Exact and Approximate Counting

Seminar on Random Walks on Graphs

Shai Vardi

December 7, 2009

Overview

- Definition of #**P**
- The relationship between sampling and approximate counting
- Approximate counting Q-coloring
- General approximate counting in BPP^NP

Reminder - NP

- A predicate $\vartheta : \sum^* \to \{0,1\}$ belongs to the class **NP** if there exists a polynomial time "witness-checking" predicate $\psi : \sum^* \times \sum^* \to \{0,1\}$ and a polynomial *p* such that for all $x \in \sum^*$,

$$\vartheta(x) \Leftrightarrow \exists w \in \sum^* . \psi(x, w) \wedge |w| \leq p(|x|)$$

The class #P

- We want to extend the NP framework to *counting problems,* which can be viewed as functions

  $$f : \Sigma^* \to \mathbb{N}$$

  which map (encodings of) problem instances to natural numbers.

Definition - #P

- A counting problem $f : \sum^* \rightarrow \mathbb{N}$ belongs to the class #P if there exists a polynomial time predicate $\psi : \sum^* \times \sum^* \rightarrow \{0,1\}$ and a polynomial *p* such that for all $x \in \sum^*$,

$$f(x) = \left| \{ w \in \sum^* . \psi(x, w) \wedge \left| w \right| \leq p(\left| x \right|) \} \right|$$

#P completeness

- A counting problem $f : \Sigma^* \to \mathbb{N}$ belongs to the complexity class **FP** if it is computable by a deterministic Turing machine in polynomial time.
- A function f is #P-hard if every function in #P is Turing reducible to f.
- (A function g is Turing reducible to f if there is a Turing machine with an oracle for f that computes g in time polynomial with the input size.)

#P completeness

- A function f is #P-complete if it is #P-hard and in addition $f \in$ #P.

- Examples of #P-complete problems:
  - # SAT
  - # Hamiltonian cycles
  - 0,1-Perm (Valiant)
  - # Bipartite perfect matchings

Toda's Theorem

- Toda's Theorem states that $PH \subseteq P^{\#P}$

i.e. if we can count in polynomial time, the hierarchy collapses.

Approximation algorithms

- There are two kinds of approximation: additive and multiplicative. For a function $f : \Sigma^* \to \mathbb{N}$,
- A function *g* is said to approximate *f* with multiplicative error ε if: $\forall x : \left| g(x) - f(x) \right| \leq \varepsilon \cdot f(x) \left|\right.$
- A function *g* is said to approximate *f* with additive error δ if: $\forall x : \left| g(x) - f(x) \right| \leq \delta$
- Which is better?

Approximate counting

- First, an example: approximate counting of q-colorings
- Theorem 1: There exists a randomized polynomial time approximation scheme for #q-coloring, where $d \geq 2$ and $q \geq 2d$.
- Why doesn't the naïve algorithm work?

Proof of Theorem 1-
A general description of the algorithm

- Let G=(V,E) a graph, where |V|=m and |E|=n.
- Enumerate the edges of G from 1 to n:
  $\{e_1, e_2, \ldots, e_n\}$
- Define $G_0, G_1, \ldots, G_n$ as follows:

$$G_j = (V, \{e_1, \ldots, e_j\})$$

  ($G_0 = (V, \varnothing)$)

- Denote by Z_k the number of q-colorings of G_k.
- Note that Z_n is the number we wish to compute.

Proof of Theorem 1- continued

- We write

$$Z_n = \frac{Z_n}{Z_{n-1}} \times \frac{Z_{n-1}}{Z_{n-2}} \times \ldots \times \frac{Z_1}{Z_0} \times Z_0$$

- Since G_0 has no edges, $Z_0 = q^m$

Proof of Theorem 1- continued

- Call the endpoints of e_j : x_j and y_j
- The q-coloring of G_j are the configurations $\xi \in \{1,\ldots,q\}^V$ that are q-colorings of G_{j-1} and and $\xi(x_j) \neq \xi(y_j)$
- Therefore

$$\frac{Z_j}{Z_{j-1}} = \rho_{G_{j-1,q}}(X(x_j) \neq X(y_j))$$

where $\rho_{G_{j-1,q}}$ is the uniform distribution of all legal q-colorings of G_{j-1} and X is a random coloring of G_{j-1} chosen uniformly from $\rho_{G_{j-1,q}}$

Proof of Theorem 1- continued

- We now use the Gibbs sampler to simulate a random q-coloring X for G_{j-1} several times. The proportion of colorings where $X(x_j) \neq X(y_j)$ is likely to be very close to $\frac{Z_j}{Z_{j-1}}$, by the law of large numbers.
- We do this for every j, to get a good estimate of Z_n

Theorem 1
Time bounds

- How many simulations do we need to estimate each $\frac{Z_j}{Z_{j-1}}$ and how many steps of the Gibbs sampler do we need to run in each simulation?

Theorem 1
Time bounds

- Denote our algorithm's random estimation of $\frac{Z_j}{Z_{j-1}}$ by Y_j.

- There are two sources of error in our procedure:

  i) the Gibbs sampler only runs for a finite number, k, of steps, so the distribution $\mu^{(k)}$ that it produces may differ from $\rho_{G_j,q}$.

  ii) only a finite number of simulations is done, so the proportion of Y_j resulting in q-colorings with $\xi(x_j) \neq \xi(y_j)$ may differ from the expected value $\mu^{(k)}(X(x_j) \neq X(y_j)$

Theorem 1
Time bounds – error ii)

For error ii), we want to ensure

$$\left|Y_j - \mu^{(k)}(X(x_j) \neq X(y_j))\right| \leq \varepsilon_2$$

Suppose we make r simulations to generate Y_j.

By the Chernoff bound, we have

$$pr\left(\left|\sum_{i=1}^{r} Y_{j,i} - \mu r\right| \geq c\mu r\right) \leq 2e^{-(c^2 \mu r)}$$

Where μ is shorthand for $\mu^{(k)}(X(x_j) \neq X(y_j)$.

Theorem 1
Time bounds – error ii)

- We set $c = \varepsilon/2n$ and we will show that $\mu \geq 1/2$
- We want to bound the error by $1/3n$ as we simulate Y_j n times and want the overall error to be less than $1/3$.
- We want to get $2e^{-(c^2 \mu r)} \leq 1/3n$ and we solve for r:

$$\frac{2}{e^{(\varepsilon^2 \mu r / m^2)}} \leq \frac{2}{e^{(\varepsilon^2 r / 2m^2)}} \leq \frac{1}{3n}$$

Theorem 1
Time bounds – error ii)

- And we get

$$\frac{\varepsilon^2 r}{2n^2} \geq \ln(6n) \Rightarrow r \geq \frac{2n^2(\ln 6n)}{\varepsilon^2}$$

- This is the number of simulations we must do for each Y_j to get a multiplicative error of $\varepsilon/2n$

Theorem 1
Time bounds – error i)

- Recall that the number of iteration needed for the Gibbs sampler to come within total variation distance ε of $\rho_{G,q}$ is $O(m(\log(m)+\log(\varepsilon^{-1})))$
- We want to get

$$|\mu^{(k)}(X(x_j)\neq X(y_j))-\rho_{G_{j-1},q}(X(x_j)\neq X(y_j))|<\varepsilon_1$$

- And it is enough that $d_{TV}(\mu^{(k)},\rho_{G_{j-1},q})<\varepsilon_1$ and so $k=O(m(\log(m)+\log(\varepsilon^{-1})))$

In conclusion

- If we take $\varepsilon_1 = \varepsilon_2 = 1/10$, we get, with probability $\geq 2/3$ a multiplicative approximation of the number of q-colorings of G.

- We have shown that if we can sample efficiently, we can approximate #q-coloring efficiently.


INTERMISSION

The relationship between counting and sampling

- If we can approximate a counting problem *f* efficiently, we can sample efficiently and vice versa.

- We have shown one direction for #q-coloring. The proof is similar for other counting problems. We will not show a generic proof here.

Approximate counting implies sampling

- Assume we can approximate a counting problem *f* efficiently with multiplicative error, i.e. we have a efficient function *g,* and

$$\forall x \in \{0,1\}^k : \left| g(x) - f(x) \right| < \varepsilon_k \cdot f(x)$$

- We calculate $g_{x1=0} = g(x \mid x_1 = 0)$ and $g_{x1=1} = g(x \mid x_1 = 1)$ and sample x_1 to be 0 with probability $g_{x1=0} / (g_{x1=0} + g_{x1=1})$ and 1 with probability $g_{x1=1} / (g_{x1=0} + g_{x1=1})$

Approximate counting implies sampling - continued

- For every i, we then calculate $g_{x_i=0} = g(x \mid x_1 \ldots x_{i-1}, x_i = 0)$,and similarly $g_{x_i=1}$, and sample x as before. We continue this until we have sampled all *k* values for *x*.
- Every calculation of *g* has an error of ε , and as we need k values of x, we can bound the error by k ε .

General Approximate counting in BPP^NP

- **BPP** is the class of decision problems solvable by a probabilistic Turing machine in polynomial time, with an error probability of at most 1/3 for all instances.
- **PC** is defined as follows: A binary relation $R \in PC$ if the following 2 conditions hold:

1. For some polynomial p, if $(x, y) \in R$ then $|y| \leq p(|x|)$
2. There exists a polynomial-time algorithm that given (x, y) determines whether or not $(x, y) \in R$

Theorem 2 – approximating #R

- Theorem: For every $R \in PC$ there exists a probabilistic-time oracle machine that when given access to NP approximates #R to within some constant factor.
- This shows that #R is in BPP^NP.

Why we cannot efficiently approximate #P

- We cannot efficiently approximate every #P problem. Efficiently approximating #R yields an efficient algorithm for deciding membership in

$$S_R = \{x : R(x) \neq \emptyset\}$$

- At best, we can hope that approximating #R is not harder than deciding S_R. This is indeed the case if S_R is NP-complete.

Proof of Theorem 2 – general idea

- We assume that $R(x) \subseteq \{0,1\}^l$ where $l = poly(|x|)$. We focus on finding some $i \in \{1,...,l\}$ such that $2^{i-4} \leq |R(x)| \leq 2^{i+4}$. We can also assume that $R(x) \neq \emptyset$ (why?)
- We will proceed iteratively. For $i = 1,...,l+1$, we find out whether or not $|R(x)| < 2^i$. If the answer is positive, we halt with output 2^i , otherwise we continue to the next iteration.

Proof of Theorem 2 – random sieve

- To check whether $|R(x)| < 2^i$ we will use a "random sieve" on the set $R(x)$ such that each element passes the sieve with probability 2^{-i} so we expect $|R(x)|/2^i$ elements to pass through the sieve. If this is the case, then $|R(x)| > 2^i$ if and only if some element of $R(x)$ passes through the sieve. If the sieve can be implemented efficiently then whether or not an element passes through the sieve can be referred to the NP-oracle.

Proof of Theorem 2 – random sieve by hashing

- We will implement a random sieve using a family of hashing functions. In the i-th iteration, we we use a family H_l^i such that each $h \in H_l^i$ has a poly(l)-bit long description and maps l-bit long strings to i-bit long strings. The family is accompanied by an efficient evaluation algorithm which maps pairs (h,x) to h(x) and satisfies, for every $S \subseteq \{0,1\}^l$

$$\Pr_{h \in H_l^i}\left[\left|\{y \in S : h(y) = 0^i\}\right| \notin (1-\varepsilon, 1+\varepsilon)\cdot 2^{-i}\left|S\right|\right] < \frac{2^i}{\varepsilon^2 \left|S\right|}$$

- The random sieve will let y pass iff $h(y) = 0^i$.

Proof of Theorem 2 – implementing the queries

- We need to determine, for some x, i and $h \in H_l^i$ whether $\{y \in R(x) : h(y) = 0^i\} = \emptyset$. This can be cast as a membership in the set

$$S_{R,H} \overset{def}{=} \{(x, i, h) : \exists y.s.t.(x, y) \in R \wedge h(y) = 0^i\}$$

- Assuming $R \in PC$ and the family of hashing functions has an efficient evaluation algorithm, $S_{R,H}$ is in NP.

Proof of Theorem 2 – the procedure

- On input $x \in S_R$ and oracle access to $S_{R,H}$, we proceed (where l is the length of the potential solutions for x):

  if $R(x) = \varnothing$ output 0

  for (i=1 to l){

  uniformly select $h \in H_l^i$

  query the oracle whether $(x, i, h) \in S_{R,H}$

  if the answer is negative, output 2^i

  }

  output 2^l

Proof of Theorem 2 – analysis

- Let $i_x = \lfloor \log_2 |R(x)| \rfloor \geq 0$. The probability that the procedure halts in a specific iteration $i < i_x$ equals

$$\Pr_{h \in H_l^i}[|\{y \in R(x) : h(y) = 0^i\}| = 0] \leq 2^i / R(x)$$

- The probability that the procedure stops before iteration $i_x - 3$ is upper bounded by $\sum_{i=0}^{i_x - 4} 2^i / |R(x)|$, which is less than $1/8$ because $|R(x)| \geq 2^{i_x}$.

Proof of Theorem 2 – analysis continued

- The probability that the procedure does not halt in $i > i_x$ iteration equals

$$\Pr_{h \in H_l^i}[\left|\{y \in R(x) : h(y) = 0^i\}\right| \geq 1]$$

  which is bounded by $\alpha/(\alpha-1)^2$ where $\alpha = 2^i/\left|R(x)\right| > 1$.

- The probability that the procedure does not halt by iteration $i_x + 4$ is upper bounded by $8/49 < 1/6$, because $i_x > (\log_2 \left|R(x)\right| - 1) \Rightarrow \alpha \geq 8$

Proof of Theorem 2 – analysis continued

- Thus, with probability at least $7/8$, the output is at least $2^{i_x-3} > |R(x)|/16$
- With probability at least $5/6$, the output is at most $2^{i_x+4} \leq 16 \cdot |R(x)|$
- And with probability $7/8 - 1/6 > 2/3$, the procedure outputs a value v such that $v/16 \leq |R(x)| \leq 16 \cdot v$

The End

Appendix A – proof that $\mu \geq 1/2$

- We need to show that $\mu^{(k)}(X(x) \neq X(y)) \geq 1/2$
- First, we notice that if x and y are neighbors in G, the left hand equals 1.
- We further notice that if x and y are not neighbors:

$$\mu^{(k)}(X(x) \neq X(y) = 1 - \mu^{(k)}(X(x) = X(y) \geq 1 - \frac{1}{q-d}$$

because x's color is uniform over all the colors except those of his neighbors, and: $1 - \frac{1}{q-d} \geq 1 - \frac{1}{2d-d} \geq \frac{1}{2}$

because $q \geq 2d$ and $d \geq 2$.