**Probability and Algorithms** **Caltech CS150b, Fall 2019**
**Leonard J. Schulman, schulman@caltech.edu** **TA: Jenish Mehta, jenishc@gmail.com**
**Problem set 2** **Out Friday Oct 18, Due Tuesday Nov 5 in Jenish's mailbox.**

1. *Like any dealer he was watching for the card*
*that is so high and wild*
*he'll never need to deal another*
Leonard Cohen, Stranger Song

A $k$-SAT formula on $n$ boolean variables is a conjunction of disjunctions, each consisting of at most $k$ literals (variables or their negations). The problem of determining whether such a formula is satisfiable is NP-complete for any $k \geq 3$. Of course it can be solved by exhaustive search in time $2^n \text{poly}(n)$.

It has been known for a long time that randomized algorithms can find a solution with high probability, if one exists, somewhat more quickly. In 1999 Schöning came up with a new randomized algorithm that was both faster and simpler than anything known previously. (Faster methods have since been developed, but are more complicated and are only slightly faster.)

For concreteness we focus on $k = 3$. As promised, the algorithm is very simple:

Do the following $(4/3)^n \text{poly}(n)$ times: Select an assignment $a$ of the variables uniformly at random and then do the following $3n$ times, or until a satisfying assignment is encountered: select an arbitrary unsatisfied clause, choose one of its literals uniformly at random, and flip that literal.

Let $a^*$ be a satisfying assignment to the formula. The basic idea is that $a^*$ exerts an attracting force on the random walk. Although the entropic force still *typically* sends the walk to a Hamming distance of about $n/2$ away from $a^*$, the attracting force is nonetheless strong enough that the walk is significantly more likely to visit $a^*$ (or another satisfying assignment if it gets trapped there first), than a uniform walk would be.

The first step in the analysis is a *lower bound* on the tail of a binomial distribution. Show:

**Lemma 1.** *Consider a sequence of iid random variables $X_1, \ldots,$ taking values in $\{1, -1\}$, with $\Pr(X_i = 1) = p < 1/2$. For $m, t > 0$ let $A_{m,t}$ be the event that $\sum_{i=1}^{t} X_i \geq m$, and let $A_m = \bigcup_t A_{m,t}$. Set $t(m) = m/(1-2p)$. Then*

$$\Pr(A_{m,t(m)}) \leq \Pr(A_m) \leq \Pr(A_{m,t(m)}) poly(m) = e^{-D(1-p||p)m/(1-2p)} poly(m).$$

(Thus there is a key moment in time which witnesses a polynomial fraction of the occurrences of this rare event.)

Next, classify rounds of the algorithm according to the Hamming distance $d$ between $a$ and $a^*$. There is a tradeoff between the success probability of the round, which decreases in $d$, and the sample size which you need to use before some $a$ is likely to achieve this $d$. By studying this tradeoff you should be able to achieve runtime $(4/3)^n \text{poly}(n)$.

2. A *perfect matching* (you may recall from cs150a) in an $n$-vertex graph is a set of $n/2$ edges, no two of which are incident on a common vertex. We are interested here in algorithms which do not have access to the graph on an "input tape", but instead access the graph by posing queries of the form "Is $(i, j)$ an edge?" where $1 \leq i < j \leq n$. ($n$ is known in advance.) That is, if $G_{ij} = 1$ for edges, $= 0$ for non-edges, we have to pay for each $G_{ij}$ we ask for.

(a) Show that any Las Vegas randomized algorithm for determining whether the graph has a perfect matching, has query complexity $\Omega(n^2)$.

*Hint:* Actually something much stronger is true, and may be more convenient to prove. Even the nondeterministic query complexity of "there is no perfect matching" is $\Omega(n^2)$. That is, there are graphs $G$ without perfect matchings for which any proof of this fact requires revealing the values of $\Omega(n^2)$ inputs $G_{ij}$.

(b) Show that the Monte Carlo complexity of this problem is also $\Omega(n^2)$. (For this, you can no longer rely on the nondeterministic lower bound. But you might be able to adapt your argument for that case to a suitable probability distribution over "yes" and "no" inputs.)

3. *This problem depends upon the material on continuous-time MWU which we'll cover in class; I'm optimistic we'll cover that "in time" but if not, we'll postpone this problem to ps3.*

In this problem you are asked to convert the continuous-time MWU argument to one in discrete time. Specifically, you have to choose among $n$ actions at every discrete time $t = 1, 2$, etc.; after each round, all possible penalties $M_k(t)$ $(1 \le k \le n)$ (including the one you incurred) are revealed. Penalties are in the range $[0, 1]$.

Start with all actions having weight $w_k(0) = 1$. For the purpose of this exercise we will assume that the time $T$, after which we want to compare the performance of our algorithm to the best single action $k$ (i.e., after which we want to prove a "no regret" bound), is known in advance. Choose some $0 < \varepsilon < 1/2$. Use the following update rule for $t \ge 1$:

$$w_i(t) = (1-\varepsilon)^{M_i(t)} w_i(t-1)$$

and play in round $t$ according to the distribution $p_k(t-1) = \frac{w_k(t-1)}{\sum_i w_i(t-1)}$.

Show that the expected penalty of this strategy, $\sum_t \sum_k p_k(t-1) M_k(t)$, is at most $\varepsilon T + \frac{\ln n}{\varepsilon} + \min_k \sum_t M_k(t)$.

**Remarks:**

**(1)** The two overhead terms here can be balanced by setting $\varepsilon = \sqrt{\ln n}/\sqrt{T}$. Then

$$\frac{1}{T} \sum_t \sum_k p_k(t-1) M_k(t) \le 2\sqrt{\ln n}/\sqrt{T} + \min_k \sum_t M_k(t).$$

**(2)** If $T$ is not known in advance one can achieve essentially the result by gradually decreasing $\varepsilon$, specifically, setting $\varepsilon_t = \min\{\frac{1}{2}, \sqrt{\ln n}/\sqrt{t}\}$.