# CS 170 Homework 14

## 1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, explicitly write "none".

## 2 Hitting Set

In the Hitting Set Problem, we are given a family of finite integer sets $\{S_1, S_2, \ldots, S_n\}$ and a budget $b$, and we wish to find an integer set $H$ of size $\leq b$ which intersects every $S_i$, if such an $H$ exists. In other words, we want $H \cap S_i \neq \emptyset$ for all $i$.

Show that the Hitting Set Problem is NP-complete. (Hint: Hitting Set generalizes one of the problems covered in Chapter 8 of the textbook).

**Solution:** We can see that the problem is in NP since we can quickly check that a potential hitting set covers all sets and has size less than $b$.

In addition, Hitting Set is a generalization of the Vertex-Cover Problem. Given a graph $G$, consider each edge $e = (u, v)$ as a set containing the elements $u$ and $v$. Then, finding a hitting set of size at most $b$ in this particular family of sets is the same as finding a vertex cover of size at most $b$ for the given graph. Since Vertex Cover is NP-Hard, Hitting Set must be NP-Hard as well.

One can also reduce from the Set Cover problem. For every set $S$ in Set Cover, we create an element $e'_S$ in Hitting Set, and for every element $e$ in Set Cover, we create a set $S'_e$ in Hitting Set. Each $S'_e$ contains all elements $e'_S$ corresponding to sets $S$ containing $e$. Again, finding a hitting set of size at most $b$ is exactly the same as finding a set cover of size at most $b$ in the original instance.

## 3 Orthogonal Vectors

In the 3-SAT problem, we have $n$ variables and $m$ clauses, where each clause is the OR of (at most) three of these variables or their negations. The goal of the problem is to find an assignment of variables that satisfies all the clauses, or correctly declare that none exists.

In the orthogonal vectors problem, we have two sets of vectors $A, B$. All vectors are in $\{0,1\}^m$, and $|A| = |B| = n$. The goal of the problem is to find two vectors $a \in A, b \in B$ whose dot product is 0, or correctly declare that none exists. The brute-force solution to this problem takes $O(n^2 m)$ time: compute all $|A||B| = n^2$ dot products between two vectors in $A, B$, and each dot product takes $O(m)$ time.

Show that if there is a $O(n^c m)$-time algorithm for the orthogonal vectors problem for some $c \in [1, 2)$, then there is a $O(2^{cn/2} m)$-time algorithm for the 3-SAT problem. For simplicity, you may assume in 3-SAT that the number of variables must be even.

*Hint: Try splitting the variables in the 3-SAT problem into two groups.*

**Solution:** We use an $O(2^{n/2} m)$-time reduction from 3-SAT to orthogonal vectors. We split the variables into two groups of size $n/2$, $V_1$ and $V_2$. For each group, we enumerate all $2^{n/2}$ possible assignments of these variables. For each assignment $x$ of the variables in $V_1$, let $v_x$ be the vector where the $i$th entry is 0 if the $i$th clause is satisfied by one of the variables in this assignment, and 1 otherwise. We ignore the variables in the clause that are in $V_2$. For example, if clause $i$ only contains variables in $V_2$, then $v_x(i) = 0$ for all $x$. Let $A$ be the $2^{n/2}$ vectors produced this way.

We construct $B$ containing $2^{n/2}$ vectors in a similar manner, except using $V_2$ instead of $V_1$.

We claim that the 3-SAT instance is satisfiable if and only if there is an orthogonal vector pair in $A \times B$. Given this claim, we can solve 3-SAT by making the orthogonal vectors instance in $O(2^{n/2} m)$ time, and then solving the instance in $O((2^{n/2})^c m) = O(2^{cn/2} m)$ time.

Suppose there is a satisfying assignment to 3-SAT. Let $x_1$ be the assignment of variables in $V_1$, and $x_2$ be the assignment of variables in $V_2$. Let $v_1, v_2$ be the vectors in $A, B$ corresponding to $x_1, x_2$. Since every clause is satisfied, one of $v_1(i)$ and $v_2(i)$ must be 0 for every $i$, and so $v_1 \cdot v_2 = 0$. So there is also a pair of orthogonal vectors in the orthogonal vectors instance.

Suppose there is a pair of orthogonal vectors $v_1, v_2$ in the orthogonal vectors instance. Then for every $i$, either $v_1(i)$ or $v_2(i)$ is 0. In turn, for the corresponding assignment of variables in $V_1, V_2$, the combination of these assignments must satisfy every clause. Hence, the combination of these assignments is a satisfying assignment for 3-SAT.

Comment: It is widely believed that SAT has no $O(2^{.999n} m)$-time algorithm - this is called the Strong Exponential Time Hypothesis (SETH). So it is also widely believed that orthogonal vectors has no $O(n^{1.99} m)$-time algorithm, since otherwise SETH would be violated. It turns out that we can reduce orthogonal vectors to string problems such as edit distance and longest common subsequence, and so if we belive SETH then we also believe those problems also don't have $O(n^{1.99})$-time algorithms. The field of research studying reductions between problems with polynomial-time algorithms such as these is known as fine-grained complexity, and orthogonal vectors is one of the central problems in this field.

## 4 Local Search for Max Cut

Sometimes, local search algorithms can give good approximations to NP-hard problems. Recall that in the Max-Cut problem, we have an unweighted graph $G = (V, E)$ and we want to find a cut $(S, T)$ that maximizes the number of edges "crossing" the cut (i.e. with one endpoint in each of $S, T$). Consider the following local search algorithm:

1. Start with any cut (e.g. $(S, T) = (V, \emptyset)$).
2. While there is some vertex $v \in S$ such that more edges cross $(S \setminus \{v\}, T \cup \{v\})$ than $(S, T)$ (or some $v \in T$ such that more edges cross $(S \cup \{v\}, T \setminus \{v\})$ than $(S, T)$), move $v$ to the other side of the cut.

Now, let us prove a couple of guarantees that this algorithm achieves.

(a) Give an upper bound on the number of iterations this algorithm can run for (i.e. the total number of times we move a vertex).

(b) Show that when this algorithm terminates, it finds a cut where at least half the edges in the graph cross the cut.

*Hint: when we move $v$ from $S$ to $T$, $v$ must have more neighbors in $S$ than $T$. What does this observation suggest about the neighbors of each vertex once the algorithm terminates? Then, what can we say about the number of edges crossing the cut vs. the number of edges within each side of the cut?*

**Solution:**

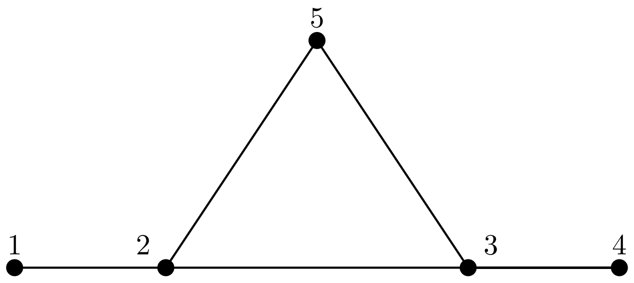
(a) $|E|$ iterations. Each iteration increases the number of edges crossing the cut by at least 1. The number of edges crossing the cut is between 0 and $|E|$, so there must be at most $|E|$ iterations.

(b) $\delta_{in}(v)$ be the number of edges from $v$ to other vertices on the same side of the cut, and $\delta_{out}(v)$ be the number of edges from $v$ to vertices on the opposite side of the cut. The total number of edges crossing the cut the algorithm finds is $\frac{1}{2}\sum_{v \in V} \delta_{out}(v)$, and the total number of edges in the graph is $\frac{1}{2}\sum_{v \in V}(\delta_{in}(v) + \delta_{out}(v))$. We know that $\delta_{out}(v) \geq \delta_{in}(v)$ for all vertices when the algorithm terminates (otherwise, the algorithm would move $v$ across the cut), so the former is at least half as large as the latter.

## 5 Second Smallest Global Min-cut

Recall that a *cut* of an undirected, unweighted graph $(V, E)$ is a partition of the vertices into two non-empty sets $S$ and $V \backslash S$. The *weight* of such a cut is the number of edges crossing the cut, which we denote $|E(S, V\backslash S)|$. Karger's contraction algorithm from class finds a cut of minimum weight with probability $1 - p$, with runtime $O(poly(n) \cdot \log(1/p))$, where $n := |V|$ and $m := |E|$.

In this problem, we will be concerned with finding not the smallest, but rather the second smallest minimum cut. Ties are broken arbitrarily. For example, consider the graph below:



It has two cuts of weight one, given by $S = \{1\}$ or $S = \{4\}$. All other cuts have weight at least two. Since we break ties arbitrarily, we would say in this example both the smallest *and* second smallest cuts have weight equal to 1.

(a) Give a general formula for the number of cuts in a graph with $n$ vertices and $m$ edges. **Note:** we are asking for the number of cuts, not necessarily minimum cuts.

(b) Focus on a particular second smallest mincut $S^*$ of $G$. Consider running just the first step of Karger's contraction algorithm, where we contract a uniformly random edge. Compute a tight upper bound $\alpha$ for the probability that the contracted edge crosses $(S^*, V\backslash S^*)$. Your answer is allowed to depend on $k := |S^*|$ or $n$, but not on $m$.

(c) Starting with $n$ vertices, imagine doing $n - 3$ random contraction steps in sequence. How many "supervertices" are left?

(d) Suppose your answer in part (c) is $c$. Imagine then picking a uniformly random cut (not necessarily a mincut) on this graph of $c$ supervertices to get a resulting cut of the original graph. Show that the probability that this resulting cut is $S^*$ is $\Omega(1/n^2)$.

(e) Give a $O(poly(n) \cdot \log(1/p))$ time algorithm for finding the second smallest mincut in a graph with success probability at least $1 - p$. You need not calculate the precise exponent of $n$ in the $poly(n)$ term; any polynomial dependence suffices.

*Hint:* $(1 + x)^T \leq e^{xT}$ *for all* $x \in \mathbb{R}$ *and* $T \geq 0$.

**Solution:**

(a) For each vertex, we can either place it on the same side of the cut as vertex 1 or on the opposing side. This gives $2^{n-1}$ partitions into two sets. We should then remove the partition in which every vertex chooses to be in the same partition as 1, since then the other side will be empty. Thus the answer is $2^{n-1} - 1$.

(b) Then $k$ be the number of vertices in the second smallest cut. Then the probability of contracting an edge across this cut is $k/m$, but we must give an answer independent of $m$. Although the smallest vertex degree can be arbitrarily small, all other vertices must have degree at least $k$ (since otherwise the second smallest cut would have size less than $k$). If $d_v$ is the degree of vertex $v$, then we know that

$$m = \frac{1}{2}\left(\sum_v d_v\right),$$

where we argued that the right hand side must be at least $\frac{(n-1)k}{2}$. Therefore $k/m \leq 2/(n-1)$, so we set $\alpha = 2/(n-1)$.

(c) 3

(d) Recall the proof of Karger's contraction algorithm from class. In order to be able to pick $S^*$ in the graph of supervertices, no edge in $S^*$ should have been contractedotherwise, we will forever be missing a few edges from $S^*$ in our graph of supervertices. The probability that no edge in $S^*$ has been contracted is lower bounded by the following:

$$\begin{aligned}\prod_{i=0}^{n-4}\left(1-\frac{2}{n-1-i}\right) &= \prod_{i=0}^{n-3}\left(\frac{n-3-i}{n-1-i}\right)\\ &= \left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\cdots\left(\frac{1}{3}\right)\\ &= \frac{1\cdot 2}{(n-1)(n-2)} \in \Theta(\frac{1}{n^2}).\end{aligned}$$

Now in the resulting graph on three supervertices, there are 3 possible cuts by part (a). Thus the probability that the cut we output is $S^*$ is one-third of the above, which is $\Omega(1/n^2)$.

(e) We first run Karger's algorithm from class to find a minimum cut $(S, V\backslash S)$ with success probability $1-p/2$. We then run the contraction algorithm $T$ times, each time contracting down to three supervertices then returning one of the three cuts uniformly at random. The probability that we fail to ever find a particular second smallest mincut is at most

$$\left(1-\frac{1}{3}\cdot\frac{2}{(n-1)(n-2)}\right)^T \leq e^{-\frac{2T}{3(n-1)(n-2)}} \leq p/2$$

for $T = \lceil(3/2)(n-1)(n-2)\rceil = \Theta(n^2\log(1/p))$. Thus the total success probability is

$$\begin{aligned}P[\text{Finds second smallest mincut}] &= P[\text{Finds second smallest mincut} \mid \text{Finds mincut}]\cdot P[\text{Finds mincut}]\\ &\geq (1-p/2)\cdot(1-p/2)\\ &\geq 1-p\end{aligned}$$

as desired. The overall the runtime is $\Theta(n^2\log(1/p)) = \Theta(poly(n)\cdot\log\frac{1}{p})$, also as desired.

## 6 How to Gamble With Little Regret

Suppose that you are gambling at a casino. Every day you play at a slot machine, and your goal is to minimize your losses. We model this as the experts problem. Every day you must take the advice of one of $n$ experts (i.e. play at a slot machine of their choosing). At the end of each day $t$, if you take advice from expert $i$, the advice costs you some $c_i^t$ in $[0, 1]$. You want to minimize the regret $R$, defined as:

$$R = \frac{1}{T}\left(\sum_{t=1}^{T} c_{i(t)}^t - \min_{1\leq i\leq n} \sum_{t=1}^{T} c_i^t\right)$$

where $i(t)$ is the expert you choose on day $t$. Notice that in this definition, you are comparing your losses to the best expert, rather than the best overall strategy.

Your strategy will be probabilities where $p_i^t$ denotes the probability with which you choose expert $i$ on day $t$. Assume an all powerful adversary (i.e. the casino) can look at your strategy ahead of time and decide the costs associated with each expert on each day. Let $C$ denote the set of costs for all experts and all days. Compute $\max_C(\mathbb{E}[R])$, or the maximum possible (expected) regret that the adversary can guarantee, for each of the following strategies, with justification.

(a) Any deterministic strategy, i.e. for each $t$, there exists some $i$ such that $p_i^t = 1$.

(b) Always choose an expert according to some fixed probability distribution at every time step. That is, fix some $p_1, \ldots, p_n$, and for all $t$, set $p_i^t = p_i$.

What distribution minimizes the regret of this strategy? In other words, what is $\text{argmin}_{p_1,\ldots,p_n} \max_C(\mathbb{E}[R])$?

This analysis should conclude that a good strategy for the problem must necessarily be randomized and adaptive.

**Solution:**

(a) $\frac{n-1}{n}$. Consider the case where the cost of the chosen expert is always 1, and the cost of each other expert is 0. Let $k$ be the least-frequently chosen expert, and let $m_k$ be the number of times that expert is chosen. This will result in a regret of $\frac{1}{T}(T - m_k)$

Since the best expert is the one that is chosen least often, the best strategy will try to maximize the number of times we choose the expert that is chosen least often. This means we want to choose all the experts equally many times, so expert $k$ is chosen in at most $T/n$ of the rounds. Therefore, $m_k \leq \frac{T}{n}$, thus the regret is at least $\frac{1}{T}(T - \frac{T}{n}) = \frac{n-1}{n}$.

(Note here that even if our strategy is adaptive, i.e. it chooses an expert on day $i$ based on the losses from days 1 to $i - 1$, rather than committing to an expert for day $i$ before seeing the loss for day 1, it still can't achieve regret better than $\frac{n-1}{n}$.)

(b) $(1 - \min_i p_i)$. Like in part (a), the distribution is fixed across all days, so we know ahead of time which expert will be chosen least often in expectation. Let $k = \text{argmin}_i p_i$ be the expert with least cost. Let $c_k^t = 0$ for all $t$, and let $c_i^t = 1$ for all $i \neq k$ and for all $t$. This way, $\mathbb{E}\left[\sum_{t=1}^{T} c_{i(t)}^t\right] = T\left(\sum_{i \neq k} p_i \cdot 1 + p_k \cdot 0\right) = T(1 - p_k)$. We also have $\min_i \sum_{t=1}^{T} c_i^t = 0$, so we end up with an expected regret of $\frac{1}{T}(T(1 - p_k) - 0) = 1 - p_k$.

To minimize the expectation of $R$ is the same as maximizing $\min_i p_i$, which is achieved by the uniform distribution. This gives us regret $\frac{n-1}{n}$ (this is the same worst case regret as in part (b)).

## 7 Variants on the Experts Problem

Consider the *realizable* experts problem: We have $n$ experts who predict it will either rain or shine tomorrow. At least one of the experts is perfect and will always make the correct prediction. Every day we guess based on the experts if it will rain or shine tomorrow. Our goal is to make as few mistakes in our predictions as possible.

(a) Suppose we always guess the prediction of the majority of experts who have been correct so far, breaking ties arbitrarily (Note that this set is always non-empty since one expert is always correct). Show that we make at most $\log n$ mistakes using this strategy.

*Hint: if you're stuck, look at the Halving Algorithm from lecture*

(b) Suppose there are now $k$ experts who are always correct instead of just one. Give a better upper bound on the number of mistakes the algorithm from the previous part makes makes.

(c) Suppose instead of one expert always being correct, we are only guaranteed that there is one expert who makes at most $k$ mistakes. Consider the following algorithm: Let $m$ be the least mistakes made by any expert so far. Use the prediction of the majority of experts who have made at most $m$ mistakes so far.

Give an upper bound on the number of mistakes made by this algorithm.

**Solution:**

(a) Every time we make a mistake, at least half of the experts who have been correct so far must also make a mistake, so the set of experts who have not made a mistake decreases by at least half. This set starts at size $n$, and can't decrease past size 1, so we must make at most $\log n$ mistakes.

(b) Every time we make a mistake, the number of experts we stop listening to decreases in size by at least 1/2. So we go from $n$ experts to at most $2k - 1$ experts in at most $\lceil \log(n/(2k-1)) \rceil$ mistakes. At this point, the experts who are always correct are the majority, so we never make a mistake.

(We will give full credit for $\log(n/k)$, since this is the best upper bound you can get without using floor/ceiling functions; this is tight in the case where $n, k$ are both powers of two and half the experts make a mistake every round. Other off-by-one solutions or solutions that are correct up to rounding also get full credit.)

(c) $k(\log n + 1) + \log n$. Every time we make a set of mistakes, the set of experts who have made at most $m$ mistakes must have decreased in size by at least 1/2. When $m < k$, this set goes from size at most $n$ to 0 before $m$ increases by 1, so with every increase in $m$ we make at most $\log n + 1$ mistakes. When $m = k$, this set can go from size $n$ to 1 which takes us at most $\log n$ mistakes.

## 8 Informed Predictions, Randomized!

Define the *realizable* experts problem to be as follows: there are $n$ experts who each make predictions every day and we need to use the expert predictions to make our own daily predictions; additionally, there is guaranteed to be at least one *perfect* expert that is always correct. In class, we found that the Halving Algorithm (which is deterministic) makes at most $\log_2 n$ mistakes.

Describe a randomized algorithm that, in expectation, makes at most $\log_e n = \ln n$ mistakes. Prove that your algorithm, indeed, makes $\ln n$ mistakes in expectation.

*Hint: you may use the identity that* $\sum_{i=1}^{n} \frac{1}{i} \leq \ln n + 1$.

**Solution:** We define our algorithm as such: at every step $i$, we consider all experts that have not made a mistake yet (call this set of experts $P_i$) and choose one of these experts at random and take their prediction.

We claim that this algorithm will make at most $\ln n$ errors. First, we notice that there are only at most $n-1$ places where a "previously-perfect" expert can make a mistake. We can notice that we do not make a mistake if none of the experts do, so without loss of generality, let's say that every round has at least one previously-perfect expert make a mistake, until we are only left with perfect experts. If at step $i$, there are $m$ out of $|P_i|$ experts that make a mistake, then the probability that we make a mistake is $\frac{m}{|P_i|}$. Note that $m$ can also be written as $|P_i| - |P_{i+1}|$. Therefore, if in total $k$ rounds included a mistake, the expected total number of mistakes made is

$$\sum_{i=1}^{k} \frac{|P_i| - |P_{i+1}|}{|P_i|} = \sum_{i=1}^{k} (|P_i| - |P_{i+1}|) \frac{1}{|P_i|} \leq \sum_{i=1}^{k} \left( \frac{1}{|P_i|} + \frac{1}{|P_i| - 1} + \cdots + \frac{1}{|P_{i+1}| + 1} \right).$$

We notice that $|P_1| = n$, so this last sum can be written as

$$\frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{|P_k| + 1} = \sum_{i=2}^{n} \frac{1}{i} = \left( \sum_{i=1}^{n} \frac{1}{i} \right) - \frac{1}{1} \leq (\ln n + 1) - 1 = \ln n.$$

Thus, we make at most $\ln n$ mistakes in expectation.

*Remark: interestingly, this is not optimal! It turns out you can actually do even better than* $\ln n$ *if you adjust the probability that you choose each expert (in particular, making majority prediction experts slightly more likely to be chosen).*