Mathematical Toolkit **Fall 2026**

Homework 3 Solutions

1. **Random Orderings.**
Let Ω be the set of all $n!$ permutations of $\{1, ..., n\}$, and ν be the uniform distribution over Ω. That is, $\nu(\omega) = 1/n!$ for each permutation $\omega \in \Omega$.

   (a) Given permutation $\omega = (\omega_1, \ldots, \omega_n)$, an *inversion* is a pair (i, j) such that $i < j$ but $\omega_i > \omega_j$. Let $X(\omega)$ be the number of inversions in ω. What is $\mathbb{E}[X]$? Be clear about your reasoning.
   Solution: Each of the $\binom{n}{2}$ pairs has probability $1/2$ of being an inversion, so by linearity of expectation, the expected number of inversions is $n(n-1)/4$.

   (b) We say that position i is a *prefix-maximum* in permutation $\omega = (\omega_1, \ldots, \omega_n)$ if $\omega_i > \omega_j$ for all $j < i$. Let $Y(\omega)$ be the number of prefix maxima in ω. What is $\mathbb{E}[Y]$? Explain your reasoning.
   Solution: Let Y_i be the indicator RV for the event that position i is a prefix-maximum. $\mathbb{E}[Y_i] = 1/i$ because we can partition the elements ω by the set of values in $\{\omega_1, ..., \omega_i\}$, and notice that within each partition, exactly a $1/i$ fraction of the elements have the maximum of these values at position i. Finally, by linearity of expectation, we have $\mathbb{E}[Y] = 1 + 1/2 + 1/3 + 1/4 + ... = H_n$.

2. **t-Universal Hashing**.
We say that H is *t-universal over range M* if, for every fixed sequence of t distinct inputs $s_1, s_2, \ldots, s_t$ and for any h chosen at random according to H, the sequence $h(s_1), h(s_2), \ldots, h(s_t)$ is equally likely to be any of the M^t sequences of length t with elements drawn from $\{0, 1, \ldots, M-1\}$. It's easy to see that if H is 2-universal then it is universal.

   Consider strings s of length n from the alphabet $\{0, 1, \ldots, k-1\}$ where $k \geq 2$. One way to construct a hash function over such inputs is to generate a 2-dimensional $n \times k$ table T of b-bit random numbers where $b = \lg(M)$. The first index of $T_{i,j}$ is in the range $[1, n]$ and the second index is in the range $[0, k-1]$. Here the hash function h_T is defined as follows:

$$h_T(s) = \bigoplus_{i=1}^{n} T_{i,s[i]}$$

   where $s[i]$ is the ith character in s and "$\oplus$" represents the xor function of integers represented in binary.

(a) Prove that this construction is *not* 4-universal.

Solution: Consider the following 4 strings of length 2 (so $n = 2$): 00, 11, 01, 10. For any T, these strings have the property that $h_T(00) \oplus h_T(11) \oplus h_T(01) \oplus h_T(10) = 0$ since each of the four entries in T is xor'ed twice. This means that the hash of any one of the strings can be determined from the hash of the other three.

(b) Prove that this construction *is* 3-universal.

Solution: Consider three distinct keys x, y, z. We now consider two cases:

Case 1: there is some index i such that all three keys differ in that index. We now argue as follows. Consider choosing all of T except for $T_{i,*}$. Thus, filling in this row of T will determine the hash codes for x, y, and z. We first fill in T_{i,x_i}: since each value for this entry produces a different value for $h_T(x)$, we have that x is equally likely to hash to all possible b-bit values. Now $h_T(x)$ is fixed. We now fill in T_{i,y_i}: again, since each value for this entry produces a different value for $h_T(y)$, we have that y is equally likely to hash to all possible b-bit values, even conditioning on everything done so far. This means that x and y's hash values are independent. Now $h_T(x)$ and $h_T(y)$ are fixed. We now fill in T_{i,z_i}: again, since each value for this entry produces a different value for $h_T(z)$, we have that z is equally likely to hash to all possible b-bit values, even given everything so far. So all triples of hash values are equally likely.

Case 2: there is no index i such that all three keys differ in that index. In this case, choose some index i such that $x_i \neq y_i$ (this must exist since $x \neq y$). We know z_i matches one of the other two, so say without loss of generality that $z_i = y_i$. Choose some other index j such that $z_j \neq y_j$. We know that x_j matches one of those two values so say without loss of generality that $x_j = y_j$. We can now argue as in Case 1. First fill in all of T except for $T_{i,x_i}, T_{i,y_i}, T_{j,z_j}$. Now, filling in T_{i,x_i} determines x's hash (all equally likely), then filling in T_{i,y_i} determines y's hash (all equally likely, no matter what x hashed to), then filling in T_{j,z_j} determines z's hash value (all equally likely, no matter what x, y hashed to).

3. **MAX-SAT Revisited.**

In class, we saw that if F is a k-CNF formula of m clauses in which every clause has size exactly k (and you are not allowed to repeat variables inside a clause), then there must exist an assignment satisfying at least $\lceil m(1 - 1/2^k) \rceil$ clauses of F. In this question, you will give an efficient deterministic algorithm to find such an assignment.

(a) Prove the following claim: Suppose we have a CNF formula F of m clauses, with m_1 clauses of size 1, m_2 of size 2, etc. ($m = m_1 + m_2 + ...$). Let X be a random variable denoting the number of clauses satisfied in a random assignment. Then $\mathbb{E}[X] = \sum_k m_k(1 - 1/2^k)$.

Solution: Define indicator RVs for the event of each clause being satisfied and use linearity of expectation.

(b) Here is a deterministic algorithm. Compute $E = \mathbb{E}[X]$ using the formula from part (a), and then compute $E_0 = \mathbb{E}[X|x_1 = 0]$ and $E_1 = \mathbb{E}[X|x_1 = 1]$ using a similar formula. Set x_1 based on whichever of E_0 and E_1 is larger. Now, recurse on the formula remaining.

i. Explain how you can efficiently compute E_0 and E_1 in this algorithm.
Solution: To compute E_1, let n_1 be the number of clauses of F containing x_1 (in non-negated form) and let F_1 be the CNF formula produced by deleting all such clauses from F and removing $\neg x_1$ from all clauses containing $\neg x_1$ (shrinking them by 1). Now, we just add n_1 to the expectation formula for the CNF F_1. Similarly for E_0.

ii. Explain why this algorithm is guaranteed to find a solution that satisfies at least $E = \mathbb{E}[X]$ clauses.
Solution: This is because $E = \frac{1}{2}E_0 + \frac{1}{2}E_1$, so the conditional expectation of the number of clauses of F satisfied given our decisions so far never decreases.

Note: this approach is called the "conditional expectation method"