7 Approximate Counting

7.1 Counting problems

A *counting problem* Π is defined like a combinatorial optimization problem. However, instead of trying to find an optimal value according to some objective function, the task is to determine the number of feasible solutions, $\#(I) = |S(I)|$. In the following, the sign # always means "number" and will specify that we have a counting problem.

Definition 7.1

(a) In the *DNF counting problem* (#DNF) we are given a Boolean expression Ψ in disjunctive normal form (DNF) and the problem is to determine the number of satisfying assignments for Ψ. (Recall that a Boolean expression in DNF has the form $\Psi = D_1 \vee D_2 \vee \ldots \vee D_m$ where D_j is an implicant, i.e. a conjunction of one or more literals.)

(b) Let $k \in \mathbb{N}$. In the *k-coloring counting problem* $\#\mathrm{COL}_k$ we are given a graph G and the problem is to determine the number of node colorings of G using at most k colors.

It is not difficult to see that counting problems are at least as hard as their decision variants. For example, let the decision variant of the DNF counting problem be the problem to decide whether a given expression Ψ in DNF has a non-satisfying assignment. In this case, an exact knowledge of the number of satisfying assignments for Ψ would immediately give us the answer whether Ψ is not satisfiable.

The complexity class $\#\mathbf{P}$ consists of all counting problems Π for which there is a polynomial time non-deterministic algorithm A (i.e. an algorithm with rules allowing alternative computations) so that for any instance I the number of accepting computations when running A on I is equal to $\#(I)$. This implies that A has an accepting computation if and only if $\#(I) \geq 1$. In other words, A can be used to *verify* in polynomial time whether the question "$\#(I) \geq 1$?" is true. Thus, recalling the definition of the complexity class **NP** in Chapter 1, all decision variants of counting problems in $\#\mathbf{P}$ are in **NP**. Like **NP**, also $\#\mathbf{P}$ has complete problems. The $\#\mathbf{P}$-complete problems represent the "most difficult" problems in $\#\mathbf{P}$. In fact, similar to the **NP**-complete problems, if a $\#\mathbf{P}$-complete problem can be solved exactly in polynomial time, then all counting problems in $\#\mathbf{P}$ can be solved in polynomial time. Since the problem of counting all satisfying assignments of a Boolean expression Φ in CNF is in $\#\mathbf{P}$ and its decision variant is the SAT problem which, as we know, is **NP**-complete, a $\#\mathbf{P}$-complete problem cannot be solved in polynomial time unless $\mathbf{P} = \mathbf{NP}$. Because #DNF is known to be $\#\mathbf{P}$-complete, we can assume that #DNF is hard to solve exactly in polynomial time.

A prominent, classical counting problem that can be solved in polynomial time (and therefore is most probably not $\#\mathbf{P}$-complete) is the problem of counting the number of spanning trees in a graph. This number is equal to the determinant of the so-called Kirchhoff matrix (also known as Laplace matrix) of the graph [Bol98, Section II.3], which can be computed in polynomial time. Also, the number of Euler tours in a graph can be computed exactly in polynomial time.

Since $\#\mathbf{P}$-complete problems seem to be difficult to solve, it seems that there is no other choice than to find good approximations. Interestingly, in this area no good *deterministic* approximation algorithms are known but only good *randomized* approximation algorithms. An excellent overview of algorithms for counting problems can be found in [Wel93].

Analogous to approximation algorithms for optimization problems, our aim will be to find approximation algorithms for counting problems that try to get as close to the optimal solution (in this case, the exact number of feasible solutions) as possible. For this we need the following notation.

Definition 7.2

(a) A *polynomial (time) approximation scheme* (PASC) for a counting problem Π is a deterministic algorithm A that, for every instance I of Π and every constant $0 < \epsilon < 1$, allows to determine a number $A(I)$ in time $O(poly(|I|))$ with

$$(1-\epsilon) \cdot \#(I) \leq A(I) \leq (1+\epsilon) \cdot \#(I) \ . \tag{1}$$

If the runtime is $O(poly(|I|, 1/\epsilon))$ for any $\epsilon > 0$, A is called a *fully polynomial approximation scheme* (FPASC).

(b) A is a *polynomial randomized approximation scheme* (PRASC) if

$$\Pr[(1-\epsilon) \cdot \#(I) \leq A(I) \leq (1+\epsilon) \cdot \#(I)] \geq \frac{3}{4} \ .$$

If the runtime is $O(poly(|I|, 1/\epsilon))$, A is called a *fully polynomial randomized approximation scheme* (FPRASC).

(c) A is an (ϵ, δ)-FPRASC if it can compute in time $O(poly(|I|, 1/\epsilon, \log(1/\delta)))$ a number $A(I)$ with

$$\Pr[(1-\epsilon) \cdot \#(I) \leq A(I) \leq (1+\epsilon) \cdot \#(I)] \geq 1-\delta \ .$$

In all abbreviations above, the "C" stands for "counting".

Note that we demand for an FPASC a runtime that is polynomial in $1/\epsilon$ and not in $\log(1/\epsilon)$. Using $\log(1/\epsilon)$ would have the consequence that the existance of an FPASC for a $\#\mathbf{P}$-complete problem would imply that $\mathbf{P} = \#\mathbf{P}$.

Condition (1) can also be expressed as $|(A(I)/\#(I)) - 1| \leq \epsilon$. Instead of a probability value of $3/4$ one could also use another value as long as this value is at least some constant larger than $1/2$, because that would suffice to transform every FPRASC via probability amplification into an (ϵ, δ)-FPRASC.

Theorem 7.3 *Every FPRASC A can be transformed into an (ϵ, δ)-FPRASC.*

The proof is an assignment.

7.2 The classical Monte Carlo method

The goal of this section is to present a fully polynomial randomized approximation scheme for the DNF counting problem that is based on the classical Monte Carlo method. First, we will see that a direct approach can be too weak, and then we will show how to modify the description of the problem so that we obtain a good approximation for all instances of #DNF.

7.2.1 The Monte Carlo algorithm and blind sampling

Let Π be a counting problem, and let for every instance I a (very large) set U_I be given whose cardinality is known and for which $S(I) \subseteq U_I$. U_I is called the *universe* or *sample space* for I. $S(I)$ is the set whose cardinality we have to determine. Let $\chi : U_I \rightarrow \{0, 1\}$ be the characteristic function of $S(I)$, i.e.

$$\chi(u) = \begin{cases} 1 & \text{if } u \in S(I) \\ 0 & \text{otherwise} \end{cases}$$

We demand two properties:

- There is a deterministic membership test MEMBER that computes for every $u \in U_I$ the value of $\chi(u)$ in time polynomial in $|I|$.
- There is a randomized algorithm UG that generates an element $u \in U_I$ in time polynomial in $|I|$ so that for any $u \in U_I$,
$$\Pr[\text{UG generates } u] = \frac{1}{|U_I|} .$$

The output of UG is called a *sample* of U_I. Such a generator of samples is also called a *uniform generator*. Consider now the following *Monte Carlo* algorithm MC:

Algorithm MC(T):

for $i = 1$ to T do
  (1) use UG to draw a sample $u \in U_I$
  (2) use MEMBER to compute $Y_i = \chi(u)$
$R = T^{-1} \sum_{i=1}^{T} Y_i$
output $Z = R \cdot |U_I|$

In the following, we will always assume that

$$r = \frac{\#(I)}{|U_I|}$$

represents the ratio of the size $S(I)$ to the size of the universe U_I. Since we use a uniform generator, it holds that $\mathrm{E}[Y_i] = r$. Hence, $\mathrm{E}[R] = r$ and therefore

$$\mathrm{E}[\mathrm{MC}(T)] = \mathrm{E}[Z] = r \cdot |U_I| = \#(I) ,$$

independent of T. Thus, the expected value for Z computed by MC is the number we are searching for. Intuitively, it should be clear that the more samples MC draws (i.e. the larger T), the closer it will get to the exact value of $\#(I)$. However, how large does T have to be chosen to ensure that a certain deviation from the expected value will only occur with small probability? This is answered by the following theorem, which is also called the *estimator theorem* of the Monte Carlo method.

Theorem 7.4 *For any ϵ, δ with $0 < \epsilon, \delta < 1$ it holds with $T_r(\epsilon, \delta) = \lceil \frac{3}{r \cdot \epsilon^2} \ln(2/\delta) \rceil$ that*

$$\Pr[(1-\epsilon) \cdot \#(I) \leq MC(T_r(\epsilon, \delta)) \leq (1+\epsilon) \cdot \#(I)] \geq 1 - \delta .$$

Proof. Using the Chernoff bounds in Chapter 5 results in

$$\begin{aligned}
&\Pr[(1-\epsilon) \cdot \#(I) \leq \mathrm{MC}(T_r(\epsilon, \delta)) \leq (1-\epsilon) \cdot \#(I)] \\
&= \Pr[(1-\epsilon) \cdot \#(I) \leq R \cdot |U_I| \leq (1-\epsilon) \cdot \#(I)] \\
&= \Pr[(1-\epsilon) \cdot r \leq R \leq (1+\epsilon) \cdot r] \\
&= \Pr[(1-\epsilon) \cdot T_r(\epsilon, \delta) \cdot r \leq \underbrace{T_r(\epsilon, \delta) \cdot R}_{\text{sum of 0-1 r.v.}} \leq (1+\epsilon) \cdot T_r(\epsilon, \delta) \cdot r] \\
&\geq 1 - 2\mathrm{e}^{-T_r(\epsilon,\delta) \cdot r \cdot \epsilon^2/3} \overset{(*)}{\geq} 1 - 2\mathrm{e}^{-\ln(2/\delta)} = 1 - \delta .
\end{aligned}$$

In $(*)$ we used the value for $T_r(\epsilon, \delta)$ defined above. □

Does therefore $\mathrm{MC}(T_r(\epsilon, \delta))$ represent an (ϵ, δ)-FPRASC for every counting problem? At first glance this may seem to be the case, but the result has two weaknesses that result in a negative answer.

1. $T_r(\epsilon, \delta)$ is linear in $1/r$ and therefore depends on the value $\#(I)$, so that we cannot determine $T_r(\epsilon, \delta)$ in advance. In order to specify $T_r(\epsilon, \delta)$, we would first need a good estimate for $1/r$.

2. Even more serious is the following observation: $1/r$ can be exponentially large! For example, if we apply MC to #DNF, then we can define a class of Boolean expressions Ψ in DNF with n Boolean variables and m implicants and the following characteristics: $m = \Theta(poly(n))$, $|\Psi| = \Theta(n \cdot m)$, $U_\Psi = \{\text{FALSE}, \text{TRUE}\}^n$, $|U_\Psi| = 2^n$, and $\#(\Psi) = 2^{n/2}$. We chose here as the universe all possible truth assignments for the n Boolean variables. For this class of truth assignments it is very easy to construct a uniform generator. Hence, $r = 1/2^{n/2}$ and therefore $T_r(\epsilon, \delta) = O(2^{n/2} \cdot (\frac{1}{\epsilon})^2 \log(\frac{1}{\delta}))$. Hence, the runtime is exponential in $|\Psi|$.

One could hope now that the Chernoff bounds are not precise enough, but unfortunately they are quite accurate in our case. Hence, a more accurate analysis would only have a slight impact on T in the algorithm MC(T).

In the example above, the universe only depends on the number n of variables but not on the inner structure of Ψ. Therefore, one also calls this approach *blind sampling*.

7.3 Importance sampling

There is an approach in which the knowledge about the problem – in our case, this is the #DNF problem – can be used to construct a universe U_I so that the ratio $r = \#(I)/|U_I|$ is significantly larger than in the blind sampling case. This approach is also called *importance sampling*.

In the following, let $\Psi = D_1 \vee \ldots \vee D_m$ a Boolean expression in DNF over n Boolean variables. The number of literals in D_j is given by k_j. We will use the following facts:

1. That we can easily construct a satisfying assignment for Ψ and

2. that we can exactly determine the number of satisfying assignments for a single implicant.

As an example, consider the first implicant of the expression $\Psi = (x_1 \wedge \bar{x}_2 \wedge x_3) \vee (\bar{x}_1 \wedge \bar{x}_4 \wedge x_5)$. It is easy to see that one can assign truth values to the variables x_1, x_2 and x_3 so that this implicant is true. For this there is exactly one assignment: $x_1 = x_3 = \text{TRUE}$ and $x_2 = \text{FALSE}$. The other variables x_4 and x_5 can be chosen in an arbitrary way. Hence, altogether there are $2^2 = 4$ satisfying assignments for the first implicant. In general, it holds:

Lemma 7.5 *Let $D = \ell_1 \wedge \ldots \wedge \ell_k$ be an implicant with k literals in a Boolean expression over n Boolean variables. Then there are exactly 2^{n-k} satisfying assignments for D.*

Consider now a truth assignment u that satisfies Ψ. In this case, it must satisfy at least one of the implicants of Ψ. Hence,

$$\begin{aligned} S(\Psi) &= \bigcup_{j=1}^{m} \{T \mid u \text{ satisfies } C_j\} \\ &= \bigcup_{j=1}^{m} \{u \mid u \text{ satisfies } C_j \text{ but no } C_i \text{ with } i < j\}\ . \end{aligned}$$

The last inequality holds, since we only removed some redundant assignments u from the sets that have already been considered earlier. Now, let us introduce a new set $S'(\Psi)$ of feasible solutions with the property that

$$S'(\Psi) = \bigcup_{j=1}^{m} \{(u, j) \mid u \text{ satisfies } C_j \text{ but no } C_i \text{ with } i < j\}\ .$$

Obviously, $\#(\Psi) = |S(\Psi)| = |S'(\Psi)|$. For $S'(\Psi)$ we can provide a universe that can be much smaller than in the *blind sampling* approach and that depends now on the inner structure of Ψ:

$$U_\Psi \quad = \quad \{(u,j) \mid u \text{ satisfies } C_j \text{ and } j \in \{1,\dots,m\}\}$$

It certainly holds that $S'(\Psi) \subseteq U_\Psi$.

What did we do here? We excluded from the universe all assignments that do not fulfill Ψ. Truth assignments appearing in pairs $(u,j) \in U_\Psi$ are only satisfying truth assignments! It may be the case that the same truth assignment appears in different pairs (u,j) and (u,j'), but no truth assignment appears more than m times. In contrast to $S'(\Psi)$, U_Ψ ignores the fact that the j in the pair (u,j) may specify the first implicant that is satisfied by u. In $S'(\Psi)$, every truth assignment can only appear once. Hence, an element $(u,j) \in S'(\Psi)$ is also called the *canonical element* of $u \in S(\Psi)$.

Due to Lemma 7.5 we can give an exact value for $|U_\Psi|$:

$$|U_\Psi| = \sum_{j=1}^{m} 2^{n-k_j} \ .$$

The ratio r that can be exponential in the blind sampling case is now guaranteed to be polynomial in $|\Psi|$:

Lemma 7.6

$$r = \frac{|S'(\Psi)|}{|U_\Psi|} \geq \frac{1}{m} \ .$$

Proof. As already mentioned above,

$$|U_\Psi| = \sum_{u \in S(\Psi)} |\{j \mid u \text{ satisfies } D_j\}| \leq m \cdot \#(\Psi) = m \cdot |S'(\Psi)| \ .$$

□

The characteristic function χ for $S'(\Psi)$ that will be used in MC is

$$\chi(u,j) = \begin{cases} 1 & \text{if } j = \min\{k \mid u \text{ satisfies } D_k\} \\ 0 & \text{otherwise} \end{cases}$$

and can obviously be computed in time $O(m \cdot n)$ by a program Member. Hence, the first condition for the successful application of the Monte Carlo algorithm is fulfilled.

It remains to fulfill the second condition, i.e. to make sure that elements can be chosen from U_Ψ with uniform probability. In other words, we need a uniform generator for U_Ψ. In contrast to the blind sampling method, this depends now on the form of Ψ. Short implicants in Ψ have many satisfying assignments, and therefore these implicants somehow have to be preferred. This can be achieved by using a random experiment in which an implicant D_j is selected with a probability that depends on its length k_j.

Algorithm UG:

1. choose a $j \in \{1,\dots,m\}$ with probability $2^{n-k_j}/|U_{\Psi|}|$
2. set the variables in D_j so that D_j is satisfied
3. choose a truth assignment for the remaining variables uniformly at random
4. output j and the resulting truth assignment u

Lemma 7.7 *Algorithm* UG *is a uniform generator for* U_Ψ.

Proof. It holds that

$$\begin{aligned}\Pr[(u,j) \in U_\Psi \text{ is selected}] &= \Pr[j \in \{1,\ldots,m\} \text{ and } u \in S(D_j) \text{ are selected}] \\ &\overset{(*)}{=} \frac{2^{n-k_j}}{|U_\Psi|} \cdot \frac{1}{2^{n-k_j}} \\ &= \frac{1}{|U|_\Psi} .\end{aligned}$$

In $(*)$ we used the probability value in (1) of UG and Lemma 7.5. □

Theorem 7.8 *Using* $S'(\Psi)$ *for* MEMBERSHIP *and* U_Ψ *for* UG *it follows that* $T_r(\epsilon,\delta) = \lceil m \cdot \frac{3}{\epsilon^2} \ln \frac{2}{\delta} \rceil$. *Thus, the algorithm MC(*$T_r(\epsilon,\delta)$*) is an* (ϵ,δ)*-FPRASC for #DNF.*

Proof. According to Lemma 7.6 it holds that $1/r \leq m$ and according to Lemma 7.7 we have a uniform generator for U_Ψ. Using this in Theorem 7.4 gives the claimed result. □

The algorithm presented in this section is due to Karp, Luby and Madras [KLM89].

7.4 The Markov chain Monte Carlo method

We present now an approach that is based on the Monte Carlo method presented earlier in this chapter. This approach has recently been used to solve a large number of counting problems. It is called the *Markov chain Monte Carlo method.*

We start with some notation from Markov chain theory. A very good introduction to the theory of Markov chains can be found in the book by Feller [Fel70].

Random walks

A *Markov chain* $\mathcal{M}$ is a discrete-time stochastic process defined over a set of states S in terms of a matrix P of *transition probabilities.* The set S is either finite or countably infinite. The transition probability matrix $P = (p_{i,j})$ has one row and one column for each state in S. The Markov chain is in one state at any time, making state transitions at discrete time steps $t = 1, 2, \ldots$. The entry $p_{i,j}$ represents the probability that the next state will be j given that the current state is i. Thus, for all $i, j \in S$ we have $0 \leq p_{i,j} \leq 1$ and $\sum_j p_{i,j} = 1$. Such a matrix P is called *stochastic.*

An important property of a Markov chain is the *memorylessness property*: the future behavior of a Markov chain depends only on its current state, and not on how it arrived at this state. We will denote by X_t the state of the Markov chain at time t. Thus, the sequence $\{X_t\}$ specifies the history or the evolution of the Markov chain. The memorylessness property implies that for all $t \geq 1$ and all $i_0, \ldots, i_t \in S$,

$$\Pr[X_t = i_t \mid X_0 = i_0, X_1 = i_1, \ldots, X_{t-1} = i_{t-1}] = \Pr[X_t = i_t \mid X_{t-1} = i_{t-1}] = p_{i_{t-1},i_t} .$$

A Markov chain is called *symmetric* if in addition,

$$\Pr[X_t = i_t \mid X_{t-1} = i_{t-1}] = \Pr[X_t = i_{t-1} \mid X_{t-1} = i_t]$$

for all $i_{t-1}, i_t \in S$. Thus, if a Markov chain is symmetric, then for all $j \in S$,

$$\sum_{i \in S} p_{i,j} = \sum_{i \in S} p_{j,i} = 1 .$$

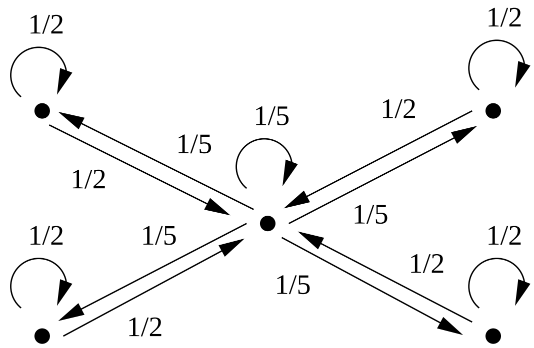

Figure 1: A uniform random walk.

That is, P is *doubly stochastic.*

Another way of representing a Markov chain is to consider a directed graph $G_P = (V, E)$ with $V = S$, $E = \{(i, j) \mid p_{i,j} > 0\}$ and edges (i, j) with weight $p_{i,j}$. Then a Markov chain $X_0, X_1, \ldots$ can be seen as a *random walk* on G starting at node X_0, then moving to node X_1, and so on.

In the following, we will simply identify a Markov chain or random walk by its transition probability matrix P. For an example of a random walk P on a graph G see Figure 1 (edges of weight 0 are left out). In this figure, all outgoing edges of a node have the same weight. Such a random walk is called *uniform.*

We will use random walks as a method to achieve a (near) uniform generation of a sample out of a sample space S. This would allow us to use the generation process in our Monte Carlo algorithms. In order to obtain a sample, the random walk would usually start at some arbitrary node X_0 and then run for a specified amount of time steps t. X_t will then be considered the random sample that is given to the Monte Carlo algorithm. To be able to determine the quality of the sample X_t, we have to determine the probabilities $\Pr[X_t = i]$ for all $i \in V$. Or in other words, we have to determine the *probability distribution* $\vec{q}^{(t)} = (q_1^{(t)}, q_2^{(t)}, \ldots, q_{|V|}^{(t)})$ so that $\Pr[X_t = i] = q_i^{(t)}$ for all $i \in V$. This will be our task for the rest of this section.

First of all, for all $t \geq 1$ it holds that

$$\vec{q}^{(t)} = \vec{q}^{(t-1)} \cdot P = \ldots = \vec{q}^{(0)} \cdot P^t$$

where $\vec{q}^{(0)}$ is the *initial probability distribution* of the random walk.

Let $P^t = (p_{i,j}^{(t)})$. A random walk P is called *irreducible* if for all i, j there is a finite t so that $p_{i,j}^{(t)} > 0$ (i.e. j is reachable from i in a finite amount of time). Obviously, if P is irreducible, then the corresponding graph G_P must be strongly connected. A random walk P is called *periodic* if there is an initial distribution $\vec{q}^{(0)}$ and a $t > 1$ such that $\vec{q}^{(0)} \cdot P^t = \vec{q}^{(0)}$ and $\vec{q}^{(0)} \cdot P^{t-1} \neq \vec{q}^{(0)}$. If a random walk is irreducible and aperiodic, it is called *ergodic.* Ergodic random walks have the following important property.

Theorem 7.9 *For every ergodic random walk P there is a unique vector $\vec{\pi}$ so that for all initial probability distributions $\vec{q}^{(0)}$, $\lim_{t\to\infty} \vec{q}^{(0)} \cdot P^t = \vec{\pi}$. $\vec{\pi}$ is called the* stationary distribution *of P.*

The word "stationary distribution" comes from the fact that $\vec{\pi}$ represents a fixpoint w.r.t. multiplication with P, i.e. $\vec{\pi} = \vec{\pi} \cdot P$.

The random walk in Figure 1 is ergodic. Figure 2 shows a symmetric random walk that is not ergodic. However, simply adding self-loops to the nodes with positive probabilities would convert it into an ergodic random walk. For ergodic random walks the following theorem can easily be shown.

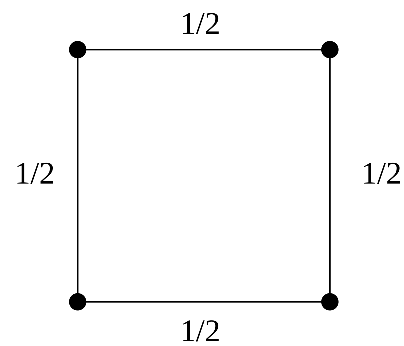

Figure 2: A symmetric, non-ergodic random walk.

Theorem 7.10 *Let P be an ergodic random walk on a graph $G = (V, E)$. Then*

(a) If P is uniform, then $\vec{\pi} = (\frac{\deg(v_1)}{2|E|}, \ldots, \frac{\deg(v_{|V|})}{2|E|})$.

(b) If P is symmetric, then $\vec{(\pi)} = (\frac{1}{|V|}, \ldots, \frac{1}{|V|})$

Proof. Since P is ergodic, we know from Theorem 7.9 that it must have a unique stationary distribution $\vec{\pi}$. Hence, it just remains to find a distribution that fulfills this property.

(a): Suppose that P is uniform. Then it holds for the $\vec{\pi}$ given in (a) that for all $w \in V$,

$$\sum_{(v,w)\in E} \pi_v \cdot p_{v,w} = \sum_{(v,w)\in E} \frac{1}{\deg(v)} \cdot \frac{\deg(v)}{2|E|} = \frac{\deg(w)}{2|E|} .$$

Hence, $\vec{\pi} \cdot P = \vec{\pi}$. Since $\sum_v \pi_v = 1$, π is also a probability distribution and therefore the stationary distribution we are searching for.

(b): Suppose that P is symmetric. Then it holds for the $\vec{\pi}$ given in (b) that for all $w \in V$,

$$\sum_{(v,w)\in E} \pi_v \cdot p_{v,w} = \frac{1}{|V|} \sum_{(v,w)\in E} p_{v,w} = \frac{1}{|V|} .$$

Obviously, $\vec{\pi}$ is also a probability distribution and therefore the stationary distribution we are looking for. □

From now on we will always assume that all random walks are symmetric and ergodic.

Theorem 7.10(b) says that in a sufficiently large amount of time, every node will be visited with equal probability. This would allow us in principle to construct a uniform generator for V. However, for this method to be practical, we have to determine, how long it takes until one is "sufficiently close" to a stationary distribution. That is, the problem is to determine a t so that $||\vec{q}^{(0)} \cdot P^t - \vec{\pi}||$ is sufficiently close to 0, no matter what $\vec{q}^{(0)}$ is selected. Here, $|| \cdot ||$ may represent an arbitrary vector norm. The most commonly used norm is the *variation distance* that is defined as

$$\delta_u(t) = \frac{1}{2} \sum_{w\in V} |\,\Pr[P \text{ started with } u \text{ is in } t \text{ steps at } w] - \pi_w| \,.$$

We can prove the following lemma.

Lemma 7.11 *Let P be a symmetric and ergodic random walk on a graph $G = (V, E)$ and let $W \subseteq V$. Then*

$$\frac{|W|}{|V|} - \delta_{u,t} \leq \Pr[P \textit{ started in } u \textit{ is after } t \textit{ steps in a node in } W] \leq \frac{|W|}{|V|} + \delta_{u,t} \ .$$

Proof. From Theorem 7.10(b) we know that the stationary distribution of P is $\pi = (\frac{1}{|V|}, \ldots, \frac{1}{|V|})$. Thus, it holds that

$$\begin{aligned}
&\Pr[P \text{ started with } u \text{ is in } t \text{ steps at a node in } W] \\
&= \sum_{w \in W} \Pr[P \text{ started with } u \text{ is in } t \text{ steps at } w] \\
&= \sum_{w \in W} \left(\frac{1}{|V|} + \Pr[P \text{ started with } u \text{ is in } t \text{ steps at } w] - \frac{1}{|V|} \right) \\
&= \frac{|W|}{|V|} + \sum_{w \in W} \left(\Pr[P \text{ started with } u \text{ is in } t \text{ steps at } w] - \pi_w \right) \ .
\end{aligned}$$

The last sum is between $-\delta_u(t)$ and $+\delta_u(t)$, as can easily be checked. Hence, the lemma is true. □

Note that in the statement of Lemma 7.11 the deviation from the value $|W|/|V|$ does not depend on $|W|$.

As an example for the speed of convergence, consider the mixing of a deck of cards. The nodes of the graph represent all possible arrangements of the cards. Two arrangements are connected by an edge if one can get in one mixing operation from one to the other. Consider each transition to be of equal probability. Theorem 7.10(b) guarantees in this case that if one would mix the cards for an infinite amount of time, every arrangement would be of equal probability. The question is, how many mixing operations have to be performed to be close to the stationary distribution. Using the example of mixing cards, the following notation was introduced.

Definition 7.12 *Let P be an ergodic random walk on a graph $G = (V, E)$. Let $t(|V|, \epsilon_0)$ be chosen so that for all $u \in V$ and all $t \geq t(|V|, \epsilon_0)$, $\delta_u(t) \leq \epsilon_0$. The time $t(|V|, \epsilon_0)$ is called the* mixing time *of P. If $t(|V|, \epsilon_0) = poly(\log |V|, 1/\epsilon_0)$, then P is called* rapidly mixing.

Note that in the definition of "rapidly mixing" the time has to be polylogarithmic in $|V|$. This is due to the fact that a random walk is usually performed in a state space that is exponentially large in the description of the problem. In this case, "polylogarithmic in $|V|$" would mean "polynomial in the size of the input". For example, in a deck of n cards there are $|V| = n!$ possible ways of arranging the cards, and therefore $\log |V| = \Theta(n \log n)$. This example also shows that one usually cannot completely write down P. Instead, the random walk is usually defined by an algorithm that specifies with which probability to move from one state to another. In the example with the deck of cards, this would be a guideline of how to perform a mixing operation.

Since in our stationary distribution all arrangements of cards should have the same probability, it is important for a rapidly mixing random walk to be able to quickly reach any arrangement from any other arrangement, i.e. the diameter of the corresponding graph G should be small.

Because of Theorem 7.10(b), it is usually easy to design random walks whose stationary distribution is a uniform distribution over all nodes. The hard work usually consists in proving that the random walk is rapidly mixing.

7.5 Counting proper node colorings

We consider now the counting problem $\#\mathrm{COL}_k$ given in Definition 7.1(b). Our proof will follow the arguments in [Jer95, Jer98]. This counting problem has interesting applications in statistical physics.

In the following, an important requirement will be that the degree $\Delta(G)$ of the graph $G = (V, E)$ is small compared to k. In particular, we will demand that $k \geq 2\Delta(G) + 1$. The reason for this is that we cannot choose k to be too small if we want to select one or more k-colorings, since the decision variant of the node coloring problem is **NP**-complete and it would therefore be difficult to determine such a k-coloring. On the other hand, the algorithm GreedyCol in Chapter 2 can always find a node coloring with at most $\Delta(G) + 1$ colors. The reason why we need the condition $k \geq 2\Delta(G) + 1$ has to do with the analysis of the random walk described below to obtain a random node coloring.

In the following, we will set $n = |V|$ and $m = |E|$. Intuitively, the more edges the graph has, the harder it is to construct a node coloring. Therefore we consider the following construction: Let $G_m = G$ and, for all $i \in \{0, \ldots, m-1\}$, let G_i be the graph that is obtained by removing an arbitrary edge from G_{i+1}. We will consider now not only the counting problem for G, but also the counting problems for $G_m, \ldots, G_0$. The starting point of our approach will be the following facts:

(a) $\#(G_0) = k^n$, since in a graph with n nodes and no edge there are k^n proper k-colorings.

(b) $\#(G_m) = \#(G)$, which is the value we intend to determine.

(c) $S(G_m) \subseteq S(G_{m-1}) \subseteq \ldots \subseteq S(G_1) \subseteq S(G_0)$.

From (a) and (b) it follows that

$$\#(G) = \#(G_m) = \frac{\#(G_m)}{\#(G_{m-1})} \cdot \frac{\#(G_{m-1})}{\#(G_{m-2})} \cdot \ldots \cdot \frac{\#(G_1)}{\#(G_0)} \cdot \underbrace{\#(G_0)}_{=k^n} . \tag{2}$$

Hence, the counting problem for G_i can be solved if it can be solved for G_{i-1}. Item (c) is important for the random walk we will describe later. Because of property (2), one can say that the counting problem is *self reducible*. In the following, we define $r_i = \#(G_i)/\#(G_{i-1})$ for all i. Thus, $\#(G) = k^n \cdot \prod_{i=1}^{m} r_i$.

As it is shown in the next lemma, the increase in the number of proper colorings caused by the deletion of a single edge is quite small; it cannot more than double.

Lemma 7.13 *For all $i \in \{1, \ldots, m\}$,*

$$\frac{1}{2} \leq \frac{\#(G_i)}{\#(G_{i-1})} \leq 1 .$$

Proof. Let $\{u, w\} \in E_i$ be the edge that is in $G_i = (V, E_i)$ but not in G_{i-1}. Since $S(G_i) \subseteq S(G_{i-1})$, it holds that $\#(G_i)/\#(G_{i-1}) \leq 1$.

It remains to show that $\#(G_{i-1}) \leq 2 \cdot \#(G_i)$. Consider the function f that maps every coloring in $S(G_i)$ to a coloring in $S(G_{i-1}) \setminus S(G_i)$ by setting the color of u equal to the color of w. Then f is surjective, since every coloring $C' \in S(G_{i-1}) \setminus S(G_i)$ has at least one coloring $C \in S(G_i)$ with $C' = f(C)$: in C', u and w must have the same color and simply selecting any color for u that does not conflict with a color of a neighbor would result in a proper coloring in $S(G_i)$.

Hence, $|S(G_i)| \geq |S(G_{i-1}) \setminus S(G_i)|$. So altogether,

$$|S(G_{i-1})| = |S(G_i)| + |S(G_{i-1}) \setminus S(G_i)| \leq 2|S(G_i)| ,$$

which concludes the proof. $\square$

Consider now the following algorithm ColCount$_k$ with the two subroutines Markov$_t$ and Ratio:

Algorithm MARKOV_t:

$C = \text{GREEDYCOL}(G)$
for $j = 1$ to t do
  choose a random $u \in V$ and color $c \in \{1, \ldots, k\}$
  $C' = C$ with the color of u replaced by c
  if C' is a proper coloring then $C = C'$
output C

Algorithm $\text{RATIO}(i)$:

for $\tau = 1$ to T do
  $C = \text{MARKOV}_t(G_{i-1})$
  if $C \in S(G_i)$
    then $X_\tau^{(i)} = 1$
    else $X_\tau^{(i)} = 0$
output $R_i = \frac{1}{T} \sum_{\tau=1}^{T} X_\tau^{(i)}$

Algorithm $\text{COLCOUNT}_k(G, \epsilon)$:

$m = |E|$; $n = |V|$
$G_m = G$
for $i = m$ downto 1 do
  $G_{i-1} = G_i$ without some arbitrarily chosen edge
  $R_i = \text{RATIO}(i)$
$Z = \prod_{i=1}^{m} R_i$
output $k^n \cdot Z$

In these algorithms two values are not determined: the t in MARKOV_t and the T in RATIO. We will select expressions for these that allow COLCOUNT to be an FPRASC for $\#\text{COL}_k$.

The subroutine MARKOV describes a random walk on a graph $\mathcal{G}(G)$ whose node set is the set of all proper node colorings for G. The random walk begins with a coloring $C_0(G)$ that is given by the algorithm $\text{GREEDYCOL}(G)$ in Chapter 2. The edges and edge weights are determined by the rule in MARKOV_t to choose a random node and a random color for that node.

$\mathcal{G}(G)$ is connected, since our rule to create a new coloring certainly allows to turn any proper coloring C into a proper coloring C' in a finite number of recoloring steps (note that $k \geq 2\Delta(G) + 1$). Or in other words, the corresponding random walk is irreducible. Furthermore, the probability to move from coloring C to coloring C' is equal to the probability to move from C' to C. Hence, the random walk is also symmetric. Since the new color c of a node v can also be its old color. Hence, the random walk has self-loops and is therefore aperiodic. So altogether the random walk corresponding to MARKOV is symmetric and ergodic, which together with Theorem 7.10(b) results in the following lemma.

Lemma 7.14 *For $t \to \infty$, $\text{MARKOV}_t(G)$ generates a proper node coloring for G with at most k colors uniformly at random.*

Thus, we can use $\text{MARKOV}_t(G)$ as a uniform generator if t is sufficiently large. For an arbitrary $\epsilon_0 \in (0, 1)$ let $t = t(G, \epsilon_0)$ be the mixing time of the random walk, i.e. the time point at which the random walk can be stopped, since for all $t' \geq t$,

$$\frac{1}{2} \sum_{C \in S(G)} \left| \Pr[\text{MAROV}_{t'}(G) \text{ outputs } C] - \frac{1}{\#G} \right| = \delta_{C_0(G)}(t') \leq \epsilon_0 \, .$$

Because of the ϵ_0 we also call MARKOV_t a *near-uniform generator*. Jerrum showed that random walk corresponding to MARKOV_t is rapidly mixing.

Theorem 7.15 ([Jer95, Jer98]) *Let $G = (V, E)$ be a graph with $2\Delta(G) + 1 \leq k$. Then the mixing time of MARKOV_t is*

$$t(G, \epsilon_0) = \left\lceil \frac{k - \Delta(G)}{k - 2\Delta(G)} \cdot |V| \cdot \ln\left(\frac{|V|}{\epsilon_0}\right) \right\rceil \, .$$

How can a near-uniform generator for the set of proper colorings for G help us? Note that in the Monte Carlo method we need a uniform generator that can generate samples out of a universe of *known* size and not just to obtain random samples out of some unknown universe. This is the place where we can use property (2):

We know the size of $\#(G_0)$, and therefore $\textsc{Markov}_t$ can help us to get a very good approximation of $\#(G_1)$. Once we know $\#(G_1)$ sufficiently well, we can again use $\textsc{Markov}_t$ to approximate $\#(G_2)$ sufficiently well, and so on, until we found a good approximation for $\#(G_m) = \#(G)$. Since due to Lemma 7.13 $r_i = \#(G_i)/\#(G_{i-1}) \geq 1/2$ for all i, the ratio of the set whose size we intend to determine and the set whose size we know is always sufficiently large so that (like in the importance sampling) this strategy should work.

In order to prove that we get a good approximation for $\#(G)$, we have to examine the R_i's and Z. Let us choose

$$T = \frac{74m}{\epsilon^2} \quad \text{and} \quad \epsilon_0 = \frac{\epsilon}{6m} .$$

Using this choice for ϵ_0 in the expression for t above, we get

$$t = \left\lceil \frac{k - \Delta(G)}{k - 2\Delta(G)} \cdot n \cdot \ln\left(\frac{6nm}{\epsilon}\right) \right\rceil .$$

Lemma 7.16

- *(a) For all $i \in \{1, \ldots, m\}$,*

$$\left(1 - \frac{\epsilon}{3m}\right) \cdot r_i \leq \mathrm{E}[R_i] \leq \left(1 + \frac{\epsilon}{3m}\right) \cdot r_i .$$

(b)

$$\left(1 - \frac{\epsilon}{2}\right) \#(G) \leq k^n \cdot \mathrm{E}[Z] \leq \left(1 + \frac{\epsilon}{2}\right) .$$

Proof. (a): It holds that $\mathrm{E}[R_i] = \mathrm{E}[X_\tau^{(i)}]$ for any τ. From Lemma 7.11 it follows that $r_i - \epsilon_0 \leq \mathrm{E}[X_\tau^{(i)}] \leq r_i + \epsilon_0$. Since $r_i \geq 1/2$, the statement follows.

(b): With $k^n \cdot \prod_{i=1}^m r_i = \#(G)$ and (a) it follows that

$$\left(1 - \frac{\epsilon}{3m}\right)^m \cdot \#(G) \leq k^n \cdot \mathrm{E}[Z] \leq \left(1 + \frac{\epsilon}{3m}\right)^m \cdot \#(G) .$$

Moreover,

$$\left(1 - \frac{\epsilon}{3m}\right)^m \geq 1 - \frac{\epsilon}{2} \quad \text{and} \quad \left(1 + \frac{\epsilon}{3m}\right)^m \leq 1 + \frac{\epsilon}{2} ,$$

which proves (b). $\square$

We can bound the deviation from the expectations above with the help of the variance.

Lemma 7.17

(a) For all $i \in \{1, \ldots, m\}$, $\frac{\mathrm{V}[R_i]}{\mathrm{E}[R_i]^2} \leq \frac{\epsilon^2}{37m}$.

(b) $\frac{\mathrm{V}[Z]}{\mathrm{E}[Z]^2} \leq \frac{\epsilon^2}{36}$.

Proof. (a): Since all $X_\tau^{(i)}$ have the same distribution,

$$\mathrm{V}[R_i] = \mathrm{V}\left[\frac{1}{T}\sum_{\tau=1}^{T} X_\tau^{(i)}\right] = \frac{1}{T^2}\sum_{\tau=1}^{T}\mathrm{V}[X_\tau^{(i)}] = \frac{1}{T}\cdot\mathrm{V}[X_1^{(i)}]\,.$$

$X_1^{(i)}$ is a binary random variable, and therefore $\mathrm{V}[X_1^{(i)}] = \mathrm{E}[X_1^{(i)}]\cdot(1-\mathrm{E}[X_1^{(i)}])$. With $\mathrm{E}[R_i] = \mathrm{E}[X_1^{(i)}]$ and $\mathrm{E}[R_i] \geq 1/3$ we get

$$\frac{\mathrm{V}[R_i]}{\mathrm{E}[R_i]^2} = \frac{1}{T}\left(\frac{1}{\mathrm{E}[R_i]} - 1\right) \leq \frac{2}{T} = \frac{\epsilon^2}{37m}\,.$$

(b): From Chapter 5 we know that

$$\begin{aligned}\frac{\mathrm{V}[Z]}{\mathrm{E}[Z]^2} &= -1 + \prod_{i=1}^{m}\left(1 + \frac{\mathrm{V}[R_i]}{\mathrm{E}[R_i]^2}\right)\\ &\overset{(a)}{\leq} -1 + \prod_{i=1}^{m}\left(1+\frac{\epsilon^2}{37m}\right) = \left(1+\frac{\epsilon^2}{37m}\right) - 1 \leq \frac{\epsilon^2}{36}\,.\end{aligned}$$

□

Now we can use the Chebychev inequality to determine the quality of $\textsc{CountCol}_k$.

Theorem 7.18 *For every $\epsilon \in (0,1)$, $\textsc{CountCol}_k$ runs in time $O((\frac{nm}{\epsilon})^2 \ln(\frac{nm}{\epsilon}))$ and achieves*

$$\Pr[(1-\epsilon)\cdot\#(G) \leq \textsc{CountCol}_k(G,\epsilon) \leq (1+\epsilon)\cdot\#(G)] \geq \frac{3}{4}\,.$$

Proof. The runtime bound is straight forward.

It holds that $\textsc{CountCol}_k(G,\epsilon) = k^n \cdot Z$. Lemma 7.17(b) implies that $\sigma[Z] \leq \frac{\epsilon}{6}\cdot\mathrm{E}[Z]$. This together with Lemma 7.16(b) and the Chebychev inequality implies:

$$\begin{aligned}&\Pr[|Z-\mathrm{E}[Z]| \geq 2\cdot\sigma[Z]] \leq \frac{1}{4}\\ \Rightarrow\ &\Pr[|Z-\mathrm{E}[Z]| \leq 2\cdot\sigma[Z]] \geq \frac{3}{4}\\ \Rightarrow\ &\Pr[\mathrm{E}[Z]-2\sigma[Z] \leq Z \leq \mathrm{E}[Z]+2\sigma[Z]] \geq \frac{3}{4}\\ \Rightarrow\ &\Pr\left[\left(1-\frac{\epsilon}{3}\right)\left(1-\frac{\epsilon}{2}\right)\cdot\#(G) \leq k^n\cdot Z \leq \left(1+\frac{\epsilon}{3}\right)\left(1+\frac{\epsilon}{2}\right)\cdot\#(G)\right] \geq \frac{3}{4}\\ \overset{(*)}{\Rightarrow}\ &\Pr[(1-\epsilon)\cdot\#(G) \leq k^n\cdot Z \leq (1+\epsilon)\cdot\#(G)] \geq \frac{3}{4}\,.\end{aligned}$$

In $(*)$ we used the fact that $1-\epsilon \leq (1-\epsilon/2)(1-\epsilon/3)$ and that $1+\epsilon \geq (1+\epsilon/2)(1+\epsilon/3)$ for all $\epsilon \in [0,1]$. □

Hence, $\textsc{ColCount}_k$ is a strongly polynomial approximation scheme (FPRASC) for $\#\mathrm{COL}_k$ and therefore can be transformed according to Theorem 7.3 into an (ϵ,δ)-FPRASC.

A Markov chain that mixes faster than the Markov chain used in our algorithm can be found in [DG98]. Also there the change of a color is very easy: Instead of choosing a node, an edge $\{u,w\} \in E$ is selected. If $A = \{(c_1,c_2) \mid c_1$ is an available color at u and c_2 is an available color for $w\}$, then a pair is selected from A uniformly at random and applied to u and w. This Markov chain is even ergodic for $k \geq \Delta(G)+1$.

7.6 Further applications of the Markov chain Monte Carlo method

A major breakthrough in the application of Markov chain Monte Carlo methods for approximation problems was the algorithm by Sinclair [Sin93] that provides an FPRASC for #MATCHING – the problem of determining the number of matchings in a graph. Sinclair was able to show for a random walk that transforms one matching into another that it rapidly mixes. The basic approach of his algorithm is similar to the approach we presented for $\#\mathrm{COL}_k$. Detailed descriptions of the random walk and its analysis can also be found in [JS96] and [MR95].

Further applications of the Markov chain Monte Carlo method have been

- determining the number of feasible knapsack fillings [JS96, MS99],
- determining the number of Hamiltonian cycles in dense graphs (i.e. graphs with many edges) [DFJ98], and
- determining the number of independent sets in sparse graphs (i.e. graphs with few edges) [DFJ99]

For all of these problems, FPRASCs are known.

References

[Bol98] B. Bollobas. *Modern Graph Theory.* Springer, New York, 1998.

[DFJ98] M. Dyer, A. Frieze, and M. Jerrum. Approximately counting Hamiltonian paths and cycles in dense graphs. *SIAM Journal on Computing*, 27:1262–1272, 1998.

[DFJ99] M. Dyer, A. Frieze, and M. Jerrum. On counting independent sets in sparse graphs. In *Proc. of the 40th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 210–217, 1999.

[DG98] M. Dyer and C. Greenhill. A more rapidly mixing markov chain for graph colorings. *Random Structures and Algorithms*, 13:285–317, 1998.

[Fel70] W. Feller. *An Introduction to Probability Theory and Applications.* Wiley, New York, 1970.

[Jer95] M. Jerrum. A very simple algorithm for estimating the number of k-colorings of a low-degree graph. *Random Structures and Algorithms*, 7:157–165, 1995.

[Jer98] M. Jerrum. Mathematical foundations of the Markov chain Monte Carlo method. In *Probabilistic Methods for Algorithmic Discrete Mathematics,* volume 16 of *Algorithms and Combinatorics*, pages 116–165. Springer, 1998.

[JS96] M. Jerrum and A. Sinclair. The markov chain monte-carlo method: An approach to approximate counting and integration. In D.S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 482–520. PWS, 1996.

[KLM89] R.M. Karp, M. Luby, and N. Madras. Monte-carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10:429–448, 1989.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, Cambridge, 1995.

[MS99] B. Morris and A. Sinclair. Random walks on truncated cubes and sampling 0-1 knapsack solutions. In *Proc. of the 40th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 230–240, 1999.

[Sin93] A. Sinclair. *Algorithms for Random Generation & Counting: A Markov Chain Approach.* Birkhäuser, Boston, 1993.

[Wel93] D. Welsh. *Complexity: Knots, Colourings and Counting.* Cambridge University Press, 1993.