MARKOV CHAINS AND RANDOM WALKS

\[(1/4, 1/4, 1/4, 1/4)P^4 = (17/192, 47/384, 737/1152, 43/288)\]

Here the last entry, 43/288, is the required answer.

7.1.1. Application: A Randomized Algorithm for 2-Satisfiability

Recall from Section 6.2.2 that an input to the general satisfiability (SAT) problem is a Boolean formula given as the conjunction (AND) of a set of clauses, where each clause is the disjunction (OR) of literals and where a literal is a Boolean variable or the negation of a Boolean variable. A solution to an instance of a SAT formula is an assignment of the variables to the values True (T) and False (F) such that all the clauses are satisfied. The general SAT problem is NP-hard. We analyze here a simple randomized algorithm for 2-SAT, a restricted case of the problem that is solvable in polynomial time.

For the k-satisfiability (k-SAT) problem, the satisfiability formula is restricted so that each clause has exactly k literals. Hence an input for 2-SAT has exactly two literals per clause. The following expression is an instance of 2-SAT:

\[(x_1 \lor \bar{x}_2) \land (\bar{x}_1 \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor \bar{x}_3) \land (x_4 \lor x_1).\] (7.2)

One natural approach to finding a solution for a 2-SAT formula is to start with an assignment, look for a clause that is not satisfied, and change the assignment so that the clause becomes satisfied. If there are two literals in the clause, then there are two possible changes to the assignment that will satisfy the clause. Our 2-SAT algorithm (Algorithm 7.1) decides which of these changes to try randomly. In the algorithm, \(n\) denotes the number of variables in the formula and \(m\) is an integer parameter that determines the probability that the algorithm terminates with a correct answer.

In the instance given in (7.2), if we begin with all variables set to False then the clause \((x_1 \lor x_2)\) is not satisfied. The algorithm might therefore choose this clause and then select \(x_1\) to be set to True. In this case the clause \((x_4 \lor \bar{x}_1)\) would be unsatisfied and the algorithm might switch the value of a variable in that clause, and so on.

If the algorithm terminates with a truth assignment, it clearly returns a correct answer. The case where the algorithm does not find a truth assignment requires some care, and we will return to this point later. Assume for now that the formula is satisfiable and that the algorithm will actually run as long as necessary to find a satisfying truth assignment.

We are mainly interested in the number of iterations of the while-loop executed by the algorithm. We refer to each time the algorithm changes a truth assignment as a step. Since a 2-SAT formula has \(O(n^2)\) distinct clauses, each step can be executed in \(O(n^2)\) time. Faster implementations are possible but we do not consider them here. Let \(S\) represent a satisfying assignment for the \(n\) variables and let \(A_i\) represent the variable assignment after the \(i\)th step of the algorithm. Let \(X_i\) denote the number of variables in the current assignment \(A_i\) that have the same value as in the satisfying assignment \(S\). When \(X_i = n\), the algorithm terminates with a satisfying assignment. In fact, the algorithm could terminate before \(X_i\) reaches \(n\) if it finds another satisfying assignment, but for our analysis the worst case is that the algorithm only stops when \(X_i = n\). Starting
2-SAT Algorithm:

1. Start with an arbitrary truth assignment.
2. Repeat up to $2mn^2$ times, terminating if all clauses are satisfied:
   (a) Choose an arbitrary clause that is not satisfied.
   (b) Choose uniformly at random one of the literals in the clause and switch the value of its variable.
3. If a valid truth assignment has been found, return it.
4. Otherwise, return that the formula is unsatisfiable.

**Algorithm 7.1:** 2-SAT algorithm.

with $X_i < n$, we consider how $X_i$ evolves over time, and in particular how long it takes before $X_i$ reaches $n$.

First, if $X_i = 0$ then, for any change in variable value on the next step, we have $X_{i+1} = 1$. Hence

$$\Pr(X_{i+1} = 1 \mid X_i = 0) = 1.$$  

Suppose now that $1 \leq X_i \leq n - 1$. At each step, we choose a clause that is unsatisfied. Since $S$ satisfies the clause, that means that $A_i$ and $S$ disagree on the value of at least one of the variables in this clause. Because the clause has no more than two variables, the probability that we increase the number of matches is at least $1/2$: the probability that we increase the number of matches could be 1 if we are in the case where $A_i$ and $S$ disagree on the value of both variables in this clause. It follows that the probability that we decrease the number of matches is at most $1/2$. Hence, for $1 \leq j \leq n - 1$,

$$\Pr(X_{i+1} = j + 1 \mid X_i = j) \geq 1/2;$$

$$\Pr(X_{i+1} = j - 1 \mid X_i = j) \leq 1/2.$$  

The stochastic process $X_0, X_1, X_2, \ldots$ is not necessarily a Markov chain, since the probability that $X_i$ increases could depend on whether $A_i$ and $S$ disagree on one or two variables in the unsatisfied clause the algorithm chooses at that step. This, in turn, might depend on the clauses that have been considered in the past. However, consider the following Markov chain $Y_0, Y_1, Y_2, \ldots$:

$$Y_0 = X_0;$$

$$\Pr(Y_{i+1} = 1 \mid Y_i = 0) = 1;$$

$$\Pr(Y_{i+1} = j + 1 \mid Y_i = j) = 1/2;$$

$$\Pr(Y_{i+1} = j - 1 \mid Y_i = j) = 1/2.$$  

The Markov chain $Y_0, Y_1, Y_2, \ldots$ is a pessimistic version of the stochastic process $X_0, X_1, X_2, \ldots$ in that, whereas $X_i$ increases at the next step with probability at least $1/2$, $Y_i$ increases with probability exactly $1/2$. It is therefore clear that the expected time to reach $n$ starting from any point is larger for the Markov chain $Y$ than for the
process \( X \), and we use this fact hereafter. (A stronger formal framework for such ideas is developed in Chapter 11.)

This Markov chain models a random walk on an undirected graph \( G \). (We elaborate further on random walks in Section 7.4.) The vertices of \( G \) are the integers \( 0, \ldots, n \) and, for \( 1 \leq i \leq n - 1 \), node \( i \) is connected to node \( i - 1 \) and node \( i + 1 \). Let \( h_j \) be the expected number of steps to reach \( n \) when starting from \( j \). For the 2-SAT algorithm, \( h_j \) is an upper bound on the expected number of steps to fully match \( S \) when starting from a truth assignment that matches \( S \) in \( j \) locations.

Clearly, \( h_n = 0 \) and \( h_0 = h_1 + 1 \), since from \( h_0 \) we always move to \( h_1 \) in one step. We use linearity of expectations to find an expression for other values of \( h_j \). Let \( Z_j \) be a random variable representing the number of steps to reach \( n \) from state \( j \). Now consider starting from state \( j \), where \( 1 \leq j \leq n - 1 \). With probability \( 1/2 \), the next state is \( j - 1 \), and in this case \( Z_j = 1 + Z_{j-1} \). With probability \( 1/2 \), the next step is \( j + 1 \), and in this case \( Z_j = 1 + Z_{j+1} \). Hence

\[
E[Z_j] = E\left[\frac{1}{2}(1 + Z_{j-1}) + \frac{1}{2}(1 + Z_{j+1})\right].
\]

But \( E[Z_j] = h_j \) and so, by applying the linearity of expectations, we obtain

\[
h_j = \frac{h_{j-1} + 1}{2} + \frac{h_{j+1} + 1}{2} = \frac{h_{j-1}}{2} + \frac{h_{j+1}}{2} + 1.
\]

We therefore have the following system of equations:

\[
\begin{align*}
h_n & = 0; \\
h_j & = \frac{h_{j-1}}{2} + \frac{h_{j+1}}{2} + 1, \quad 1 \leq j \leq n - 1; \\
h_0 & = h_1 + 1.
\end{align*}
\]

We can show inductively that, for \( 0 \leq j \leq n - 1 \),

\[
h_j = h_{j+1} + 2j + 1.
\]

It is true when \( j = 0 \), since \( h_1 = h_0 - 1 \). For other values of \( j \), we use the equation

\[
h_j = \frac{h_{j-1}}{2} + \frac{h_{j+1}}{2} + 1
\]

to obtain

\[
h_{j+1} = 2h_j - h_{j-1} - 2
\]

\[
= 2h_j - (h_j + 2(j - 1) + 1) - 2
\]

\[
= h_j - 2j - 1,
\]

using the induction hypothesis in the second line. We can conclude that

\[
h_0 = h_1 + 1 = h_2 + 1 + 3 = \cdots = \sum_{i=0}^{n-1} 2i + 1 = n^2.
\]
An alternative approach for solving the system of equations for the \( h_j \) is to guess and verify the solution \( h_j = n^2 - j^2 \). The system has \( n + 1 \) linearly independent equations and \( n + 1 \) unknowns, and hence there is a unique solution for each value of \( n \). Therefore, if this solution satisfies the foregoing equations then it must be correct. We have \( h_n = 0 \). For \( 1 \leq j \leq n - 1 \), we check

\[
h_j = \frac{n^2 - (j - 1)^2}{2} + \frac{n^2 - (j + 1)^2}{2} + 1
\]

and

\[
h_0 = (n^2 - 1) + 1
\]

\[
= n^2.
\]

Thus we have proven the following fact.

**Lemma 7.1:** Assume that a 2-SAT formula with \( n \) variables has a satisfying assignment and that the 2-SAT algorithm is allowed to run until it finds a satisfying assignment. Then the expected number of steps until the algorithm finds an assignment is at most \( n^2 \).

We now return to the issue of dealing with unsatisfiable formulas by forcing the algorithm to stop after a fixed number of steps.

**Theorem 7.2:** The 2-SAT algorithm always returns a correct answer if the formula is unsatisfiable. If the formula is satisfiable, then with probability at least \( 1 - 2^{-m} \) the algorithm returns a satisfying assignment. Otherwise, it incorrectly returns that the formula is unsatisfiable.

**Proof:** It is clear that if there is no satisfying assignment then the algorithm correctly returns that the formula is unsatisfiable. Suppose the formula is satisfiable. Divide the execution of the algorithm into segments of \( 2n^2 \) steps each. Given that no satisfying assignment was found in the first \( i - 1 \) segments, what is the conditional probability that the algorithm did not find a satisfying assignment in the \( i \)th segment? By Lemma 7.1, the expected time to find a satisfying assignment, regardless of its starting position, is bounded by \( n^2 \). Let \( Z \) be the number of steps from the start of segment \( i \) until the algorithm finds a satisfying assignment. Applying Markov's inequality.

\[
\Pr(Z > 2n^2) \leq \frac{n^2}{2n^2} = \frac{1}{2}.
\]

Thus the probability that the algorithm fails to find a satisfying assignment after \( m \) segments is bounded above by \( (1/2)^m \).

**7.1.2. Application: A Randomized Algorithm for 3-Satisfiability**

We now generalize the technique used to develop an algorithm for 2-SAT to obtain a randomized algorithm for 3-SAT. This problem is NP-complete, so it would be rather