CS 683: Advanced Algorithms	Set Cover with Finite VC-Dimension
April 12, 2001	Scribe: Tim Roughgarden

1 Set Cover and Hitting Set

Today, we study an application of VC-dimension and ϵ-nets to a well-studied NP-hard problem.

Set Cover:

- *Input:* A ground set X of elements, and a collection $\mathcal{C}$ of subsets of X.
- *Goal:* Find the smallest subcollection $\mathcal{C}' \subseteq \mathcal{C}$ whose union is X.

On one hand, it may seem like the approximability of Set Cover is completely resolved: it is known that the greedy algorithm[1] gives an $O(\log n)$ approximation for the problem, and that no better approximation is possible unless P=NP. On the other hand, we can still hope for stronger positive results if we restrict our attention to "sufficiently simple" set systems. In particular, it is natural to ask for an $o(\log n)$-approximation algorithm for Set Cover instances with bounded VC-dimension.

Before giving an approximation algorithm for a special case of an NP-hard problem, one should of course check that the special case is itself NP-hard. For our problem, we need only note that the Set Cover instances arising from the NP-hard problem Vertex Cover[2] have bounded VC-dimension. To this end, consider any undirected graph $G = (V, E)$; vertex covers of G correspond to set covers in the set system $(X, \mathcal{C})$, where $X = E$ and $\mathcal{C}$ is the collection of stars in G (where a star is the set of edges incident to a single vertex).

Claim 1.1 *The VC-dimension of the above set system $(X, \mathcal{C})$ is at most 2.*

Proof. Suppose for contradiction that a set $F = \{e_1, e_2, e_3\} \subseteq E$ of three edges of G is shattered by $\mathcal{C}$. Note that F is shattered if and only if for any subset $F' \subseteq F$, we can find a star $S \in \mathcal{C}$ such that $F \cap S = F'$. Taking $F' = F$, we deduce that e_1, e_2, e_3 must share a common endpoint v (then, if S_v is the star around v, we have $F \cap S_v = F$); write $e_i = (v, w_i)$ for $i = 1, 2, 3$. Then, for any star $S \in \mathcal{C}$, we have either $|S \cap F| = 3$ (if S is the star around v), $|S \cap F| = 1$ (if S is the star around some w_i), or $|S \cap F| = 0$ (otherwise). Taking F' to be any two-element subset of F, there is no star $S \in \mathcal{C}$ with $F \cap S = F'$; this contradicts the assumption that F is shattered by $\mathcal{C}$. ■

[1]Iteratively pick the set covering as many as-yet-uncovered elements as possible, until you have a feasible set cover.

[2]Given an undirected graph G, find the smallest set of vertices that meets every edge.

Having assured ourselves that the goal of finding an approximation algorithm for Set Cover with bounded VC-dimension is a reasonable one, we will next reduce Set Cover to the equivalent Hitting Set problem, as the latter will be more convenient for us.

Hitting Set:

- *Input:* A ground set X of elements, and a collection $\mathcal{C}$ of subsets of X.
- *Goal:* Find the smallest subset $H \subseteq X$ of elements that hits every set of $\mathcal{C}$ – i.e., for which $H \cap S \neq \emptyset$ for every $S \in \mathcal{C}$.

Comparing the Set Cover and Hitting Set problems, the reader can hopefully sense that some type of duality is afoot. To make this precise, we will show that for any instance $(X, \mathcal{C})$ of Set Cover, there is an equivalent instance (that we will call the *dual set system*) $(X', \mathcal{C}')$ of Hitting Set, as follows. Write $X = \{x_1, \ldots, x_n\}$, define $X' = \mathcal{C}$, and $\mathcal{C}' = \{T_1, \ldots, T_n\}$ where $T_i = \{S \in \mathcal{C} : x_i \in S\}$. In English, our new ground set is the old collection $\mathcal{C}$ of subsets, and a generic new set T_i is the collection of sets $S \in \mathcal{C}$ that contain the element x_i. The reader should verify that set covers of $(X, \mathcal{C})$ are in one-to-one correspondence with hitting sets of $(X', \mathcal{C}')$, and that this correspondence preserves the cardinalities of the solutions; the two problems are thus equivalent from the perspectives of both optimization and approximation.

There is, however, one final detail to consider. We are assuming that the input Set Cover instance $(X, \mathcal{C})$ has bounded VC-dimension, say VC-dimension d. What does this imply for the VC-dimension of the our new set system $(X', \mathcal{C}')$? By "dualizing", have we thrown away the guarantee that our set systems are "simple"? Fortunately, the following lemma shows that if the VC-dimension of $(X, \mathcal{C})$ is a fixed constant, then so is that of $(X', \mathcal{C}')$.

Lemma 1.2 *If the set system $(X, \mathcal{C})$ has VC-dimension d, then the dual set system $(X', \mathcal{C}')$ has VC-dimension at most $2^{d+1} - 1$.*

Proof. Suppose for contradiction that a subset of X' of size 2^{d+1} is shattered by $\mathcal{C}'$; we will then show that a subset of X of size $d + 1$ is shattered by $\mathcal{C}$.

The reader should verify that a set $A \subseteq X'$ is shattered by $\mathcal{C}'$ if and only if for each subset $A' \subseteq A$, there is an element $x \in X$ such that $x \in S$ if $S \in A'$ and $x \notin S$ if $S \in A \setminus A'$ (this requires a bit of definition-chasing).[3] Let $A \subseteq X'$ be a shattered subset of size $k \equiv 2^{d+1}$, and write $A = \{S_0, S_1, \ldots, S_{k-1}\}$ (with each $S_i \in \mathcal{C}$). For $i = 1, 2, \ldots, d + 1$, let $x_i \in X$ be an element that belongs to S_j if and only if the ith significant bit of j (written in binary) is 1 (x_i exists since A is shattered). Then, the set $B = \{x_1, \ldots, x_{d+1}\}$ is shattered by $\mathcal{C}$: for $B' \subseteq B$, we have $B \cap S_j = B'$, where the ith bit of j is 1 if and only if $x_i \in B'$. ■

[3]Remember, $A' \subseteq A \subseteq X'$ and thus each element of A' is a set of $\mathcal{C}$ — and in particular, a subset of X.

2 Weighted ϵ-Nets

The reason for preferring Hitting Set to Set Cover is that the definition of a hitting set for a set system is very close to that of an ϵ-net, and we already know how to find small ϵ-nets in set systems of low VC-dimension. There is, however, a pesky deficiency of ϵ-nets: they are only required to intersect big sets of a set system, whereas a hitting set must intersect *all* sets of a set system, big or small. To overcome this difficulty, we next introduce a generalization of ϵ-nets, namely *weighted ϵ-nets.* The idea is to view an ϵ-net as a sample that hits all "sufficiently important" sets, where importance is measured by cardinality; we wish to introduce weights so that the importance of a set can be measured in a more flexible way.

Definition 2.1 *Suppose $(X, \mathcal{C})$ is a set system and $w : X \rightarrow \mathcal{Z}^+$ is a weight function. Define $w(S) = \sum_{x \in S} w(x)$ for $S \subseteq X$. A set $N \subseteq X$ is a* weighted ϵ-net *for $(X, \mathcal{C}, w)$ if $N \cap S \neq \emptyset$ whenever $S \in \mathcal{C}$ and $w(S) \geq \epsilon \cdot w(X)$.*

It is not immediately clear how adding weights to ϵ-nets will rectify their deficiencies. To illustrate the potential use of weighted ϵ-nets in finding hitting sets, suppose we knew the optimal solution H^* to a Hitting Set instance $(X, \mathcal{C})$. Define a weight function w^* by $w^*(x) = 1$ if $x \in H^*$ and $w^*(x) = 0$ otherwise, and put $\epsilon = \frac{1}{OPT}$ (where $OPT = |H^*|$). Now consider a weighted ϵ-net N for $(X, \mathcal{C}, w^*)$: we have $w^*(X) = OPT$ and $w^*(S) \geq 1$ for all $S \in \mathcal{C}$ (since H^* is a hitting set for $(X, \mathcal{C})$), hence $w^*(S) \geq 1 \geq \epsilon \cdot w^*(X)$ for all $S \in \mathcal{C}$. In other words, any weighted ϵ-net for $(X, \mathcal{C}, w^*)$ is in fact a hitting set for $(X, \mathcal{C})$! Moreover, assuming that the VC-dimension of $(X, \mathcal{C})$ is a fixed constant and that weighted ϵ-nets have the same size as unweighted ϵ-nets (we'll justify this below), the size of our weighted ϵ-net is only $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon}) = O(OPT \log OPT)$. To sum up, a judicious choice of a weight function and a value of ϵ allows us to find a small (within a $O(\log OPT)$ factor of optimal) hitting set by simply invoking a black box for finding weighted ϵ-nets. Of course, our algorithm will not know H^* (after all, we're trying to approximately compute H^*) and thus cannot compute w^* or ϵ as above; nevertheless, this example suggests that weighted ϵ-nets may be the right tool for finding small hitting sets.

Convinced that weighted ϵ-nets are worth investigating, we turn our attention toward computing them efficiently. As we saw in the previous lecture, proving from scratch that set systems with low VC-dimension admit small ϵ-nets is non-trivial. We would like to avoid duplicating this work for the weighted case, preferring instead to reduce the weighted problem to the unweighted one. Happily, this is not difficult, and we sketch the reduction below.

Let $(X, \mathcal{C})$ be a set system with VC-dimension d and $w : X \rightarrow \mathcal{N}$ a weight function. Obtain $(X', \mathcal{C}')$ from $(X, \mathcal{C})$ by replicating each $x \in X$ $w(x)$ times. Formally, define $X' = \cup_{x \in X} \{x^1, \ldots, x^{w(x)}\}$ and $\mathcal{C}' = \{\cup_{x \in S} \cup_{i=1}^{w(x)} x^i : S \in \mathcal{C}\}$; in words, each set S' of $\mathcal{C}'$ corresponds to a set S of $\mathcal{C}$ and contains all duplicates of the elements of S. We would like to make the following naive argument: select a random sample N' of X' of size $\frac{8d}{\epsilon} \log \frac{8d}{\epsilon} + \frac{4}{\epsilon} \log \frac{2}{\delta}$ (from previous lectures, this is an ϵ-net for $(X', \mathcal{C}')$ with probability at least $1 - \delta$) and take as our weighted ϵ-net $N = \{x \in X : x^i \in N'$ for some $i\}$ (the elements of X that had at least one of their duplicates selected for N'). For this argument to hold water, we need to check two things:

(i) if N' is an ϵ-net for $(X', \mathcal{C}')$, then N is a weighted ϵ-net for $(X, \mathcal{C}, w)$

(ii) the VC-dimension of $(X', \mathcal{C}')$ is at most that of $(X, \mathcal{C})$.

Both of these statements are true, easy to prove, and left as exercises to the reader (hint for (ii): note that no subset of X' that contains two copies of some $x \in X$ can be shattered by $\mathcal{C}'$).

Remark: If we want to use the naive sampling above for computing weighted ϵ-nets and also want a polynomial-time algorithm, we need to ensure that all of our weights remain polynomial in the size of the original (unweighted) input.

3 The Algorithm

The main idea of the algorithm is to "learn" or "evolve" a weight function w that approximates the weight function w^* of the previous section. The approach is, roughly speaking, trial and error:

Algorithm BG

```
(1) initialize $w(x) \leftarrow 1$ for all $x \in X$
(2) find a weighted $\epsilon$-net $N$ for $(X, \mathcal{C}, w)$
    (2a) if $N$ is a hitting set for $(X, \mathcal{C})$, halt and return $N$
    (2b) else, find a set $S \in \mathcal{C}$ not hit by $N$, double the weights of all elements in $S$, and
         goto (2).
```

The algorithm is underdetermined: we have yet to specify a value of ϵ. Recall in the previous section we took $\epsilon = \frac{1}{OPT}$, but of course we don't know the value of OPT. For now, however, we will assume that we do (toward the end of the lecture, we will show that this assumption is easily removed), and we will take $\epsilon = \frac{1}{2OPT}$ (we need a bit of slack since our weight vector w will only be an approximation of w^*).

Digression: Our struggle in choosing a value for ϵ highlights a fundamental difference between the current application of ϵ-nets and those we've seen in previous lectures (approximate center points, test sets for polyhedra, and approximate learning). In previous applications, the problem description included some type of "slack parameter" (e.g., the fraction of data that may be misclassified), which in turn plugged in quite directly as a value of ϵ for the ϵ-nets to be computed. Here, there is no slack allowed in the algorithm output (we insist on a feasible hitting set) and we use ϵ as an internal parameter to control the size of our weighted ϵ-nets (and hence of our eventual hitting sets).

We now analyze the running time of the algorithm. Note that finiteness of the algorithm is not obvious (perhaps our weight function w never approaches a reasonable approximation of w^*, and hence our weighted ϵ-nets always fail to be hitting sets). The main result of the lecture is that the algorithm not only terminates, but does so in very few iterations.

Theorem 3.1 (Brönnimann, Goodrich '94) *Algorithm BG terminates in $O(n)$ iterations, where $n = |X|$.*

Proof. Fix a HITTING SET instance $(X, \mathcal{C})$ and an optimal hitting set $H^* \subseteq X$ for $(X, \mathcal{C})$. At the highest level, the proof idea is as follows:

(1) prove that the total weight of the system, $w(X)$, does not grow too quickly as a function of the number of iterations of the algorithm (roughly, since all we do is double the weight of a set which previously had very small weight)

(2) prove that the total weight concentrated on H^*, $w(H^*)$, grows pretty fast (since some element of H^* has its weight doubled in every iteration)

(3) noting that $w(H^*) \leq w(X)$ always holds, (1) and (2) imply that the algorithm cannot run indefinitely.

We make these steps precise in turn. We begin by upper bounding the growth rate of the total weight in the system. Initially, we have $w(X) = |X| = n$. In each iteration, the set S selected in step (2b) satisfies $w(S) < \epsilon \cdot w(X)$ (since the sample N is a weighted ϵ-net); since doubling the weight of elements in S adds a total of $w(S)$ new weight to the system, we find that $w(X)$ grows by at most a $(1+\epsilon)$ factor in each iteration. Thus, after k iterations, we have

$$w(X) \leq n(1+\epsilon)^k.$$

Next, we lower bound the growth rate of $w(H^*)$. Initially, we have $w(H^*) = |H^*| = OPT$. Since H^* is a hitting set, there is at least one element of H^* in each set of $\mathcal{C}$; thus, no matter which set S is chosen in step (2b), some element of H^* will have its weight doubled. By convexity of the function 2^x, $w(H^*)$ will be minimized if these doublings are spread out over the elements of H^* as evenly as possible. Thus, after k iterations, we have

$$w(H^*) \geq OPT \cdot 2^{k/OPT}.$$

We have already noted that $w(H^*) \leq w(X)$ always holds (all weights are positive and $H^* \subseteq X$); it thus remains to determine the largest k for which

$$OPT \cdot 2^{k/OPT} \leq n\left(1 + \frac{1}{2OPT}\right)^k$$

can be true. Taking logs, we obtain that any such k must satisfy

$$\log_2 OPT + \frac{k}{OPT} \leq \log_2 n + k \log_2\left(1 + \frac{1}{2OPT}\right).$$

Solving for k, we derive

$$k \leq \frac{\log_2 n - \log_2 OPT}{\frac{1}{OPT} - \log_2(1 + \frac{1}{2OPT})} \leq 4 \cdot OPT \log_2 \frac{n}{OPT}$$

where we have used that $\log_2(1+x) \leq \frac{3}{2}x$ for $x \geq 0$. Since the expression on the RHS is $O(n)$ for all possible values of OPT, the theorem follows. ■

Remarks:

1. Our analysis assumes that the sample N obtained in step (2) of the algorithm is in fact a weighted ϵ-net for $(X, \mathcal{C}, w)$ (recall that there is a probability δ that a random sample fails to be an ϵ-net). The easiest way to enforce this assumption is to simply verify that the sample N is a weighted ϵ-net (i.e., for each $S \in \mathcal{C}$, check if $w(S) \geq \epsilon \cdot w(X)$ and, if so, that $N \cap S \neq \emptyset$), and to resample if it is not. This is easy to do efficiently since we are given an explicit description of $(X, \mathcal{C})$.

2. Recall that, to achieve a polynomial running time, we need to check that our weights always remain polynomial in the original input size. Plugging in our upper bound for the largest value that k can achieve, our bound $w(X) \leq n(1 + \frac{1}{2OPT})^k$ shows that $w(X) = O(n^4)$ throughout the course of the algorithm.

3. We assumed in the algorithm and in the analysis that we knew the value of OPT. To remove this assumption, we run the previous algorithm with successive guesses $OPT = 1, 2, 4, 8, \ldots$. If we run the algorithm with a guess that is less than the true optimum, Theorem 3.1 no longer holds; however, if the algorithm runs for $4 \cdot OPT \log_2 (n/OPT)$ iterations without terminating, we know that our current value of OPT is too small and that we can abort and restart with a larger value of OPT. On the other hand, if we get lucky and our algorithm terminates using a small guess for OPT, we will only have a smaller hitting set and hence a better approximation. Finally, Theorem 3.1 *does* hold if OPT is larger then the true optimum; hence, in the worst case, our algorithm will terminate using a value of OPT that is twice as big as the true optimum, and our final hitting set will be a little bit bigger than promised (but only by a constant factor).

4. There are set systems that admit linear size ϵ-nets (i.e., set systems with VC-dimension $O(1)$ and ϵ-nets of size $O(\frac{1}{\epsilon})$). For these types of set systems, our algorithm and analysis yield a constant-factor approximation algorithm. For example, this algorithm was used to give the first constant-factor approximation algorithm for the problem of covering a given set of points in the plane with the smallest subcollection of a given collection of discs (a set of discs in the plane has bounded VC-dimension and admits a linear size ϵ-net).