CSE 632 (Fall 2019): Analysis of Algorithms II : Randomized Algorithms

Lecture 18 (10/25/2019): Random walk and 2-SAT problem

Lecturer: Shi Li *Scriber: Chen Xu*

1 Random walk on a line

Last lecture we see the example of the gambler's problem that can be modeled by a Markov Chain. Now consider a Markov Chain of n nodes, and we are interested in the expected number of iterations for the game to terminate i.e. the expected number of iterations it takes from node 0 to node n. The transition is described as follows:

1: $X_0 \leftarrow 0$
2: **for** $t \in 0, 1, 2, ...$ **do**
3: **if** $X_t = 0$ **then** $X_{t+1} \leftarrow 1$
4: **else if** $X_t = n$ **then** stop
5: **else** $X_{t+1} = \begin{cases} X_{t-1} \text{ with probability } 1/2 \\ X_{t+1} \text{ with probability } 1/2 \end{cases}$

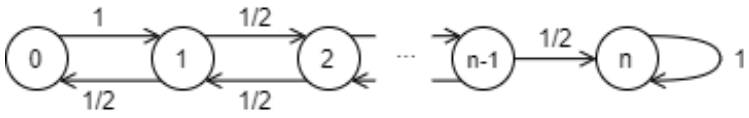

Figure 1: Markov Chain for the random walk problem

We have the following lemma:

Lemma 1. *Let T the number of steps the game takes to end (T is a random variable). Then we have $\mathbb{E}[T] = n^2$.*

Proof. For every $i \in \{0, 1, 2, \cdots, n^2\}$, let $h_i = \mathbb{E}[\text{number of steps the game takes if start with } X_0 = i]$. Then we have $h_n = 0$, $h_0 = h_1 + 1$, and for every other i in the set, we have

$$h_i = \frac{h_{i-1}}{2} + \frac{h_{i+1}}{2} + 1 \tag{1}$$

We then argue about the above equality. If $X_0 = i$, then $\Pr[X_1 = i - 1] = \Pr[X_1 = i + 1] = \frac{1}{2}$. Then conditioned on $X_1 = i - 1$ ($X_1 = i + 1$), the game takes h_{i+1} (h_{i-1}) more steps to end in expectation. Taking the first step into consideration, we have $h_i = \frac{h_{i-1}}{2} + \frac{h_{i+1}}{2} + 1$.

Transform (1) we get:

$$(h_i - h_{i+1}) - (h_{i-1} - h_i) = 2$$

Let $\Delta_i = h_i - h_{i+1}$, we have $\Delta_0 = 1$, $\Delta_i - \Delta_{i-1} = 2$ for every $i = 1, 2, 3, \cdots, n - 1$. Then $\Delta_i = 2i + 1$ for every $i = 1, 2, 3, \cdots, n - 1$.

$$h_0 = (h_0 - h_1) + (h_1 - h_2) + \cdots + (h_{n-1} - h_n) = \sum_{i=1}^{n-1} \Delta_i = 1 + 3 + ... + (2n - 1) = n^2. \quad \square$$

2 Randomized Algorithm for 2-SAT

We introduce the following notations first. Denote n boolean variables as $x_1, x_2, ..., x_n$. For each boolean variable x_i and its negation $\neg x_i$ we call them *literals*. A *clause* is the OR of two literals i.e. $x_i \vee x_j$, $x_i \vee \neg x_j$, $\neg x_i \vee \neg x_j$. A 2-SAT formula is the AND of many clauses. For example $(x_1 \vee x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3)$ and $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$ are 2-SAT formulas.

The 2-SAT problem is to check if there exists an assignment $x \in \{0,1\}^n$ of $\{0,1\}$-values (0 for 'false' and 1 for 'true') to variables s.t. the formula is satisfied, which is equivalent to that all the clauses are satisfied, which is equivalent to that every clause has one satisfied literal. In the above examples, the former is satisfiable and has a truth assignment of (1, 1, 0) while the latter is unsatisfiable.

The randomized algorithm is stated as follows:

1: Let $x \in \{0,1\}^n$ be an arbitrary assignment of the boolean variables
2: **if** all clauses are satisfied **then return** true
3: $counter = 0$
4: **for** $t \leftarrow 1$ to $2mn^2$ **do**
5: choose some clause c is not satisfied arbitrarily
6: choose one of the two literals of c randomly
7: flip the variable for the literal in assignment x
8: **if** all clauses are satisfied **then return** true
9: **return** false

2.1 Correctness

If the boolean formula is unsatisfiable the algorithm will never return true. If the boolean formula is satisfiable the algorithm will output false with some probability. Let us focus on a satisfiable instance. x^* be a satisfying assignment for the formula. Let Y_t be a random variable for the number of agreeing variables at the end of time step t (a variable is agreeing if $x_i = x_i^*$). We fix Y_t and consider the distribution for Y_{t+1}. There can be two cases:

- Case 1: if exactly one of the two variables in c is disagreeing, then we have

$$Y_{t+1} = \begin{cases} Y_t - 1 \text{ with probability } 1/2 \\ Y_t + 1 \text{ with probability } 1/2 \end{cases}$$

- Case 2: if both variables are disagreeing, then

$$Y_{t+1} = Y_t + 1$$

We algorithm will succeed if for some t, Y_t becomes n. (The algorithm may succeed at t even if $Y_t < n$ since there might be some other satisfying assignments.) The expected number of steps for the algorithm to succeed is at most the expected number of steps for the game walk to end in the last section. Indeed, our algorithm has advantages in the following 3 aspects, but we use the random walk game to upper bound the expected number of steps for the algorithm to succeed.

- Initially, we may have $Y_0 > 0$; but the random walk game has $X_0 = 0$.
- In case 2 above, we may have $Y_{t+1} = Y_t + 1$.
- As we argued, the algorithm can succeed even if we have not reached n.

Thus, in expectation, our algorithm will find a satisfying assignment in at most n^2 time steps. We divide the algorithm in to m phases, each containing $2n^2$ iterations. By Markov's inequality, the probability that the algorithm does not succeed in one phase is at least $1/2$. So, overall, the probability that the algorithm succeeds is at least $1 - (1/2)^m$.