CSC2420: Algorithm Design, Analysis and Theory
Topic: Markov Chain and Randomized Rounding **Date:** Nov 24, 2010
Lecturer: Prof. Allan Borodin **Scribe:** Chun Kong Yung

22.1 Outline

In the first half of this lecture, we will cover some topics concerning Markov chains and random walks. In the second half, we will discuss the exact Max-k-SAT problem. Then we will see how to apply randomized rounding to a LP relaxation for (arbitrary) Max-Sat.

22.2 Markov Chain

22.2.1 The Basics

A finite Markov chain M is a discrete-time random process defined over a set of states S and a matrix $P = \{P_{ij}\}$ of transition probabilities. Denote by X_t the state of the Markov chain at time t. It is a memoryless process that the future behavior of a Markov chain depends only on its current state:

$$P_{ij} = Pr[X_{t+1} = j | X_t = i].$$

Given an initial state i, denote by r_{ij}^t the probability that the first time the process reaches state j at time t:

$$r_{ij}^t = Pr[X_t = j \text{ and } X_s \neq j \text{ for } 1 \leq s \leq t-1 | X_0 = i].$$

Denote by f_{ij} the probability that state j is reachable from initial state i:

$$f_{ij} = \sum_{t>0} r_{ij}^t.$$

Denote by h_{ij} the expected number of steps to reach state j starting from state i (hitting time):

$$h_{ij} = \sum_{t>0} t \cdot r_{ij}^t.$$

Finally, denote by c_{ij} the commute time, which is the expected number of steps to reach state j starting from state i, and then return to i from j:

$$c_{ij} = h_{ij} + h_{ji}.$$

A state i is said to be *transient* if $f_{ii} < 1$; and it is said to be *persistent* if $f_{ii} = 1$.

Denote by $q^t = (q_1^t, q_2^t, \ldots, q_n^t)$ the state probability vector (the distribution of the chain at time t), to be a row vector whose i-th component is the probability that the Markov chain is in state i at time t. A distribution π is a *stationary* distribution for a Markov chain with transition matrix P if $\pi = \pi P$.

Define the underlying directed graph of a Markov chain as follows: each vertex in the graph corresponds to each state of Markov chain and there is an directed edge from vertex i to vertex j iff $P_{ij} > 0$. A Markov chain is *irreducible* if its underlying graph consists of a single strongly connected component. We end this section by the following theorem.

Theorem 22.2.1 *For any finite, irreducible and aperiodic Markov chain,*

(i) There exists a unique stationary distribution π.

(ii) All states i have $h_{ii} < \infty$, and $h_{ii} = 1/\pi_i$.

22.2.2 Uniform Random Walk on Graphs

Let $G = (V, E)$ be a connected, non-bipartite, undirected graph with $|V| = n$ and $|E| = m$. A uniform random walk induces a Markov chain M_G as follows: the states of M_G are the vertices of G; and for any $u, v \in V$, $P_{uv} = 1/deg(u)$ if $(u, v) \in E$, and $P_{uv} = 0$ otherwise.

Denote by $(d_1, d_2, \ldots, d_n)$ the vertex degrees. M_G has a stationary distribution $(d_1/2m, \ldots, d_n/2m)$. If G is a k-regular graph then it has a uniform stationary distribution.

Denote by $C_u(G)$ the expected time to visit every vertex, starting from u. Denote by $C(G) = \max_u C_u(G)$ the *cover time* of G.

Theorem 22.2.2 (Theorem 6.8 in [1]) $C(G) \leq 2m(n-1)$.

As an outline of the proof, consider a spanning tree of the graph G. There are $n-1$ edges, and it takes at most $2m$ steps to commute the end vertices of an edge (by the lemma below). Hence, it takes at most $2m(n-1)$ steps to visit all vertices.

Lemma 22.2.3 (Lemma 6.5 in [1]) $C_{ij} = h_{ij} + h_{ji} \leq 2m$ *for every* $(i, j) \in E$.

Theorem 22.2.2 provides an application of random walks in checking connectivity. Given an undirected graph G and a vertex pair (s, t) in G, the undirected s-t connectivity (USTCON) problem asks whether s and t are in the same connected component. Since a uniform random walk is a memoryless process, it takes only log space for a walk of length $2m(n-1)$. Hence, USTCON can be solved in randomized logarithmic-space polynomial-time, i.e. $USTCON \in RLP$ (Theorem 6.11 in [1]).

We end this section with a randomized algorithm for 2-SAT. Start with any assignment of 2-SAT, pick an unsatisfied clause and flip one of its (randomly chosen) variables; repeat this for $m = O(n^2)$ times. Assume the input instance is satisfiable and consider a satisfying assignment $\tau*$. If the current assignment matches $\tau*$ at k variables, then after flipping one variable, it matches $\tau*$ at either $k+1$ or $k-1$ variables. In the worst case, each case happens with $1/2$ probability. Hence, the worst-case behavior of this algorithm corresponds to a random walk on a line. By Theorem 22.2.2, the expected cover time is bounded by $2n(n-1) = O(n^2)$.

22.2.3 Expander Graphs

Consider a d-regular bipartite graph $G(X, Y; E)$ with $|X| = |Y| = n/2$. G is a (n, d, c)-expander if for all $S \subseteq X$,

$$|N(S)| \geq (1 + c \cdot (1 - \frac{2|S|}{n}))|S|.$$

Let A be the adjacency matrix of G. If G is a d-regular graph, then $d = \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$, where λ_i's are eigenvalues of A. G is an expander if and only if $(\lambda_1 - \lambda_2)/d$ is large enough (Theorem 6.16 and 6.17 in [1]).

22.2.4 Rapidly Mixing Markov Chain

Denote by q^t the state probability vector of a Markov chain defined by matrix M at time t. Denote π the stationary distribution of M. The *relative pointwise distance* of the Markov chain at time t is defined as

$$\Delta(t) = \max_i \frac{|q_i^t - \pi_i|}{\pi_i}.$$

The change in $\Delta(t)$ with respect to t measures the rate of convergence to the stationary distribution.

Consider a (n, d, c)-expander G with adjacent matrix $A = \{a_{ij}\}$. Let $P = A/d$ and $Q = (I + P)/2$, where I is the identity matrix. The Markov chain defined by Q still has G as its underlying graph. The eigenvalues of Q are given by

$$\lambda_i' = \frac{1 + \lambda_i/d}{2},$$

where λ_i's are eigenvalues of A. The following theorem states that the relative pointwise distance for a random walk on an (n, d, c)-expander converges to zero at exponential rate.

Theorem 22.2.4 (Theorem 6.21 in [1]) $\Delta(t) \leq n^{1.5}(\lambda_2')^t$.

The rapidly mixing property means that after a small number of steps, the random walk is expected to be close to a uniformly distributed vertex, independent of the choice of the initial vertex. Hence, we can generate random strings from random walks on an expander graph. This allows us to perform probability amplification by using a suitable random walk (refer to Section 6.8 in [1] for more details).

22.3 Randomized Rounding

22.3.1 Max-k-SAT

Let $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ and $C_i = l_i^1 \vee l_i^2 \cdots \vee l_i^k$ is a clause having k distinct variables. The goal of Max-k-SAT is to find an assignment τ that satisfies as many clauses in F as possible. In a weighted version, each clause C_i has a weight w_i and the goal is to maximize the total weight of satisfied clauses. This problem is NP-hard for $k \geq 2$.

As a simple randomized algorithm, we choose τ randomly, i.e. assign $\tau(x_i)$ to True and False with equal probability 1/2. The probability that a clause is satisfied is hence $\frac{2^k-1}{2^k}$. The expected weight

a clause C_j contributing in τ is

$$E_\tau[C_j] = \frac{2^k - 1}{2^k} \cdot w_j.$$

The expected total weight of satisfied clauses in τ is

$$E_\tau[F] = \frac{2^k - 1}{2^k} \cdot \sum_{j=1}^{m} w_j \geq \frac{2^k - 1}{2^k} OPT.$$

We can de-randomize this algorithm by using conditional expectation. First, notice that

$$E_\tau[F] = \frac{1}{2} E_{\tau_1}[F|_{x_1=\text{True}}] + \frac{1}{2} E_{\tau_0}[F|_{x_1=\text{False}}].$$

Hence, one of $E_{\tau_1}[F|_{x_1=\text{True}}]$ and $E_{\tau_0}[F|_{x_1=\text{False}}]$ is at least $\frac{2^k-1}{2^k} OPT$. Therefore, we can set x_1 to the one getting better total weight. This gives a (deterministic) priority algorithm.

For any $k \geq 3$, it is NP-hard to approximate Max-k-SAT better. However, we can do better by using vector program rounding when $k = 2$. Note also that for non exact Max-k-Sat (when there can also be unit clauses), we only get a 1/2 approximation. Using randomized rounding of an LP we can do better as we shall see in the next section.

22.3.2 Linear Programming Relaxation and Rounding

First, we give an integer programming formulation of Max-k-SAT. Here, each z_j corresponds to a clause C_j and each y_i corresponds to a variable x_i.

$$\begin{array}{lll} \max & \sum_{j=1}^{m} w_j \cdot z_j & \\ \textit{subject to} & \sum_{x_i \in C_j} y_i + \sum_{\overline{x_i} \in C_j} (1 - y_i) \geq z_j, & \forall j \\ & y_i \in \{0, 1\}, & \forall i \\ & z_i \in \{0, 1\}, & \forall j \end{array}$$

Then, we relax the integer constraint to give a linear programming formulation. In other words, we replace constraints $y_i \in \{0, 1\}$ and $z_j \in \{0, 1\}$ by $0 \leq y_i \leq 1$ and $0 \leq z_j \leq 1$ for all i and j. Now, we can solve the linear program in polynomial time by the well-known methods. This gives a solution $(\{y_i^*\}, \{z_j^*\})$ with objective value at least OPT. However, y_i^* and z_j^* may be fractional as we relaxed the constraints.

Hence, we have to round the solution to 0-1 values. We assign x_i to 1 (True) with probability y_i, and 0 (False) with probability $1 - y_i$. Consider (with out loss of generality by renaming) a clause $C_j = x_1 \vee \cdots \vee x_k$. The probability that the clause C_j is not satisfied is

$$\prod_{i=1}^{k} (1 - y_i).$$

By arithmetic-geometric mean, the probability is upper bounded by

$$[\frac{1}{k}\sum_{i=1}^{k}(1-y_i)]^k = [1-\sum_{i=1}^{k} y_i/k]^k \leq [1-z_j/k]^k.$$

Hence, the expected total weight of satisfied clauses in the rounded solution is at least

$$\sum_{j=1}^{m}(1-[1-z_j/k]^k)\cdot w_j \geq (1-\frac{1}{e})\cdot OPT.$$

This improves the 1/2-approximation for non exact Max-2-Sat obtained by the naive randomized/priority algorithm. By maximizing this LP approach with the naive randomized algorithm, the approximation ratio is 3/4 fo arbitrary Max-Sat. The LP method (and its combination with the naive method) can be de-randomized but the result is no longer a priority algorithm.

22.3.3 Vector Programming Relaxation

Given a weighted undirected graph, the goal of Max-Cut problem is to find a vertex set S such that the total weight of edges crossing S and $\overline{S}$ is maximized. Denote by $\{x_i\}$ the set of vertices. Let $y_i = 1$ if $x_i \in S$ and $y_i = -1$ otherwise. We have the following quadratic programming formulation.

$$\begin{array}{ll} \max & \sum \frac{1}{2} w_{ij}(1-y_i y_j) \\ & y_i \in \{-1,1\} \qquad \forall i \end{array}$$

We can relax it to a vector programming formulation - replace y_i by a vector $v_i \in \mathbb{R}^n$ with $||v_i|| = 1$; and replace $y_i y_j$ by inner product $v_i \cdot v_j = \sum_{k=1}^{n} v_i(k)v_j(k)$. This is a relaxation because $v_i = (y_i, 0, 0, \ldots, 0)$ is a special case of $||v_i|| = 1$.

$$\begin{array}{ll} \max & \sum \frac{1}{2} w_{ij}(1-v_i \cdot v_j) \\ & ||v_i|| = 1 \qquad \forall i \end{array}$$

A vector program (in this case, a semi-definite program) can be solved (almost exactly) in polynomial time. Then we apply randomized rounding to get an integral approximate solution.

References

[1] Randomized Algorithms. Rajeev Motwani and Prabhakar Raghavan. Cambridge University Press, 1995.