Two Randomized Algorithms

George Voutsadakis[1]

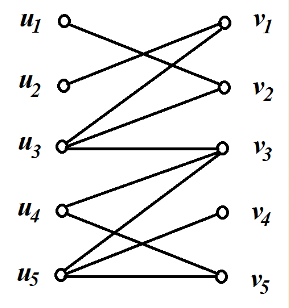
[1]Mathematics and Computer Science
Lake Superior State University

Seminar Presentation

Lake Superior State University

The Bipartite Matching Problem

- A **bipartite graph** $G = (U, V, E)$ consists of
  - two sets of vertices $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$;
  - A set $E \subseteq U \times V$ of edges.



- A **perfect matching** in a bipartite graph G is a subset $M \subseteq E$, such that, for any two edges $(u, v), (u', v') \in M$, $u \neq u'$ and $v \neq v'$.
  Equivalently, a perfect matching may be viewed as a permutation π of $\{1, 2, \ldots, n\}$, such that $(u_i, v_{\pi(i)}) \in E$, for all $i = 1, \ldots, n$.

BipartiteMatching: Given a bipartite graph $G = (U, V, E)$, does it have a perfect matching?

Matrices and Determinants

- Given a bipartite graph G, consider the $n \times n$ matrix A^G whose (i,j)-th element is a variable x_{ij}, if $(u_i, v_j) \in E$, and zero otherwise.
- The **determinant** of A^G is defined as

$$\det(A^G) = \sum_{\pi} \sigma(\pi) \prod_{i=1}^{n} A^G_{i,\pi(i)},$$

  - π ranges over all permutations of n elements;
  - $\sigma(\pi)$ is 1 if π is composed of an even number of transpositions, and -1 otherwise.
- Note the following:
  - The only nonzero terms in this sum are those that correspond to perfect matchings π.
  - Since all variables appear once, all of these terms are different monomials, and hence they do not cancel in the end result.

So, G has a perfect matching iff $\det(A^G)$ is not identically zero.

Gaussian Elimination

- The simplest and oldest method to compute determinants is Gaussian elimination.

$$\begin{bmatrix} 1 & 3 & 2 \\ 1 & 7 & -2 \\ -1 & 3 & -2 \end{bmatrix} \xrightarrow[r_3 \leftarrow r_3 + r_1]{r_2 \leftarrow r_2 - r_1} \begin{bmatrix} 1 & 3 & 2 \\ 0 & 4 & -4 \\ 0 & 6 & 0 \end{bmatrix} \xrightarrow{r_3 \leftarrow r_3 - \frac{3}{2} r_2} \begin{bmatrix} 1 & 3 & 2 \\ 0 & 4 & -4 \\ 0 & 0 & 6 \end{bmatrix}$$

  It follows that $\det(A) = 1 \cdot 4 \cdot 6 = 24$.
- For numerical matrices, this algorithm runs in polynomial-time.
- Unfortunately, for symbolic matrices, its worst-case running time becomes exponential.

The Symbolic Identity Problem

- But we are not interested in actually evaluating the symbolic determinant!

  We just need to tell whether it is identically zero or not.
- So we can pursue the following idea:
  - We substitute arbitrary integers for the variables.
  - Then we obtain a numerical matrix, whose determinant we can calculate in polynomial time by Gaussian elimination.
    - If this determinant is not zero, then the symbolic determinant cannot be identically zero!
    - But the numerical determinant may be zero, although the symbolic one was not.

      This happens if we stumble upon one of the roots of the determinant (seen as a polynomial).

How Unlucky Can We Be?

Lemma

Let $p(x_1, \ldots, x_m) \not\equiv 0$ be a polynomial in m variables each of degree at most d in it, and let $M > 0$ be an integer. Then the number of m-tuples $(x_1, \ldots, x_m) \in \{0, 1, \ldots, M-1\}^m$ such that $p(x_1, \ldots, x_m) = 0$ is at most mdM^{m-1}.

- The proof is by induction on m, the number of variables.
- When $m = 1$ the lemma says that no polynomial of degree $\leq d$ can have more than d roots.

  Suppose the result is true for $m - 1$ variables.

  Write p as a polynomial in x_m, whose coefficients are polynomials in $x_1, \ldots, x_{m-1}$.

  E.g.,
  $$\begin{aligned} & x_1^3x_2^2 + x_1^3x_3^3 + x_1^2x_2x_3^2 + x_1^2x_2x_3 + x_1x_2^2x_3 + x_1x_3^3 \\ & \quad = (x_1^3 + x_1)x_3^3 + (x_1^2x_2)x_3^2 + (x_1^2x_2 + x_1x_2^2)x_3 + (x_1^3x_2^2). \end{aligned}$$

The Induction Step

- If this polynomial evaluated at some integer point is zero, then
  - either the highest-degree coefficient of x_m in p is zero,...
    By induction this can occur for at most $(m-1)dM^{m-2}$ values of $x_1, \ldots, x_{m-1}$.
    For each such value, p will be zero for at most M values of x_m.
    Hence, for at most $(m-1)dM^{m-1}$ values of $x_1, \ldots, x_m$.
  - ... or it is not.
    We have a polynomial of degree $\leq d$ in x_m which can have at most d roots for each combination of values of $x_1, \ldots, x_{m-1}$.
    So we get at most dM^{m-1} new roots of p.

Adding these, we upper bound the total number of roots of p:

$$(m-1)dM^{m-1} + dM^{m-1} = mdM^{m-1}.$$

A Monte Carlo Algorithm

- The Lemma alllows the following randomized algorithm for deciding if a graph G has a perfect matching.
- We denote by $A^G(x_1, \ldots, x_m)$ the matrix A^G with its m variables.
- $\det(A^G(x_1, \ldots, x_m))$ has degree at most one in each of the variables.

  Choose m random integers $i_1, \ldots, i_m$ between 0 and $M = 2m - 1$.
  Compute the determinant $\det(A^G(i_1, \ldots, i_m))$ by Gaussian elimination.
  If $\det(A^G(i_1, \ldots, i_m)) \neq 0$ then "G has a perfect matching";
  If $\det(A^G(i_1, \ldots, i_m)) = 0$ then "G probably has no perfect matching".
- This is a **polynomial Monte Carlo algorithm**:
  - If the algorithm finds that a matching exists, its decision is reliable.
  - But if the algorithm answers "probably no matching", then there is a possibility of a false negative.
- If G has a matching, the probability of a false negative answer is

$$P(\text{hitting a } 0) \leq \frac{m(2m)^{m-1}}{(2m)^m} = \frac{m}{2m} = \frac{1}{2}.$$

Amplification

- By taking M much larger than md we could reduce the probability of a false negative answer as much as desired (at the expense of applying Gaussian elimination to a matrix with larger numbers).
- However, there is a much more widely applicable (and more appealing) way of reducing the chance of false negative answers:

  Perform many independent experiments.
- We repeat k times the evaluation of the determinant of a symbolic matrix, each time with independently chosen random integer values for the variables in the range $0, \ldots, 2m-1$.
  - If the answer always comes out zero, then our confidence on the outcome that G has no perfect matching is boosted to $1 - (\frac{1}{2})^k$.
  - If the answer is different from zero even once, then we know that a perfect matching exists.

The Satisfiability Problem

- Let $x_1, \ldots, x_n$ be Boolean variables.
- A **literal** is one of $x_1, \ldots, x_n$ or $\neg x_1, \ldots, \neg x_n$.
- A **clause** c is a disjunction $c = \ell_1 \vee \cdots \vee \ell_k$, where ℓ_i is a literal.
- A **CNF formula** is a formula $\varphi = \bigwedge_{i=1}^m c_i$, where c_i is a clause, say

$$c_i = \ell_{i1} \vee \cdots \vee \ell_{ik_i}.$$

- φ is **satisfiable** if there exists an assignment $\tau : \{x_1, \ldots, x_n\} \to \{0, 1\}$ of Boolean values to its variables, such that $\tau(\varphi) = 1$.

SAT: Given a CNF formula φ, is φ satisfiable?

Random Walk Algorithm for Satisfiability

- Consider the following randomized algorithm for SAT:

  Start with any truth assignment τ;
  Repeat the following *r* times:
      If there is no unsatisfied clause, reply "formula is satisfiable";
      else
          take any unsatisfied clause (all of its literals are false under τ);
          Pick any of these literals at random and flip it, updating τ.
  Reply "formula is probably unsatisfiable".

- We will fix the value of parameter *r* later.
- We call this the **random walk algorithm**.

Performance of the Algorithm

- If the given expression is unsatisfiable, then our algorithm is "correct":

  It concludes that the expression is "probably unsatisfiable".
- But if the expression is satisfiable, we may have a false negative.
- If we allow exponentially many repetitions we will eventually find a satisfying assignment with very high probability.
- If r is only allowed to be polynomial in the number of Boolean variables, there are simple satisfiable instances of 3-SAT (3 literals allowed per clause) for which the "random walk algorithm" performs badly.
- When applied to 2-SAT (2 literals allowed per clause) the random walk algorithm performs quite decently.

Performance for 2-SAT

Theorem

Suppose that the random walk algorithm with $r = 2n^2$ is applied to any satisfiable instance of 2-SAT with n variables. Then the probability that a satisfying truth assignment will be discovered is at least $\frac{1}{2}$.

- Let $\hat{\tau}$ be a truth assignment that satisfies the given 2-SAT instance.
  Let $t(i)$ denote the expected number of repetitions of the flipping step until a satisfying truth assignment is found, assuming that our starting truth assignment τ differs from $\hat{\tau}$ in exactly i values.
  We know that $t(0) = 0$.
  Also we need not flip when we are at another satisfying assignment.
  Otherwise, we must flip at least once.
  When we flip, we choose among the two literals of a clause.
  At least one of these two literals is true under $\hat{\tau}$.
  Thus, when flipping, we have at least $\frac{1}{2}$ chance of moving closer to $\hat{\tau}$.

Writing an Inequality

- For $0 < i < n$ we can write the inequality:

$$t(i) \leq \frac{1}{2}(t(i-1) + t(i+1)) + 1,$$

  where the added unit stands for the flip just made.

  It is an inequality because the situation could be brighter:
  - Perhaps the current τ also satisfies the expression;
  - Perhaps it differs from $\hat{\tau}$ in both literals, not just the guaranteed one.
  - Also $t(n) \leq t(n-1) + 1$, since at $i = n$ we can only decrease i.

  Consider the situation, where the relation holds as an equation.
  - This way we give up the occasional chance of stumbling upon another satisfying truth assignment, or a clause where τ and $\hat{\tau}$ differ in both literals.
  - It is clear that this can only increase the $t(i)$'s.

Dealing with an Equation

- We define the function $x(i)$ to obey
  - $x(0) = 0$;
  - $x(n) = x(n-1) + 1$;
  - $x(i) = \frac{1}{2}(x(i-1) + x(i+1)) + 1$.

  The $x(i)$'s are easy to calculate and $x(i) \geq t(i)$ for all i.

  We have a "one-dimensional random walk with a reflecting and an absorbing barrier" or a "gambler's ruin against the sheriff".
  - If we add all equations on the $x(i)$'s together, we get $x(1) = 2n - 1$;
  - Then solving the $x(1)$-equation for $x(2)$ we get $x(2) = 4n - 4$;
  - Continuing like this $x(i) = 2in - i^2$.

  As expected, the worst starting i is n, with $x(n) = n^2$.

  We have thus proved that the expected number of repetitions needed to discover a satisfying truth assignment is $t(i) \leq x(i) \leq x(n) = n^2$.

Bounding the Probability of Failure

- No matter where we start, our expected number of steps is $\leq n^2$.
- The following lemma, with $k = 2$, completes the proof.

Lemma

If x is a random variable taking nonnegative integer values, then for any $k > 0$,

$$\mathrm{P}\left[x > k \cdot E(x)\right] < \frac{1}{k}.$$

- Let p_i be the probability that $x = i$.

$$E(x) = \sum_i ip_i = \sum_{i \leq kE(x)} ip_i + \sum_{i > kE(x)} ip_i > kE(x)\mathrm{P}[x > kE(x)].$$

- Hence, the random walk algorithm with $r = 2n^2$ is a polynomial Monte Carlo algorithm for 2-SAT:
  - There there are no false positives;
  - The probability of a false negative is less than $\frac{1}{2}$.

Thank you!

- In closing...

Thank you for your Attention!!