your information.

Do not be afraid of the many specific technical details of this analysis. The take-home message is the basic idea that was used: to assign pieces of the costs of both solutions to each other, in such a way that they can be related to each other, one by one. Specifically, we have moved the costs from the sets to the elements, because the elements are the same in both solutions. For other problems we may have to assign the costs in other ways.

Weighted Vertex Cover – The Pricing Method

We are given a graph $G = (V, E)$, where we index the nodes by integers, that is, $V = \{1, \ldots, n\}$, and node i has a weight w_i. The problem is to find a vertex cover of minimum weight. This problem is more general than the unweighted Vertex Cover problem but is a special case of Weighted Set Cover. (Think a moment: What are the elememts to be covered, and what are the sets?)

Thus we can apply the previous $H(d)$-approximation, where the maximum node degree takes over the role of d. (Why?) But, luckily, we can obtain a better approximation ratio. It will be the constant 2. This is not only a nice result as such, but also the method we present is of more general relevance in Optimization. Again we use prices, but now already in the algorithm itself, not only in its analysis. The technique is called the pricing method, or primal-dual method, because the given "primal" problem is attacked by means of some "dual" problem (see below).

We let every edge e pay a price $p_e \geq 0$ for being covered by some node. We will fix these prices later. The prices are called *fair* if $\sum_{e=(i,j)} p_e \leq w_i$ holds for all nodes i. That is, the payments of all edges incident to node i must not exceed the weight of i.

Terminology remark: Strictly speaking, the phrase "the prices are fair" is not accurate. We only use it for brevity. Fairness is a property of the whole function that assigns prices p_e to all edges $e \in E$, not a property of its single values p_e.

For fair prices it follows $\sum_{i \in S} \sum_{e=(i,j)} p_e \leq w(S)$ for every subset S of nodes. In particular, if S is a vertex cover, then every edge appears at least once in this sum, thus $\sum_{e \in E} p_e \leq \sum_{i \in S} \sum_{e=(i,j)} p_e \leq w(S)$. This inequality says that the sum of *any* fair prices is a lower bound on the cost of any vertex cover, thus even for the cost of an optimal vertex cover.

While the derivation of this inequality was only simple combinatorics, it plays a fundamental role: Instead of tackling the problem directly, we may first construct prices that are fair but as large as possible (this is going to be our "dual" problem) and then construct in an ad-hoc way a cheap vertex cover from these fair prices. In fact, this is easier to do than one might expect:

We call a node i *tight* if $\sum_{e=(i,j)} p_e = w_i$. Initially let all $p_e = 0$. Now we take some e without tight endnodes and simply raise p_e until one endnode is tight. This step is repeated as long as possible. After that, let S be the set of tight nodes.

This was the algorithm! We skip a time analysis, but it is easy to see that the algorithm terminates (since every step produces a new tight node) and the time is polynomial.

Clearly, S is a vertex cover, otherwise the algorithm could do more steps. Moreover, $\sum_{e=(i,j)} p_e = w_i$ holds for all $i \in S$, by definition of S. Summation over all $i \in S$ yields $\sum_{i \in S} \sum_{e=(i,j)} p_e = w(S)$. Every edge e appears at most twice in this sum, hence $w(S) \leq 2 \sum_{e \in E} p_e$. This shows that $w(S)$ has at most twice the weight of an optimal vertex cover.

In the unweighted case, when $w_i = 1$ for all nodes i, the edges with positive prices $p_e = 1$ form a maximal matching in the graph, and the vertex cover consists of all nodes contained in the edges of this matching. That is, the pricing method yields the algorithm for this special case that we have already seen.