Algorithms, Probability, and Computing  Fall 2011
Special Assignment Set 2

- The solution is due on Tuesday, December 6. Please bring a print-out of your solution with you to the lecture. If you cannot attend (and please only then), you may alternatively send your solution as a PDF to robin.moser@inf.ethz.ch. We will send out a confirmation that we have received your file. Make sure you receive this confirmation within the day of the due date, otherwise complain timely.

- Please solve the exercises carefully and then write a nice and complete exposition of your solution using a computer, where we strongly recommend to use \LaTeX. A tutorial can be found at http://www.cadmo.ethz.ch/education/thesis/latex

- For geometric drawings that can easily be integrated into \LaTeX documents, we recommend the drawing editor IPE, retrievable at http://ipe7.sourceforge.net/ in source code and as an executable for Windows.

- You are welcome to discuss the tasks with your colleagues, but we expect each of you to hand in your own, individual write-up.

- There will be two special assignments this semester. Both of them will be graded and the average grade will contribute 20% to your final grade.

- This is a theory course, which means: if an exercise does not explicitly say “you do not need to prove your answer” or “justify intuitively”, then a formal proof is always required.

- As with all exercises, the material of the special assignments is relevant for the (midterm and final) exams.

Exercise 1 (The LLL Solver Irrespective) (60 Points)

Recall that the algorithm \textsc{Local-Lemma-Solver}(F) delivers a satisfying assignment to a CSP F very quickly if F satisfies the conditions of the Lovász Local Lemma. In this exercise, we are going to “abuse” the algorithm and run it on any CSP, no matter whether it satisfies the conditions or not. Since solving a general CSP is widely believed to be very hard, the algorithm is going to be rather slow. We would like to check how slow.

In this task, let F be a \((d,k)\)-CSP on variable set \(V = \{x_1, x_2, \ldots, x_n\}\) which is satisfiable (that’s the only thing we ask here, no condition on neighborhoods or such). As you know, the algorithm starts by selecting an initial assignment \(\alpha_0 : V \rightarrow \{1..d\}\) uniformly at random. If \(\alpha_0\) is not already satisfying, it selects a constraint \(C_1\) for resampling. Let us call \(\alpha_1\) the assignment after the first correction step. Unless \(\alpha_1\) is satisfying, another constraint \(C_2\) is selected for resampling and we get an assignment \(\alpha_2\). In general, let \(C_i\) be as in the lecture notes and \(\alpha_i\) the assignment the algorithm currently maintains after the \(i\)-th correction step. Let again \(N\) be the number of steps needed until the algorithm terminates.

Contrary to the analysis in the Local Lemma case, we will now fix any satisfying assignment \(\alpha^* : V \rightarrow \{1..d\}\) and consider, for \(0 \leq i \leq N\), the distances

\[
D_i := | \{ j \in \{1..n\} \mid \alpha_i(x_j) \neq \alpha^*(x_j) \} |,
\]

i.e. the number of variables in which \(\alpha_i\) and \(\alpha^*\) differ. These are random variables that track how close the algorithm is to the (fixed) satisfying assignment in every step. Furthermore, let for \(1 \leq i \leq N\) the random variable \(L_i\) be the number of literals in clause \(C_i\) which are satisfied by \(\alpha^*\). Clearly \(1 \leq L_i \leq k\) for all \(1 \leq i \leq N\), because \(\alpha^*\) is a satisfying assignment. Here is your first task:
We now define new random variables as follows. Let $B_1, B_2, \ldots$ be an unlimited supply of biassed coin flips, i.e. for all $i \in \mathbb{N}$:

$$B_i = \begin{cases} 1 & \text{with probability } (d - 1)/d \\ 0 & \text{with probability } 1/d \end{cases}$$

and such that the set $\{B_i | i \in \mathbb{N}\}$ is mutually independent. Now define a random variable $X_i$ as follows:

$$X_i = \begin{cases} D_i - D_{i-1} + L_i & \text{if } 1 \leq i \leq N \\ B_{ki+1} + B_{ki+2} + \ldots + B_{ki+k} & \text{if } i > N \end{cases}$$

This definition may look very obscure to you at first, but it provides a very elegant way of bounding the running time of our algorithm. You can easily check that for all $i \in \mathbb{N}$, $0 \leq X_i \leq k$. Here are your further tasks:

(b) Prove that all $\{X_i\}_i$ are mutually independent.

**Hint:** Express $X_i$ as a function of only $C_i$ and the random choices made in the $i$-th step. Then condition on the different values of $C_i$ and establish that the distribution of $X_i$ is always the same.

(c) Calculate the distribution of $X_i$, i.e. for all $0 \leq j \leq k$, determine $p_j := \Pr[X_i = j]$.

(d) Prove that for any fixed $t \in \mathbb{N}$, if $X_1 + X_2 + \ldots + X_t \leq t - D_0$, then $N \leq t$.

(e) Let $t \in \mathbb{N}$ and $r_0, r_1, \ldots, r_k \in \mathbb{N}_0$ be fixed numbers s.t. $r_0 + r_1 + \ldots + r_k = t$. Prove:

$$\Pr \left[ \sum_{i=1}^{t} X_i \leq \sum_{j=0}^{k} j r_j \right] \geq \left( \frac{t}{r_0, r_1, \ldots, r_k} \right)^k \prod_{j=0}^{k} p_j^{r_j}.$$  

(f) Prove: if $t \in \mathbb{N}$ and $0 \leq r \leq n$ and $r_0, r_1, \ldots, r_k$ are fixed numbers such that both

$$\sum_{j=0}^{k} r_j = t \quad \text{and} \quad \sum_{j=0}^{k} j r_j = t - r,$$

then

$$\Pr[N \leq t] \geq q_r \left( \frac{t}{r_0, r_1, \ldots, r_k} \right)^k \prod_{j=0}^{k} p_j^{r_j}.$$  

Now, the term on the right in Task (f) bounds from below the probability that the algorithm is successful within $t$ correction steps. The bound is valid no matter what we plug in for $t, r, r_0, r_1, \ldots, r_k$, so when given concrete values for $d$ and $k$, we must only find among the values satisfying the conditions the ones giving us the best bound. For large values of $d$ and $k$, this becomes a considerable amount of work. For small values, it is not so difficult.

(g) Prove: for $d = 2$ and $k = 3$ (this would be the case of a 3-CNF), we obtain $\Pr[N \leq n] \geq 1.6181^{-n+o(n)}$.

**Hint:** Use the values $t = 0.4472n$, $r = 0.1910n$, $r_0 = 0.2368n$, $r_1 = 0.1677n$, $r_2 = 0.0396n$ and $r_3 = 0.0031n$. These can be obtained by numerically optimizing the expression derived for the success probability, but we want to spare you this considerable amount of work. Plug them in and calculate numerically using the approximation

$$\left( \frac{z}{a_1 z, a_2 z, \ldots, a_i z} \right) = 2^{-\left( \sum_{j=1}^{i} a_j \log(a_j) \right) z + o(z)}.$$

for multinomial coefficients, where $a_1, \ldots, a_i \in (0, 1)$ are constants summing up to 1.

**Remark:** In fact, if one does the calculation precisely instead of numerically (which is an even more considerable amount of work), one finds that 1.6181 should in fact be $\varphi$, the golden ratio. One can also prove that this is tight. Amplifying by repetition gives an algorithm of expected running time $\varphi^{n+o(n)}$. This is not spectacular at all, albeit faster than a trivial algorithm.
Exercise 2 (Random Self-Reducibility and Performance Amplification of a Distinction Problem) (40 Points)

Let $s$ be a given binary (secret) string of length $n$. Suppose one is given a sequence of randomly chosen vectors $a_1, a_2, \ldots, a_{\ell} \in \mathbb{Z}_2^n$, and $z_i = (a_i, s) + e_i$ for $1 \leq i \leq \ell$, where $e_i \in \mathbb{Z}_2$ is a randomly chosen (“error”) bit, such that $\Pr(e_i = 1) = \alpha$. Note that if $\alpha = 0$, then it is easy to find $s$ if $\ell$ is a little larger than $n$. One simply has to wait until the set $\{a_i|1 \leq i \leq \ell\}$ has $n$ linearly independent vectors, and then use Gaussian elimination to compute $s$. This problem however becomes significantly difficult for $\alpha$ being a constant in $(0, \frac{1}{2})$.

There are several cryptosystems based on the assumption that the set of pairs $(a_i, z_i)$ for an $s$ chosen uniformly at random is indistinguishable from such pairs chosen uniformly at random from $\mathbb{Z}_2^n \times \mathbb{Z}$. In this exercise, we will show that if it is easy to distinguish these distributions in the average case, i.e., for a random $s$, with a ‘small’ success probability, then it is easy to distinguish them in the worst case, i.e., for any fixed $s$ with a ‘high’ success probability.

Let $0 < \alpha < 1/2$ be a fixed constant. Consider the following systems.

- Given $s \in \mathbb{Z}_2^n$, let $S^s$ be a system that on each invocation chooses $a \in \mathbb{Z}_2^n$ independently and uniformly at random and an $e \in \mathbb{Z}_2$ independently such that $\Pr(e = 1) = \alpha$, and then outputs $(a, (a, s) + e)$.
- Let $U$ be a system that on each invocation chooses $a \in \mathbb{Z}_2^n$ independently and uniformly at random and $b \in \mathbb{Z}_2$ independently and uniformly at random, and outputs $(a, b)$.
- Let $T$ be a system that chooses an $r \in \mathbb{Z}_2^n$ uniformly at random and then, on each invocation chooses $a \in \mathbb{Z}_2^n$ independently and uniformly at random and an $e \in \mathbb{Z}_2$ independently such that $\Pr(e = 1) = \alpha$, and outputs $(a, (a, r) + e)$.

(a) Give a reduction $R$ such that for any $s \in \mathbb{Z}_2^n$, $RS^s \equiv T$ and $RU \equiv U$.

(b) Let $D$ be a distinguisher that distinguishes $T$ and $U$. Give a distinguisher $D'$ such that for any $s \in \mathbb{Z}_2^n$, $\Delta D'(S^s, U) = \Delta D(T, U)$.

(c) Let $B$ be a binary random source that outputs independently chosen bits $b_1, b_2, \ldots$ such that for all $i$, $\Pr(b_i = 1) = p$. Give an algorithm that reads at most $k$ bits output by $B$, and outputs an estimate $p'$ of $p$, such that for any constant $\delta > 0$, $\Pr(p - \delta < p' < p + \delta) = 1 - 2^{-\Omega(k)}$.

HINT: Use a Chernoff Bound.

(d) Let $c > 0$ be a constant. Let $D$ be a distinguisher such that $\Delta D(T, U) \geq c$. Give a distinguisher $D''$ that is allowed at most $2k$ invocations to $D$ and runs in time polynomial in $n$, such that $\Delta D''(S^s, U) \geq 1 - 2^{-\Omega(n)}$.

---

1. here, for $x, y \in \mathbb{Z}_2^n$, $(x, y)$ denotes the inner product of $x$ and $y$
2. the sign $\equiv$ here is to be interpreted as follows: for two systems $A$ and $B$, we write $A \equiv B$ if for an identical input, the outputs of $A$ and $B$ are identically distributed
3. if you do not know these, consult Wikipedia
4. here you can assume that an invocation to $D$ on any system takes constant time