HW6

Lubah Nelson

December 6, 2024

Q1

The coffee maker can be thought of as a finite set of states, $Q = \{q_1, q_2, \dots, q_n\}$, where each state represents a specific configuration of the machine, such as "Power on," "select coffee type," or "brew." Transitions between these states occur through button presses, which form the edges of a directed graph $G = (Q, E)$. For example, pressing "start" in the select coffee type state transitions the machine to the "Brew" state. So, each state q has a set of possible outgoing transitions, and the number of such transitions is denoted as the out-degree, $\deg(q)$.

To turn this graph into a Markov chain, we assign probabilities to each transition. A natural choice is to use a uniform distribution, such that the probability of transitioning from q to any neighbor q' is:

$$P(X_{t+1} = q' \mid X_t = q) = \frac{1}{\deg(q)}.$$

This uniform probability ensures that all available transitions are equally likely. With this setup, the coffee maker's interface becomes a stochastic process, where a sequence of states represents a series of button presses. To ensure comprehensive testing, the Markov chain must satisfy two key properties. First, it must be irreducible, meaning every state is reachable from any other state. This can be achieved by adding reset or back transitions. Second, it must be aperiodic, meaning the chain does not cycle predictably. This can be ensured by including self-loops or transitions that break periodicity. Under these conditions, the Markov chain has a unique stationary distribution π, which describes the long-term likelihood of being in each state. This distribution helps determine whether the test sequence sufficiently explores the machine's functionality.

Once the Markov chain is established, the next step is to generate sequences of button presses that simulate random interactions with the coffee maker. A simple approach is a *random walk*, where we start at an initial state, such as $q_0 =$ "Power On", and at each step, select one of the available transitions uniformly at random. This generates a sequence:

$$q_0 \xrightarrow{\text{button}_1} q_1 \xrightarrow{\text{button}_2} q_2 \cdots q_{T-1} \xrightarrow{\text{button}_T} q_T.$$

The resulting sequence covers different states based on the structure of the Markov chain and the number of steps T. This approach is effective for exploring all reachable states when transitions are uniformly distributed.

However, random walks may not explore critical functionalities. To address this, we can bias the sequence generation using the Metropolis-Hastings algorithm. This method allows us to generate sequences according to a target distribution $P(q)$, which reflects the importance of each state. At each step, starting from the current state q, a candidate state q' is proposed based on a proposal distribution $g(q' \mid q)$, such as uniform over neighbors. The proposed state is accepted with probability:

$$A(q' \mid q) = \min\left\{1, \frac{P(q')\, g(q \mid q')}{P(q)\, g(q' \mid q)}\right\}.$$

A random number $u \sim \text{Uniform}[0, 1]$ is generated, and if $u \leq A(q' \mid q)$, the algorithm transitions to q'; otherwise, it remains at q. This process biases the sequence towards states with higher target probabilities in $P(q)$, ensuring sufficient exploration of critical features.

To evaluate the effectiveness of the generated sequences, we can measure the follwoing:

$$\lambda = -\sum_{q \in Q} \pi(q) \sum_{q' \in Q} P(q' \mid q) \log P(q' \mid q).$$

Higher entropy rates indicate greater randomness and more thorough exploration of the state space. Adjusting the transition probabilities in the chain allows us to balance between exploring all states and focusing on specific functionalities.

0.1 Q2

To prove that $\#3SAT$ is $\#P$-complete, we rely on the fact that $\#CIRCUITSAT$, the problem of counting satisfying inputs for a Boolean circuit, is $\#P$-complete. We reduce $\#CIRCUITSAT$ to $\#3SAT$ by constructing a 3-CNF formula φ from a circuit C such that the number of satisfying assignments of φ equals the number of inputs that make C output 1. This reduction preserves the count of solutions (parsimonious) and is computable in polynomial time, proving that $\#3SAT$ is $\#P$-complete.

The task in $\#CIRCUITSAT$ is to count how many assignments to the input variables $x_1, \ldots, x_n$ satisfy $C(x) = 1$. To reduce $\#CIRCUITSAT$ to $\#3SAT$, we construct a formula φ in 3-CNF that accurately represents the circuit C. For each input variable x_i, φ includes a corresponding variable. Additionally, for each gate in the circuit, we introduce a new variable g_j representing the gate's output and translate the gate's logic into clauses. For instance, if a gate computes a function $g_j = f(a, b)$, where a and b are its inputs, this relationship is described using at most four clauses in 3-CNF form, covering all possible input combinations.

The formula also includes a clause requiring the final output gate g_{out} to be 1, ensuring that satisfying assignments of φ correspond to input assignments for C that produce an output of 1. The resulting formula φ has $n+m$ variables (inputs and gate outputs) and at most $4m$ clauses, where m is the number of gates in C.

Every input assignment $x = (x_1, \ldots, x_n)$ that satisfies $C(x) = 1$ extends to a unique satisfying assignment of φ, including values for all g_j variables that maintain consistency with the circuit. Similarly, every satisfying assignment of φ corresponds to an input assignment that makes $C(x) = 1$. This one-to-one correspondence ensures that the number of solutions is preserved, making the reduction parsimonious.

Since $\#CIRCUITSAT$ is $\#P$-complete, and the reduction from $\#CIRCUITSAT$ to $\#3SAT$ is parsimonious and polynomial-time computable, $\#3SAT$ is also $\#P$-complete. Thus, $\#3SAT$ is as computationally difficult as any problem in $\#P$.

0.2 Q3

We can prove that 3SAT is fixed-parameter tractable (FPT) with respect to the size k of a variable cover C. A variable cover $C \subseteq \{x_1, \ldots, x_n\}$ satisfies the property that for every clause C_i in the formula F, there exists a variable $x_j \in C$ such that $x_j \in C_i$ or $\neg x_j \in C_i$. Let F be a 3SAT formula with m clauses, and let n be the number of variables in F. We aim to show that deciding satisfiability of F can be done in $O(2^k \cdot \text{poly}(n))$.

Let $C = \{y_1, \ldots, y_k\}$. For each assignment $A : C \to \{0, 1\}$, substitute the values of the variables in C into F. After substitution, for any clause C_i, there are three cases: 1. C_i is satisfied and can be removed. 2. C_i is partially simplified (fewer literals remain). 3. C_i is unaffected if it does not contain variables from C.

The substitution produces a simplified formula $F'(A)$, containing at most $n - k$ variables. The remaining formula can be solved in polynomial time, since its size depends only on $n - k$. Enumerating all 2^k assignments to the variables in C yields a total complexity of:

$$T(n, k) = 2^k \cdot (\text{poly}(n) + \text{SAT-solving complexity for } F').$$

Since SAT-solving for F' is polynomial in $n-k$, the total complexity becomes:

$$T(n, k) = O(2^k \cdot \text{poly}(n)).$$

Alternatively, using a graph-based argument, let G be the primal graph of F, where vertices represent variables and edges connect variables appearing together in a clause. The cover C ensures that all clauses are incident to at least one vertex in C. Using tree decomposition, the treewidth of G is at most k. Eliminating variables iteratively and constructing a truth table for each eliminated variable contributes a complexity of:

$$T'(n, k) = O(n \cdot k \cdot 2^k).$$

Both methods confirm that the problem is exponential in k but polynomial in n, meeting the definition of fixed-parameter tractability. Thus, 3SAT is FPT with respect to k, as required.