DigiSem
Wir beschaffen und
digitalisieren

UNIVERSITÄT
BERN

Universitätsbibliothek Bern

Informationen Digitale Semesterapparate:
www.digisem.unibe.ch
Fragen und Support:
digisem@ub.unibe.ch oder Telefon 031 631 93 26

Probability and Computing

Randomization and Probabilistic Techniques in Algorithms and Data Analysis

Second Edition

Michael Mitzenmacher Eli Upfal

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

One Liberty Plaza, 20th Floor, New York, NY 10006, USA

477 Williamstown Road, Port Melbourne, VIC 3207, Australia

4843/24, 2nd Floor, Ansari Road, Daryaganj, Delhi - 110002, India

79 Anson Road, #06-04/06, Singapore 079906

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning, and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9781107154889
10.1017/9781316651124

© Michael Mitzenmacher and Eli Upfal 2017

This publication is in copyright. Subject to statutory exception and to the provisions of relevant collective licensing agreements, no reproduction of any part may take place without the written permission of Cambridge University Press.

First published 2017

Printed in the United Kingdom by Clays, St Ives plc

A catalogue record for this publication is available from the British Library.

Library of Congress Cataloging in Publication Data
Names: Mitzenmacher, Michael, 1969– author. | Upfal, Eli, 1954– author.
Title: Probability and computing / Michael Mitzenmacher Eli Upfal.
Description: Second edition. | Cambridge, United Kingdom ;
New York, NY, USA : Cambridge University Press, [2017] |
Includes bibliographical references and index.
Identifiers: LCCN 2016041654 | ISBN 9781107154889
Subjects: LCSH: Algorithms. | Probabilities. | Stochastic analysis.
Classification: LCC QA274.M574 2017 | DDC 518/.1 – dc23
LC record available at https://lccn.loc.gov/2016041654

ISBN 978-1-107-15488-9 Hardback

Additional resources for this publication at www.cambridge.org/Mitzenmacher.

Cambridge University Press has no responsibility for the persistence or accuracy of URLs for external or third-party Internet Web sites referred to in this publication and does not guarantee that any content on such Web sites is, or will remain, accurate or appropriate.

CHAPTER SIX

The Probabilistic Method

The *probabilistic method* is a way of proving the existence of objects. The underlying principle is simple: to prove the existence of an object with certain properties, we demonstrate a sample space of objects in which the probability is positive that a randomly selected object has the required properties. If the probability of selecting an object with the required properties is positive, then the sample space must contain such an object, and therefore such an object exists. For example, if there is a positive probability of winning a million-dollar prize in a raffle, then there must be at least one raffle ticket that wins that prize.

Although the basic principle of the probabilistic method is simple, its application to specific problems often involves sophisticated combinatorial arguments. In this chapter we study a number of techniques for constructing proofs based on the probabilistic method, starting with simple counting and averaging arguments and then introducing two more advanced tools, the Lovász local lemma and the second moment method.

In the context of algorithms we are generally interested in explicit constructions of objects, not merely in proofs of existence. In many cases the proofs of existence obtained by the probabilistic method can be converted into efficient randomized construction algorithms. In some cases, these proofs can be converted into efficient deterministic construction algorithms; this process is called *derandomization*, since it converts a probabilistic argument into a deterministic one. We give examples of both randomized and deterministic construction algorithms arising from the probabilistic method.

6.1. The Basic Counting Argument

To prove the existence of an object with specific properties, we construct an appropriate probability space $\mathcal{S}$ of objects and then show that the probability that an object in $\mathcal{S}$ with the required properties is selected is strictly greater than 0.

For our first example, we consider the problem of coloring the edges of a graph with two colors so that there are no large cliques with all edges having the same color.

Let K_n be a complete graph (with all $\binom{n}{2}$ edges) on n vertices. A clique of k vertices in K_n is a complete subgraph K_k.

Theorem 6.1: *If* $\binom{n}{k}2^{-\binom{k}{2}+1} < 1$ *then it is possible to color the edges of* K_n *with two colors so that it has no monochromatic* K_k *subgraph.*

Proof: Define a sample space consisting of all possible colorings of the edges of K_n using two colors. There are $2^{\binom{n}{2}}$ possible colorings, so if one is chosen uniformly at random then the probability of choosing each coloring in our probability space is $2^{-\binom{n}{2}}$. A nice way to think about this probability space is: if we color each edge of the graph independently, with each edge taking each of the two colors with probability 1/2, then we obtain a random coloring chosen uniformly from this sample space. That is, we flip an independent fair coin to determine the color of each edge.

Fix an arbitrary ordering of all of the $\binom{n}{k}$ different k-vertex cliques of K_n, and for $i = 1, \ldots, \binom{n}{k}$ let A_i be the event that clique i is monochromatic. Once the first edge in clique i is colored, the remaining $\binom{k}{2} - 1$ edges must all be given the same color. It follows that

$$\Pr(A_i) = 2^{-\binom{k}{2}+1}.$$

Using a union bound then yields

$$\Pr\left(\bigcup_{i=1}^{\binom{n}{k}} A_i\right) \leq \sum_{i=1}^{\binom{n}{k}} \Pr(A_i) = \binom{n}{k} 2^{-\binom{k}{2}+1} < 1,$$

where the last inequality follows from the assumptions of the theorem. Hence

$$\Pr\left(\bigcap_{i=1}^{\binom{n}{k}} \overline{A_i}\right) = 1 - \Pr\left(\bigcup_{i=1}^{\binom{n}{k}} A_i\right) > 0.$$

Since the probability of choosing a coloring with no monochromatic k-vertex clique from our sample space is strictly greater than 0, there must exist a coloring with no monochromatic k-vertex clique. ∎

As an example, consider whether the edges of K_{1000} can be 2-colored in such a way that there is no monochromatic K_{20}. Our calculations are simplified if we note that, for $n \leq 2^{k/2}$ and $k \geq 3$,

$$\begin{aligned}\binom{n}{k} 2^{-\binom{k}{2}+1} &\leq \frac{n^k}{k!} 2^{-(k(k-1)/2)+1} \\ &\leq \frac{2^{k/2+1}}{k!} \\ &< 1.\end{aligned}$$

Observing that for our example $n = 1000 \leq 2^{10} = 2^{k/2}$, we see that by Theorem 6.1 there exists a 2-coloring of the edges of K_{1000} with no monochromatic K_{20}.

Can we use this proof to design an efficient algorithm to construct such a coloring? Let us consider a general approach that gives a randomized construction algorithm. First, we require that we can efficiently sample a coloring from the sample space. In this case sampling is easy, because we can simply color each edge independently with a randomly chosen color. In general, however, there might not be an efficient sampling algorithm.

If we have an efficient sampling algorithm, the next question is: How many samples must we generate before obtaining a sample that satisfies our requirements? If the probability of obtaining a sample with the desired properties is p and if we sample independently at each trial, then the number of samples needed before finding a sample with the required properties is a geometric random variable with expectation $1/p$. Hence we need that $1/p$ be polynomial in the problem size in order to have an algorithm that finds a suitable sample in polynomial expected time.

If $p = 1 - o(1)$, then sampling once gives a Monte Carlo construction algorithm that is incorrect with probability $o(1)$. In our specific example of finding a coloring on a graph of 1000 vertices with no monochromatic K_{20}, we know that the probability that a random coloring has a monochromatic K_{20} is at most

$$\frac{2^{20/2+1}}{20!} < 8.5 \cdot 10^{-16}.$$

Hence we have a Monte Carlo algorithm with a small probability of failure.

If we want a Las Vegas algorithm – that is, one that always gives a correct construction – then we need a third ingredient. We require a polynomial time procedure for verifying that a sample object satisfies the requirements; then we can test samples until we find one that does so. An upper bound on the expected time for this construction can be found by multiplying together the expected number of samples $1/p$ by the sum of an upper bound on the time to generate each sample and an upper bound on the time to check each sample.[1] For the coloring problem, there is a polynomial time verification algorithm when k is a constant: simply check all $\binom{n}{k}$ cliques and make sure they are not monochromatic. It does not seem that this approach can be extended to yield polynomial time algorithms when k grows with n.

6.2. The Expectation Argument

As we have seen, in order to prove that an object with certain properties exists, we can design a probability space from which an element chosen at random yields an object with the desired properties with positive probability. A similar and sometimes easier approach for proving that such an object exists is to use an averaging argument. The intuition behind this approach is that, in a discrete probability space, a random variable must with positive probability assume at least one value that is no greater than its expectation and at least one value that is not smaller than its expectation.

[1] Sometimes the time to generate or check a sample may itself be a random variable. In this case, Wald's equation (discussed in Chapter 13) may apply.

For example, if the expected value of a raffle ticket is \$3, then there must be at least one ticket that ends up being worth no more than \$3 and at least one that ends up being worth no less than \$3.

More formally, we have the following lemma.

Lemma 6.2: *Suppose we have a probability space S and a random variable X defined on S such that* $\mathrm{E}[X] = \mu$. *Then* $\Pr(X \geq \mu) > 0$ *and* $\Pr(X \leq \mu) > 0$.

Proof: We have

$$\mu = \mathrm{E}[X] = \sum_x x \Pr(X = x),$$

where the summation ranges over all values in the range of X. If $\Pr(X \geq \mu) = 0$, then

$$\mu = \sum_x x \Pr(X = x) = \sum_{x<\mu} x \Pr(X = x) < \sum_{x<\mu} \mu \Pr(X = x) = \mu,$$

giving a contradiction. Similarly, if $\Pr(X \leq \mu) = 0$ then

$$\mu = \sum_x x \Pr(X = x) = \sum_{x>\mu} x \Pr(X = x) > \sum_{x>\mu} \mu \Pr(X = x) = \mu,$$

again yielding a contradiction. ■

Thus, there must be at least one instance in the sample space of S for which the value of X is at least μ and at least one instance for which the value of X is no greater than μ.

6.2.1. *Application: Finding a Large Cut*

We consider the problem of finding a large cut in an undirected graph. A cut is a partition of the vertices into two disjoint sets, and the *value* of a cut is the weight of all edges crossing from one side of the partition to the other. Here we consider the case where all edges in the graph have the same weight 1. The problem of finding a maximum cut is NP-hard. Using the probabilistic method, we show that the value of the maximum cut must be at least $1/2$ the number of edges in the graph.

Theorem 6.3: *Given an undirected graph G with m edges, there is a partition of V into two disjoint sets A and B such that at least $m/2$ edges connect a vertex in A to a vertex in B. That is, there is a cut with value at least $m/2$.*

Proof: Construct sets A and B by randomly and independently assigning each vertex to one of the two sets. Let $e_1, \ldots, e_m$ be an arbitrary enumeration of the edges of G. For $i = 1, \ldots, m$, define X_i such that

$$X_i = \begin{cases} 1 & \text{if edge } i \text{ connects } A \text{ to } B, \\ 0 & \text{otherwise.} \end{cases}$$

The probability that edge e_i connects a vertex in A to a vertex in B is $1/2$, and thus

$$\mathrm{E}[X_i] = \frac{1}{2}.$$

Let $C(A, B)$ be a random variable denoting the value of the cut corresponding to the sets A and B. Then

$$\mathbf{E}[C(A,B)] = \mathbf{E}\left[\sum_{i=1}^{m} X_i\right] = \sum_{i=1}^{m} \mathbf{E}[X_i] = m \cdot \frac{1}{2} = \frac{m}{2}.$$

Since the expectation of the random variable $C(A, B)$ is $m/2$, there exists a partition A and B with at least $m/2$ edges connecting the set A to the set B. ■

We can transform this argument into an efficient algorithm for finding a cut with value at least $m/2$. We first show how to obtain a Las Vegas algorithm. In Section 6.3, we show how to construct a deterministic polynomial time algorithm.

It is easy to randomly choose a partition as described in the proof. The expectation argument does not give a lower bound on the probability that a random partition has a cut of value at least $m/2$. To derive such a bound, let

$$p = \Pr\left(C(A,B) \geq \frac{m}{2}\right),$$

and observe that $C(A, B) \leq m$. Then

$$\begin{aligned}
\frac{m}{2} &= \mathbf{E}[C(A,B)] \\
&= \sum_{i<m/2} i \Pr(C(A,B) = i) + \sum_{i \geq m/2} i \Pr(C(A,B) = i) \\
&\leq (1-p)\left(\frac{m}{2} - 1\right) + pm,
\end{aligned}$$

which implies that

$$p \geq \frac{1}{m/2+1}.$$

The expected number of samples before finding a cut with value at least $m/2$ is therefore just $m/2 + 1$. Testing to see if the value of the cut determined by the sample is at least $m/2$ can be done in polynomial time simply by counting the edges crossing the cut. We therefore have a Las Vegas algorithm for finding the cut.

6.2.2. *Application: Maximum Satisfiability*

We can apply a similar argument to the maximum satisfiability (MAXSAT) problem. In a logical formula, a *literal* is either a Boolean variable or the negation of a Boolean variable. We use $\bar{x}$ to denote the negation of the variable x. A satisfiability (SAT) problem, or a SAT formula, is a logical expression that is the conjunction (AND) of a set of clauses, where each clause is the disjunction (OR) of literals. For example, the following expression is an instance of SAT:

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1}).$$

A solution to an instance of a SAT formula is an assignment of the variables to the values True and False so that all the clauses are satisfied. That is, there is at least one true literal in each clause. For example, assigning x_1 to True, x_2 to False, x_3 to False, and x_4 to True satisfies the preceding SAT formula. In general, determining if a SAT formula has a solution is NP-hard.

A related goal, given a SAT formula, is satisfying as many of the clauses as possible. In what follows, let us assume that no clause contains both a variable and its complement, since in this case the clause is always satisfied.

Theorem 6.4: *Given a set of m clauses, let k_i be the number of literals in the ith clause for $i = 1, \ldots, m$. Let $k = \min_{i=1}^{m} k_i$. Then there is a truth assignment that satisfies at least*

$$\sum_{i=1}^{m}(1 - 2^{-k_i}) \geq m(1 - 2^{-k})$$

clauses.

Proof: Assign values independently and uniformly at random to the variables. The probability that the ith clause with k_i literals is satisfied is at least $(1 - 2^{-k_i})$. The expected number of satisfied clauses is therefore at least

$$\sum_{i=1}^{m}(1 - 2^{-k_i}) \geq m(1 - 2^{-k}),$$

and there must be an assignment that satisfies at least that many clauses. ■

The foregoing argument can also be easily transformed into an efficient randomized algorithm; the case where all $k_i = k$ is left as Exercise 6.1.

6.3. Derandomization Using Conditional Expectations

The probabilistic method can yield insight into how to construct deterministic algorithms. As an example, we apply the *method of conditional expectations* in order to *derandomize* the algorithm of Section 6.2.1 for finding a large cut.

Recall that we find a partition of the n vertices V of a graph into sets A and B by placing each vertex independently and uniformly at random in one of the two sets. This gives a cut with expected value $\mathbf{E}[C(A, B)] \geq m/2$. Now imagine placing the vertices deterministically, one at a time, in an arbitrary order $v_1, v_2, \ldots, v_n$. Let x_i be the set where v_i is placed (so x_i is either A or B). Suppose that we have placed the first k vertices, and consider the expected value of the cut if the *remaining* vertices are then placed independently and uniformly into one of the two sets. We write this quantity as $\mathbf{E}[C(A, B) \mid x_1, x_2, \ldots, x_k]$; it is the conditional expectation of the value of the cut given the locations $x_1, x_2, \ldots, x_k$ of the first k vertices. We show inductively how to place the next vertex so that

$$\mathbf{E}[C(A, B) \mid x_1, x_2, \ldots, x_k] \leq \mathbf{E}[C(A, B) \mid x_1, x_2, \ldots, x_{k+1}].$$

It follows that

$$\mathbf{E}[C(A, B)] \leq \mathbf{E}[C(A, B) \mid x_1, x_2, \ldots, x_n].$$

The right-hand side is the value of the cut determined by our placement algorithm, since if $x_1, x_2, \ldots, x_n$ are all determined then we have a cut of the graph. Hence our algorithm returns a cut whose value is at least $\mathbf{E}[C(A, B)] \geq m/2$.

The base case in the induction is

$$\mathbf{E}[C(A, B) \mid x_1] = \mathbf{E}[C(A, B)],$$

which holds by symmetry because it does not matter where we place the first vertex.

We now prove the inductive step, that

$$\mathbf{E}[C(A, B) \mid x_1, x_2, \ldots, x_k] \leq \mathbf{E}[C(A, B) \mid x_1, x_2, \ldots, x_{k+1}]. \tag{6.1}$$

Consider placing v_{k+1} randomly, so that it is placed in A or B with probability $1/2$ each, and let Y_{k+1} be a random variable representing the set where it is placed. Then

$$\begin{aligned}\mathbf{E}[C(A, B) \mid x_1, x_2, \ldots, x_k] = &\frac{1}{2}\mathbf{E}[C(A, B) \mid x_1, x_2, \ldots, x_k, Y_{k+1} = A] \\ &+ \frac{1}{2}\mathbf{E}[C(A, B) \mid x_1, x_2, \ldots, x_k, Y_{k+1} = B].\end{aligned}$$

It follows that

$$\begin{aligned}&\max\left(\mathbf{E}[C(A, B) \mid x_1, x_2, \ldots, x_k, Y_{k+1} = A], \mathbf{E}[C(A, B) \mid x_1, x_2, \ldots, x_k, Y_{k+1} = B]\right) \\ &\quad \geq \mathbf{E}[C(A, B) \mid x_1, x_2, \ldots, x_k].\end{aligned}$$

Therefore, all we have to do is compute the two quantities $\mathbf{E}[C(A, B) \mid x_1, x_2, \ldots, x_k, Y_{k+1} = A]$ and $\mathbf{E}[C(A, B) \mid x_1, x_2, \ldots, x_k, Y_{k+1} = B]$ and then place the v_{k+1} in the set that yields the larger expectation. Once we do this, we will have a placement satisfying

$$\mathbf{E}[C(A, B) \mid x_1, x_2, \ldots, x_k] \leq \mathbf{E}[C(A, B) \mid x_1, x_2, \ldots, x_{k+1}].$$

To compute $\mathbf{E}[C(A, B) \mid x_1, x_2, \ldots, x_k, Y_{k+1} = A]$, note that the conditioning gives the placement of the first $k + 1$ vertices. We can therefore compute the number of edges among these vertices that contribute to the value of the cut. For all other edges, the probability that it will later contribute to the cut is $1/2$, since this is the probability its two endpoints end up on different sides of the cut. By linearity of expectations, $\mathbf{E}[C(A, B) \mid x_1, x_2, \ldots, x_k, Y_{k+1} = A]$ is the number of edges crossing the cut whose endpoints are both among the first $k + 1$ vertices, plus half of the remaining edges. This is easy to compute in linear time. The same is true for $\mathbf{E}[C(A, B) \mid x_1, x_2, \ldots, x_k, Y_{k+1} = B]$.

In fact, from this argument, we see that the larger of the two quantities is determined just by whether v_{k+1} has more neighbors in A or in B. All edges that do not have v_{k+1} as an endpoint contribute the same amount to the two expectations. Our derandomized algorithm therefore has the following simple form: Take the vertices in some order. Place the first vertex arbitrarily in A. Place each successive vertex to maximize the number of edges crossing the cut. Equivalently, place each vertex on the side with

fewer neighbors, breaking ties arbitrarily. This is a simple greedy algorithm, and our analysis shows that it always guarantees a cut with at least $m/2$ edges.

6.4. Sample and Modify

Thus far we have used the probabilistic method to construct random structures with the desired properties directly. In some cases it is easier to work indirectly, breaking the argument into two stages. In the first stage we construct a random structure that does not have the required properties. In the second stage we then modify the random structure so that it does have the required property. We give two examples of this sample-and-modify technique.

6.4.1. *Application: Independent Sets*

An *independent set* in a graph G is a set of vertices with no edges between them. Finding the largest independent set in a graph is an NP-hard problem. The following theorem shows that the probabilistic method can yield bounds on the size of the largest independent set of a graph.

Theorem 6.5: *Let $G = (V, E)$ be a connected graph on n vertices with $m \geq n/2$ edges. Then G has an independent set with at least $n^2/4m$ vertices.*

Proof: Let $d = 2m/n \geq 1$ be the average degree of the vertices in G. Consider the following randomized algorithm.

1. Delete each vertex of G (together with its incident edges) independently with probability $1 - 1/d$.
2. For each remaining edge, remove it and one of its adjacent vertices.

The remaining vertices form an independent set, since all edges have been removed. This is an example of the sample-and-modify technique. We first sample the vertices, and then we modify the remaining graph.

Let X be the number of vertices that survive the first step of the algorithm. Since the graph has n vertices and since each vertex survives with probability $1/d$, it follows that

$$\mathrm{E}[X] = \frac{n}{d}.$$

Let Y be the number of edges that survive the first step. There are $nd/2$ edges in the graph, and an edge survives if and only if its two adjacent vertices survive. Thus

$$\mathrm{E}[Y] = \frac{nd}{2}\left(\frac{1}{d}\right)^2 = \frac{n}{2d}.$$

The second step of the algorithm removes all the remaining edges and at most Y vertices. When the algorithm terminates, it outputs an independent set of size at least

$X - Y$, and

$$\mathbf{E}[X - Y] = \frac{n}{d} - \frac{n}{2d} = \frac{n}{2d}.$$

The expected size of the independent set generated by the algorithm is at least $n/2d$, so the graph has an independent set with at least $n/2d = n^2/4m$ vertices. ■

6.4.2. *Application: Graphs with Large Girth*

As another example we consider the *girth* of a graph, which is the length of its smallest cycle. Intuitively we expect dense graphs to have small girth. We can show, however, that there are dense graphs with relatively large girth.

Theorem 6.6: *For any integer $k \geq 3$, for n sufficiently large there is a graph with n nodes, at least $\frac{1}{4}n^{1+1/k}$ edges, and girth at least k.*

Proof: We first sample a random graph $G \in G_{n,p}$ with $p = n^{1/k-1}$. Let X be the number of edges in the graph. Then

$$\mathbf{E}[X] = p\binom{n}{2} = \frac{1}{2}\left(1 - \frac{1}{n}\right)n^{1/k+1}.$$

Let Y be the number of cycles in the graph of length at most $k - 1$. Any specific possible cycle of length i, where $3 \leq i \leq k - 1$, occurs with probability p^i. Also, there are $\binom{n}{i}\frac{(i-1)!}{2}$ possible cycles of length i; to see this, first consider choosing the i vertices, then consider the possible orders, and finally keep in mind that reversing the order yields the same cycle. Hence,

$$\mathbf{E}[Y] = \sum_{i=3}^{k-1}\binom{n}{i}\frac{(i-1)!}{2}p^i \leq \sum_{i=3}^{k-1} n^i p^i = \sum_{i=3}^{k-1} n^{i/k} < kn^{(k-1)/k}.$$

We modify the original randomly chosen graph G by eliminating one edge from each cycle of length up to $k - 1$. The modified graph therefore has girth at least k. When n is sufficiently large, the expected number of edges in the resulting graph is

$$\mathbf{E}[X - Y] \geq \frac{1}{2}\left(1 - \frac{1}{n}\right)n^{1/k+1} - kn^{(k-1)/k} \geq \frac{1}{4}n^{1/k+1}.$$

Hence there exists a graph with at least $\frac{1}{4}n^{1+1/k}$ edges and girth at least k. ■

6.5. The Second Moment Method

The second moment method is another useful way to apply the probabilistic method. The standard approach typically makes use of the following inequality, which is easily derived from Chebyshev's inequality.

Theorem 6.7: *If X is an integer-valued random variable, then*

$$\Pr(X = 0) \leq \frac{\mathbf{Var}[X]}{(\mathbf{E}[X])^2}. \tag{6.2}$$

Proof:

$$\Pr(X = 0) \leq \Pr(|X - \mathbf{E}[X]| \geq \mathbf{E}[X]) \leq \frac{\mathbf{Var}[X]}{(\mathbf{E}[X])^2}.$$ ■

6.5.1. *Application: Threshold Behavior in Random Graphs*

The second moment method can be used to prove the threshold behavior of certain random graph properties. That is, in the $G_{n,p}$ model it is often the case that there is a *threshold function* f such that: (a) when p is just less than $f(n)$, almost no graph has the desired property; whereas (b) when p is just larger than $f(n)$, almost every graph has the desired property. We present here a relatively simple example.

Theorem 6.8: *In $G_{n,p}$, suppose that $p = f(n)$, where $f(n) = o(n^{-2/3})$. Then, for any $\varepsilon > 0$ and for sufficiently large n, the probability that a random graph chosen from $G_{n,p}$ has a clique of four or more vertices is less than ε. Similarly, if $f(n) = \omega(n^{-2/3})$ then, for sufficiently large n, the probability that a random graph chosen from $G_{n,p}$ does not have a clique with four or more vertices is less than ε.*

Proof: We first consider the case in which $p = f(n)$ and $f(n) = o(n^{-2/3})$. Let $C_1, \ldots, C_{\binom{n}{4}}$ be an enumeration of all the subsets of four vertices in G. Let

$$X_i = \begin{cases} 1 & \text{if } C_i \text{ is a 4-clique,} \\ 0 & \text{otherwise.} \end{cases}$$

Let

$$X = \sum_{i=1}^{\binom{n}{4}} X_i,$$

so that

$$\mathbf{E}[X] = \binom{n}{4} p^6.$$

In this case $\mathbf{E}[X] = o(1)$, which means that $\mathbf{E}[X] < \varepsilon$ for sufficiently large n. Since X is a nonnegative integer-valued random variable, it follows that $\Pr(X \geq 1) \leq \mathbf{E}[X] < \varepsilon$. Hence, the probability that a random graph chosen from $G_{n,p}$ has a clique of four or more vertices is less than ε.

We now consider the case when $p = f(n)$ and $f(n) = \omega(n^{-2/3})$. In this case, $\mathbf{E}[X] \to \infty$ as n grows large. This in itself is not sufficient to conclude that, with high probability, a graph chosen random from $G_{n,p}$ has a clique of at least four vertices. We can, however, use Theorem 6.7 to prove that $\Pr(X = 0) = o(1)$ in this case. To do so we must show that $\mathbf{Var}[X] = o((\mathbf{E}[X])^2)$. Here we shall bound the variance directly; an alternative approach is given as Exercise 6.12.

We begin with the following useful formula.

Lemma 6.9: *Let Y_i, $i = 1, \ldots, m$, be 0–1 random variables, and let $Y = \sum_{i=1}^{m} Y_i$. Then*

$$\mathbf{Var}[Y] \leq \mathbf{E}[Y] + \sum_{1 \leq i,j \leq m; i \neq j} \mathbf{Cov}(Y_i, Y_j).$$

Proof: For any sequence of random variables $Y_1, \ldots, Y_m$,

$$\mathbf{Var}\left[\sum_{i=1}^{m} Y_i\right] = \sum_{i=1}^{m} \mathbf{Var}[Y_i] + \sum_{1 \le i,j \le m; i \ne j} \mathbf{Cov}(Y_i, Y_j).$$

This is the generalization of Theorem 3.2 to m variables.

When Y_i is a 0–1 random variable, $\mathbf{E}[Y_i^2] = \mathbf{E}[Y_i]$ and so

$$\mathbf{Var}[Y_i] = \mathbf{E}\left[Y_i^2\right] - (\mathbf{E}[Y_i])^2 \le \mathbf{E}[Y_i],$$

giving the lemma. ■

We wish to compute

$$\mathbf{Var}[X] = \mathbf{Var}\left[\sum_{i=1}^{\binom{n}{4}} X_i\right].$$

Applying Lemma 6.9, we see that we need to consider the covariance of the X_i. If $|C_i \cap C_j| = 0$ then the corresponding cliques are disjoint, and it follows that X_i and X_j are independent. Hence, in this case, $\mathbf{E}[X_iX_j] - \mathbf{E}[X_i]\mathbf{E}[X_j] = 0$. The same is true if $|C_i \cap C_j| = 1$.

If $|C_i \cap C_j| = 2$, then the corresponding cliques share one edge. For both cliques to be in the graph, the eleven corresponding edges must appear in the graph. Hence, in this case $\mathbf{E}[X_iX_j] - \mathbf{E}[X_i]\mathbf{E}[X_j] \le \mathbf{E}[X_iX_j] \le p^{11}$. There are $\binom{n}{6}$ ways to choose the six vertices and $\binom{6}{2;2;2}$ ways to split them into C_i and C_j (because we choose two vertices for $C_i \cap C_j$, two for C_i alone, and two for C_j alone).

If $|C_i \cap C_j| = 3$, then the corresponding cliques share three edges. For both cliques to be in the graph, the nine corresponding edges must appear in the graph. Hence, in this case $\mathbf{E}[X_iX_j] - \mathbf{E}[X_i]\mathbf{E}[X_j] \le \mathbf{E}[X_iX_j] \le p^9$. There are $\binom{n}{5}$ ways to choose the five vertices, and $\binom{5}{3;1;1}$ ways to split them into C_i and C_j.

Finally, recall again that $\mathbf{E}[X] = \binom{n}{4}p^6$ and $p = f(n) = \omega(n^{-2/3})$. Therefore,

$$\mathbf{Var}[X] \le \binom{n}{4}p^6 + \binom{n}{6}\binom{6}{2;2;2}p^{11} + \binom{n}{5}\binom{5}{3;1;1}p^9 = o(n^8p^{12}) = o((\mathbf{E}[X])^2),$$

since

$$(\mathbf{E}[X])^2 = \left(\binom{n}{4}p^6\right)^2 = \Theta(n^8p^{12}).$$

Theorem 6.7 now applies, showing that $\Pr(X = 0) = o(1)$ and thus the second part of the theorem. ■

6.6. The Conditional Expectation Inequality

For a sum of Bernoulli random variables, we can derive an alternative to the second moment method that is often easier to apply.

Theorem 6.10: *Let $X = \sum_{i=1}^{n} X_i$, where each X_i is a 0–1 random variable. Then*

$$\Pr(X > 0) \geq \sum_{i=1}^{n} \frac{\Pr(X_i = 1)}{\mathbf{E}[X \mid X_i = 1]}. \tag{6.3}$$

Notice that the X_i need not be independent for Eqn. (6.3) to hold.

Proof: Let $Y = 1/X$ if $X > 0$, with $Y = 0$ otherwise. Then

$$\Pr(X > 0) = \mathbf{E}[XY].$$

However,

$$\begin{aligned}
\mathbf{E}[XY] &= \mathbf{E}\left[\sum_{i=1}^{n} X_i Y\right] \\
&= \sum_{i=1}^{n} \mathbf{E}[X_i Y] \\
&= \sum_{i=1}^{n} \left(\mathbf{E}[X_i Y \mid X_i = 1] \Pr(X_i = 1) + \mathbf{E}[X_i Y \mid X_i = 0] \Pr(X_i = 0)\right) \\
&= \sum_{i=1}^{n} \mathbf{E}[Y \mid X_i = 1] \Pr(X_i = 1) \\
&= \sum_{i=1}^{n} \mathbf{E}[1/X \mid X_i = 1] \Pr(X_i = 1) \\
&\geq \sum_{i=1}^{n} \frac{\Pr(X_i = 1)}{\mathbf{E}[X \mid X_i = 1]}.
\end{aligned}$$

The key step is from the third to the fourth line, where we use conditional expectations in a fruitful way by taking advantage of the fact that $\mathbf{E}[X_i Y \mid X_i = 0] = 0$. The last line makes use of Jensen's inequality, with the convex function $f(x) = 1/x$. ■

We can use Theorem 6.10 to give an alternate proof of Theorem 6.8. Specifically, if $p = f(n) = \omega(n^{-2/3})$, we use Theorem 6.10 to show that, for any constant $\varepsilon > 0$ and for sufficiently large n, the probability that a random graph chosen from $G_{n,p}$ does not have a clique with four or more vertices is less than ε.

As in the proof of Theorem 6.8, let $X = \sum_{i=1}^{\binom{n}{4}} X_i$, where X_i is 1 if the subset of four vertices C_i is a 4-clique and 0 otherwise. For a specific X_j, we have $\Pr(X_j = 1) = p^6$. Using the linearity of expectations, we compute

$$\mathbf{E}[X \mid X_j = 1] = \mathbf{E}\left[\sum_{i=1}^{\binom{n}{4}} X_i \,\middle|\, X_j = 1\right] = \sum_{i=1}^{\binom{n}{4}} \mathbf{E}[X_i \mid X_j = 1].$$

Conditioning on $X_j = 1$, we now compute $\mathbf{E}[X_i \mid X_j = 1]$ by using that, for a 0–1 random variable,

$$\mathbf{E}[X_i \mid X_j = 1] = \Pr(X_i = 1 \mid X_j = 1).$$

There are $\binom{n-4}{4}$ sets of vertices C_i that do not intersect C_j. Each corresponding X_i is 1 with probability p^6. Similarly, $X_i = 1$ with probability p^6 for the $4\binom{n-4}{3}$ sets C_i that have one vertex in common with C_j.

For the remaining cases, we have $\Pr(X_i = 1 \mid X_j = 1) = p^5$ for the $6\binom{n-4}{2}$ sets C_i that have two vertices in common with C_j and $\Pr(X_i = 1 \mid X_j = 1) = p^3$ for the $4\binom{n-4}{1}$ sets C_i that have three vertices in common with C_j. Summing, we have

$$\begin{aligned}\mathbf{E}[X \mid X_j = 1] &= \sum_{i=1}^{\binom{n}{4}} \mathbf{E}[X_i \mid X_j = 1] \\ &= 1 + \binom{n-4}{4} p^6 + 4\binom{n-4}{3} p^6 + 6\binom{n-4}{2} p^5 + 4\binom{n-4}{1} p^3.\end{aligned}$$

Applying Theorem 6.10 yields

$$\Pr(X > 0) \geq \frac{\binom{n}{4} p^6}{1 + \binom{n-4}{4} p^6 + 4\binom{n-4}{3} p^6 + 6\binom{n-4}{2} p^5 + 4\binom{n-4}{1} p^3},$$

which approaches 1 as n grows large when $p = f(n) = \omega(n^{-2/3})$.

6.7. The Lovász Local Lemma

One of the most elegant and useful tools in applying the probabilistic method is the Lovász Local Lemma. Let $E_1, \ldots, E_n$ be a set of bad events in some probability space. We want to show that there is an element in the sample space that is not included in any of the bad events.

This would be easy to do if the events were mutually independent. Recall that events $E_1, E_2, \ldots, E_n$ are mutually independent if and only if, for *any* subset $I \subseteq [1, n]$,

$$\Pr\left(\bigcap_{i \in I} E_i\right) = \prod_{i \in I} \Pr(E_i).$$

Also, if $E_1, \ldots, E_n$ are mutually independent then so are $\bar{E}_1, \ldots, \bar{E}_n$. (This was left as Exercise 1.20.) If $\Pr(E_i) < 1$ for all i, then

$$\Pr\left(\bigcap_{i=1}^{n} \bar{E}_i\right) = \prod_{i=1}^{n} \Pr(\bar{E}_i) > 0,$$

and there is an element of the sample space that is not included in any bad event.

Mutual independence is too much to ask for in many arguments. The Lovász local lemma generalizes the preceding argument to the case where the n events are not mutually independent but the dependency is limited. Specifically, following from the definition of mutual independence, we say that an event E_{n+1} is *mutually independent* of

the events $E_1, E_2, \ldots, E_n$ if, for any subset $I \subseteq [1, n]$,

$$\Pr\left(E_{n+1} \mid \bigcap_{j \in I} E_j\right) = \Pr(E_{n+1}).$$

The dependency between events can be represented in terms of a dependency graph.

Definition 6.1: *A* dependency graph *for a set of events $E_1, \ldots, E_n$ is a graph $G = (V, E)$ such that $V = \{1, \ldots, n\}$ and, for $i = 1, \ldots, n$, event E_i is mutually independent of the events $\{E_j \mid (i, j) \notin E\}$. The* degree *of the dependency graph is the maximum degree of any vertex in the graph.*

We discuss first a special case, the symmetric version of the Lovász Local Lemma, which is more intuitive and is sufficient for most algorithmic applications.

Theorem 6.11 [Lovász Local Lemma]: *Let $E_1, \ldots, E_n$ be a set of events, and assume that the following hold:*

1. *for all i, $\Pr(E_i) \leq p$;*
2. *the degree of the dependency graph given by $E_1, \ldots, E_n$ is bounded by d;*
3. $4dp \leq 1$.

Then

$$\Pr\left(\bigcap_{i=1}^{n} \bar{E}_i\right) > 0.$$

Proof: Let $S \subset \{1, \ldots, n\}$. We prove by induction on $s = 0, \ldots, n-1$ that, if $|S| \leq s$, then for all $k \notin S$ we have

$$\Pr\left(E_k \mid \bigcap_{j \in S} \bar{E}_j\right) \leq 2p.$$

For this expression to be well-defined when S is not empty, we need $\Pr\left(\bigcap_{j \in S} \bar{E}_j\right) > 0$.

The base case $s = 0$ follows from the assumption that $\Pr(E_k) \leq p$. To perform the inductive step, we first show that $\Pr\left(\bigcap_{j \in S} \bar{E}_j\right) > 0$. This is true when $s = 1$, because $\Pr(\bar{E}_j) \geq 1 - p > 0$. For $s > 1$, without loss of generality let $S = \{1, 2, \ldots, s\}$. Then

$$\begin{aligned}
\Pr\left(\bigcap_{i=1}^{s} \bar{E}_i\right) &= \prod_{i=1}^{s} \Pr\left(\bar{E}_i \mid \bigcap_{j=1}^{i-1} \bar{E}_j\right) \\
&= \prod_{i=1}^{s} \left(1 - \Pr\left(E_i \mid \bigcap_{j=1}^{i-1} \bar{E}_j\right)\right) \\
&\geq \prod_{i=1}^{s} (1 - 2p) > 0.
\end{aligned}$$

In obtaining the last line we used the induction hypothesis.

For the rest of the induction, let $S_1 = \{j \in S \mid (k, j) \in E\}$ and $S_2 = S - S_1$. If $S_2 = S$ then E_k is mutually independent of the events $\bar{E}_i$, $i \in S$, and

$$\Pr\left(E_k \mid \bigcap_{j \in S} \bar{E}_j\right) = \Pr(E_k) \leq p.$$

We continue with the case $|S_2| < s$. It will be helpful to introduce the following notation. Let F_S be defined by

$$F_S = \bigcap_{j \in S} \bar{E}_j,$$

and similarly define F_{S_1} and F_{S_2}. Notice that $F_S = F_{S_1} \cap F_{S_2}$.

Applying the definition of conditional probability yields

$$\Pr(E_k \mid F_S) = \frac{\Pr(E_k \cap F_S)}{\Pr(F_S)}. \tag{6.4}$$

Applying the definition of conditional probability to the numerator of Eqn. (6.4), we obtain

$$\begin{aligned} \Pr(E_k \cap F_S) &= \Pr(E_k \cap F_{S_1} \cap F_{S_2}) \\ &= \Pr(E_k \cap F_{S_1} \mid F_{S_2}) \Pr(F_{S_2}). \end{aligned}$$

The denominator can be written as

$$\begin{aligned} \Pr(F_S) &= \Pr(F_{S_1} \cap F_{S_2}) \\ &= \Pr(F_{S_1} \mid F_{S_2}) \Pr(F_{S_2}). \end{aligned}$$

Canceling the common factor, which we have already shown to be nonzero, yields

$$\Pr(E_k \mid F_S) = \frac{\Pr(E_k \cap F_{S_1} \mid F_{S_2})}{\Pr(F_{S_1} \mid F_{S_2})}. \tag{6.5}$$

Note that Eqn. (6.5) is valid even when $S_2 = \emptyset$.

Since the probability of an intersection of events is bounded by the probability of any one of the events and since E_k is independent of the events in S_2, we can bound the numerator of Eqn. (6.5) by

$$\Pr(E_k \cap F_{S_1} \mid F_{S_2}) \leq \Pr(E_k \mid F_{S_2}) = \Pr(E_k) \leq p.$$

Because $|S_2| < |S| = s$, we can apply the induction hypothesis to

$$\Pr(E_i \mid F_{S_2}) = \Pr\left(E_i \mid \bigcap_{j \in S_2} \bar{E}_j\right).$$

Using also the fact that $|S_1| \leq d$, we establish a lower bound on the denominator of Eqn. (6.5) as follows:

$$\begin{aligned}
\Pr(F_{S_1} \mid F_{S_2}) &= \Pr\left(\bigcap_{i \in S_1} \bar{E}_i \mid \bigcap_{j \in S_2} \bar{E}_j\right) \\
&\geq 1 - \sum_{i \in S_1} \Pr\left(E_i \mid \bigcap_{j \in S_2} \bar{E}_j\right) \\
&\geq 1 - \sum_{i \in S_1} 2p \\
&\geq 1 - 2pd \\
&\geq \frac{1}{2}.
\end{aligned}$$

Using the upper bound for the numerator and the lower bound for the denominator, we prove the induction:

$$\begin{aligned}
\Pr(E_k \mid F_S) &= \frac{\Pr(E_k \cap F_{S_1} \mid F_{S_2})}{\Pr(F_{S_1} \mid F_{S_2})} \\
&\leq \frac{p}{1/2} = 2p.
\end{aligned}$$

The theorem follows from

$$\begin{aligned}
\Pr\left(\bigcap_{i=1}^{n} \bar{E}_i\right) &= \prod_{i=1}^{n} \Pr\left(\bar{E}_i \mid \bigcap_{j=1}^{i-1} \bar{E}_j\right) \\
&= \prod_{i=1}^{n} \left(1 - \Pr\left(E_i \mid \bigcap_{j=1}^{i-1} \bar{E}_j\right)\right) \\
&\geq \prod_{i=1}^{n} (1 - 2p) > 0.
\end{aligned}$$

■

6.7.1. *Application: Edge-Disjoint Paths*

Assume that n pairs of users need to communicate using edge-disjoint paths on a given network. Each pair $i = 1, \ldots, n$ can choose a path from a collection F_i of m paths. We show using the Lovász local lemma that, if the possible paths do not share too many edges, then there is a way to choose n edge-disjoint paths connecting the n pairs.

Theorem 6.12: *If any path in F_i shares edges with no more than k paths in F_j, where $i \neq j$ and $8nk/m \leq 1$, then there is a way to choose n edge-disjoint paths connecting the n pairs.*

Proof: Consider the probability space defined by each pair choosing a path independently and uniformly at random from its set of m paths. Define $E_{i,j}$ to represent the

event that the paths chosen by pairs i and j share at least one edge. Since a path in F_i shares edges with no more than k paths in F_j,

$$p = \Pr(E_{i,j}) \leq \frac{k}{m}.$$

Let d be the degree of the dependency graph. Since event $E_{i,j}$ is independent of all events $E_{i',j'}$ when $i' \notin \{i, j\}$ and $j' \notin \{i, j\}$, we have $d < 2n$. Since

$$4dp < \frac{8nk}{m} \leq 1,$$

all of the conditions of the Lovász Local Lemma are satisfied, proving

$$\Pr\left(\bigcap_{i \neq j} \bar{E}_{i,j}\right) > 0.$$

Hence, there is a choice of paths such that the n paths are edge disjoint. ■

6.7.2. *Application: Satisfiability*

As a second example, we return to the satisfiability question. For the k-satisfiability (k-SAT) problem, the formula is restricted so that each clause has exactly k literals. Again, we assume that no clause contains both a literal and its negation, as these clauses are trivial. We prove that any k-SAT formula in which no variable appears in too many clauses has a satisfying assignment.

Theorem 6.13: *If no variable in a k-SAT formula appears in more than $T = 2^k/4k$ clauses, then the formula has a satisfying assignment.*

Proof: Consider the probability space defined by giving a random assignment to the variables. For $i = 1, \ldots, m$, let E_i denote the event that the ith clause is not satisfied by the random assignment. Since each clause has k literals,

$$\Pr(E_i) = 2^{-k}.$$

The event E_i is mutually independent of all of the events related to clauses that do not share variables with clause i. Because each of the k variables in clause i can appear in no more than $T = 2^k/4k$ clauses, the degree of the dependency graph is bounded by $d \leq kT \leq 2^{k-2}$.

In this case,

$$4dp \leq 4 \cdot 2^{k-2} 2^{-k} \leq 1,$$

so we can apply the Lovász Local Lemma to conclude that

$$\Pr\left(\bigcap_{i=1}^{m} \bar{E}_i\right) > 0;$$

hence there is a satisfying assignment for the formula. ■

6.8.* Explicit Constructions Using the Local Lemma

The Lovász Local Lemma proves that a random element in an appropriately defined sample space has a nonzero probability of satisfying our requirement. However, this probability might be too small for an algorithm that is based on simple sampling. The number of objects that we need to sample before we find an element that satisfies our requirements might be exponential in the problem size.

In a number of interesting applications, the existential result of the Lovász Local Lemma can be used to derive efficient construction algorithms. Although the details differ in the specific applications, many known algorithms are based on a common two-phase scheme. In the first phase, a subset of the variables of the problem are assigned random values; the remaining variables are deferred to the second stage. The subset of variables that are assigned values in the first stage is chosen so that:

1. using the Local Lemma, one can show that the random partial solution fixed in the first phase can be extended to a full solution of the problem without modifying any of the variables fixed in the first phase; and
2. the dependency graph H between events defined by the variables deferred to the second phase has, with high probability, only small connected components.

When the dependency graph consists of connected components, a solution for the variables of one component can be found independently of the other components. Thus, the first phase of the two-phase algorithm breaks the original problem into smaller subproblems. Each of the smaller subproblems can then be solved independently in the second phase by an exhaustive search.

6.8.1. *Application: A Satisfiability Algorithm*

We demonstrate this technique in an algorithm for finding a satisfying assignment for a k-SAT formula. The explicit construction result will be significantly weaker than the existence result proven in the previous section. In particular, we obtain a polynomial time algorithm only for the case when k is a constant. This result is still interesting, since for $k \geq 3$ the problem of k-satisfiability is NP-complete. For notational convenience we treat here only the case where k is an even constant with $k \geq 12$; the case where k is a sufficiently large odd constant is similar.

Consider a k-SAT formula $\mathcal{F}$, with k an even constant, such that each variable appears in no more than $T = 2^{\alpha k}$ clauses for some constant $\alpha > 0$ determined in the proof. Let $x_1, \ldots, x_\ell$ be the ℓ variables and $C_1, \ldots, C_m$ the m clauses of $\mathcal{F}$.

Following the outline suggested in Section 6.8, our algorithm for finding a satisfying assignment for $\mathcal{F}$ has two phases. Some of the variables are fixed at the first phase, and the remaining variables are deferred to the second phase. While executing the first phase, we call a clause C_i *dangerous* if both the following conditions hold:

1. $k/2$ literals of the clause C_i have been fixed; and
2. C_i is not yet satisfied.

Phase I can be described as follows. Consider the variables $x_1, \ldots, x_\ell$ sequentially. If x_i is not in a dangerous clause, assign it independently and uniformly at random a value in $\{0, 1\}$.

A clause is a *surviving* clause if it is not satisfied by the variables fixed in phase I. Note that a surviving clause has no more than $k/2$ of its variables fixed in the first phase. A *deferred* variable is a variable that was not assigned a value in the first phase. In phase II, we use exhaustive search in order to assign values to the deferred variables and so complete a satisfying assignment for the formula.

In the next two lemmas we show that:

1. the partial solution computed in phase I can be extended to a full satisfying assignment of $\mathcal{F}$, and
2. with high probability, the exhaustive search in phase II is completed in time that is polynomial in m.

Lemma 6.14: *There is an assignment of values to the deferred variables such that all the surviving clauses are satisfied.*

Proof: Let $H = (V, E)$ be a graph on m nodes, where $V = \{1, \ldots, m\}$, and let $(i, j) \in E$ if and only if $C_i \cap C_j \neq \emptyset$. That is, H is the dependency graph for the original problem. Let $H' = (V', E')$ be a graph with $V' \subseteq V$ and $E' \subseteq E$ such that (a) $i \in V'$ if and only if C_i is a surviving clause and (b) $(i, j) \in E'$ if and only if C_i and C_j share a deferred variable. In the following discussion we do not distinguish between node i and clause i.

Consider the probability space defined by assigning a random value in $\{0, 1\}$ independently to each deferred variable. The assignment of values to the nondeferred variables in phase I, together with the random assignment of values to the deferred variables, defines an assignment to all the ℓ variables. For $i = 1, \ldots, m$, let E_i be the event that surviving clause C_i is not satisfied by this assignment. Associate the event E_i with node i in V'. The graph H' is then the dependency graph for this set of events.

A surviving clause has at least $k/2$ deferred variables, so

$$p = \Pr(E_i) \leq 2^{-k/2}.$$

A variable appears in no more than T clauses; therefore, the degree of the dependency graph is bounded by

$$d = kT \leq k2^{\alpha k}.$$

For any $k \geq 12$, there is a corresponding suitably small constant $\alpha > 0$ so that

$$4dp = 4k2^{\alpha k}2^{-k/2} \leq 1$$

and so, by the Lovász Local Lemma, there is an assignment for the deferred variables that – together with the assignment of values to variables in phase I – satisfies the formula. ■

The assignment of values to a subset of the variables in phase I partitions the problem into as many as m independent subformulas, so that each deferred variable appears in only one subformula. The subformulas are given by the connected components of H'. If we can show that each connected component in H' has size $O(\log m)$, then each

subformula will have no more than $O(k \log m)$ deferred variables. An exhaustive search of all the possible assignments for all variables in each subformula can then be done in polynomial time. Hence we focus on the following lemma.

Lemma 6.15: *All connected components in H' are of size $O(\log m)$ with probability $1 - o(1)$.*

Proof: Consider a connected component R of r vertices in H. If R is a connected component in H', then all its r nodes are surviving clauses. A surviving clause is either a dangerous clause or it shares at least one deferred variable with a dangerous clause (i.e., it has a neighbor in H' that is a dangerous clause). The probability that a given clause is dangerous is at most $2^{-k/2}$, since exactly $k/2$ of its variables were given random values in phase I yet none of these values satisfied the clause. The probability that a given clause survives is the probability that either this clause or at least one of its direct neighbors is dangerous, which is bounded by

$$(d+1)2^{-k/2},$$

where again $d = kT > 1$.

If the survival of individual clauses were independent events then we would be in excellent shape. However, from our description here it is evident that such events are not independent. Instead, we identify a subset of the vertices in R such that the survival of the clauses represented by the vertices of this subset are independent events. A *4-tree* S of a connected component R in H is defined as follows:

1. S is a rooted tree;
2. any two nodes in S are at distance at least 4 in H;
3. there can be an edge in S only between two nodes with distance exactly 4 between them in H;
4. any node of R is either in S or is at distance 3 or less from a node in S.

Considering the nodes in a 4-tree proves useful because the event that a node u in a 4-tree survives and the event that another node v in a 4-tree survives are actually independent. Any clause that could cause u to survive has distance at least 2 from any clause that could cause v to survive. Clauses at distance 2 share no variables, and hence the events that they are dangerous are independent. We can take advantage of this independence to conclude that, for any 4-tree S, the probability that the nodes in the 4-tree survive is at most

$$((d+1)2^{-k/2})^{|S|}.$$

A maximal 4-tree S of a connected component R is the 4-tree with the largest possible number of vertices. Since the degree of the dependency graph is bounded by d, there are no more than

$$d + d(d-1) + d(d-1)(d-1) \leq d^3 - 1$$

nodes at distance 3 or less from any given vertex. We therefore claim that a maximal 4-tree of R must have at least r/d^3 vertices. Otherwise, when we consider the vertices of the maximal 4-tree S and all neighbors within distance 3 or less of these vertices, we obtain fewer than r vertices. Hence there must be a vertex of distance at least 4

from all vertices in S. If this vertex has distance exactly 4 from some vertex in S, then it can be added to S and thus S is not maximal, yielding a contradiction. If its distance is larger than 4 from all vertices in S, consider any path that brings it closer to S; such a path must eventually pass through a vertex of distance at least 4 from all vertices in S and of distance 4 from some vertex in S, again contradicting the maximality of S.

To show that with probability $1 - o(1)$ there is no connected component R of size $r \geq c \log_2 m$ for some constant c in H', we show that there is no 4-tree of H of size r/d^3 that survives with probability $1 - o(1)$. Since a surviving connected component R would have a maximal 4-tree of size r/d^3, the absence of such a 4-tree implies the absence of such a component.

We need to count the number of 4-trees of size $s = r/d^3$ in H. We can choose the root of the 4-tree in m ways. A tree with root v is uniquely defined by an Eulerian tour that starts and ends at v and traverses each edge of the tree twice, once in each direction. Since an edge of S represents a path of length 4 in H, at each vertex in the 4-tree the Eulerian path can continue in as many as d^4 different ways, and therefore the number of 4-trees of size $s = r/d^3$ in H is bounded by

$$m(d^4)^{2s} = md^{8r/d^3}.$$

The probability that the nodes of each such 4-tree survive in H' is at most

$$((d+1)2^{-k/2})^s = ((d+1)2^{-k/2})^{r/d^3}.$$

Hence the probability that H' has a connected component of size r is bounded by

$$md^{8r/d^3}((d+1)2^{-k/2})^{r/d^3} \leq m2^{(rk/d^3)(8\alpha+2\alpha-1/2)} = o(1)$$

for $r \geq c \log_2 m$ and for a suitably large constant c and a sufficiently small constant $\alpha > 0$. ∎

Thus, we have the following theorem.

Theorem 6.16: *Consider a k-SAT formula with m clauses, where $k \geq 12$ is an even constant and each variable appears in up to $2^{\alpha k}$ clauses for a sufficiently small constant $\alpha > 0$. Then there is an algorithm that finds a satisfying assignment for the formula in expected time that is polynomial in m.*

Proof: As we have described, if the first phase partitions the problem into subformulas involving only $O(k \log m)$ variables, then a solution can be found by solving each subformula exhaustively in time that is polynomial in m. The probability of the first phase partitioning the problem appropriately is $1 - o(1)$, so we need only run phase I a constant number of times on average before obtaining a good partition. The theorem follows. ∎

6.9. Lovász Local Lemma: The General Case

For completeness we include the statement and proof of the general case of the Lovász Local Lemma.

Theorem 6.17: *Let $E_1, \ldots, E_n$ be a set of events in an arbitrary probability space, and let $G = (V, E)$ be the dependency graph for these events. Assume there exist $x_1, \ldots, x_n \in [0, 1]$ such that, for all $1 \leq i \leq n$,*

$$\Pr(E_i) \leq x_i \prod_{(i,j)\in E} (1 - x_j).$$

Then

$$\Pr\left(\bigcap_{i=1}^{n} \bar{E}_i\right) \geq \prod_{i=1}^{n} (1 - x_i).$$

Proof: Let $S \subseteq \{1, \ldots, n\}$. We prove by induction on $s = 0, \ldots, n$ that, if $|S| \leq s$, then for all k we have

$$\Pr\left(E_k \mid \bigcap_{j\in S} \bar{E}_j\right) \leq x_k.$$

As in the case of the symmetric version of the Local Lemma, we must be careful that the conditional probability is well-defined. This follows using the same approach as in the symmetric case, so we focus on the rest of the induction.

The base case $s = 0$ follows from the assumption that

$$\Pr(E_k) \leq x_k \prod_{(k,j)\in E} (1 - x_j) \leq x_k.$$

For the inductive step, let $S_1 = \{j \in S \mid (k, j) \in E\}$ and $S_2 = S - S_1$. If $S_2 = S$ then E_k is mutually independent of the events $\bar{E}_i$, $i \in S$, and

$$\Pr\left(E_k \mid \bigcap_{j\in S} \bar{E}_j\right) = \Pr(E_k) \leq x_k.$$

We continue with the case $|S_2| < s$. We again use the notation

$$F_S = \bigcap_{j\in S} \bar{E}_j$$

and define F_{S_1} and F_{S_2} similarly, so that $F_S = F_{S_1} \cap F_{S_2}$.

Applying the definition of conditional probability yields

$$\Pr(E_k \mid F_S) = \frac{\Pr(E_k \cap F_S)}{\Pr(F_S)}. \tag{6.6}$$

By once again applying the definition of conditional probability, the numerator of Eqn. (6.6) can be written as

$$\Pr(E_k \cap F_S) = \Pr(E_k \cap F_{S_1} \mid F_{S_2}) \Pr(F_{S_2})$$

and the denominator as

$$\Pr(F_S) = \Pr(F_{S_1} \mid F_{S_2}) \Pr(F_{S_2}).$$

Canceling the common factor then yields

$$\Pr(E_k \mid F_S) = \frac{\Pr(E_k \cap F_{S_1} \mid F_{S_2})}{\Pr(F_{S_1} \mid F_{S_2})}. \tag{6.7}$$

Since the probability of an intersection of events is bounded by the probability of each of the events and since E_k is independent of the events in S_2, we can bound the numerator of Eqn. (6.7) by

$$\Pr(E_k \cap F_{S_1} \mid F_{S_2}) \leq \Pr(E_k \mid F_{S_2}) = \Pr(E_k) \leq x_k \prod_{(k,j)\in E} (1 - x_j).$$

To bound the denominator of Eqn. (6.7), let $S_1 = \{j_1, \ldots, j_r\}$. Applying the induction hypothesis, we have

$$\begin{aligned}
\Pr(F_{S_1} \mid F_{S_2}) &= \Pr\left(\bigcap_{j\in S_1} \bar{E}_j \mid \bigcap_{j\in S_2} \bar{E}_j\right) \\
&= \prod_{i=1}^{r} \left(1 - \Pr\left(E_{j_i} \mid \left(\bigcap_{t=1}^{i-1} \bar{E}_{j_t}\right) \cap \left(\bigcap_{j\in S_2} \bar{E}_j\right)\right)\right) \\
&\geq \prod_{i=1}^{r} (1 - x_{j_i}) \\
&\geq \prod_{(k,j)\in E} (1 - x_j).
\end{aligned}$$

Using the upper bound for the numerator and the lower bound for the denominator, we can prove the induction hypothesis:

$$\begin{aligned}
\Pr\left(E_k \mid \bigcap_{j\in S} \bar{E}_j\right) &= \Pr(E_k \mid F_S) \\
&= \frac{\Pr(E_k \cap F_{S_1} \mid F_{S_2})}{\Pr(F_{S_1} \mid F_{S_2})} \\
&\leq \frac{x_k \prod_{(k,j)\in E}(1 - x_j)}{\prod_{(k,j)\in E}(1 - x_j)} \\
&= x_k.
\end{aligned}$$

The theorem now follows from:

$$\begin{aligned}
\Pr(\bar{E}_1, \ldots, \bar{E}_n) &= \prod_{i=1}^{n} \Pr(\bar{E}_i \mid \bar{E}_1, \ldots, \bar{E}_{i-1}) \\
&= \prod_{i=1}^{n} (1 - \Pr(E_i \mid \bar{E}_1, \ldots, \bar{E}_{i-1})) \\
&\geq \prod_{i=1}^{n} (1 - x_i) > 0.
\end{aligned}$$

■

6.10.* The Algorithmic Lovász Local Lemma

Recently, there have been several advances in extending the Lovász Local Lemma. We briefly summarize the key points here, and start by looking again to the k-SAT problem to provide an example of these ideas in action.

We have shown previously that if no variable in a k-SAT formula appears in more than $2^k/(4k)$ clauses, then the formula has a satisfying assignment, and we have shown that if each variable appears in no more that $2^{\alpha k}$ clauses for some constant α a solution can be found in expected polynomial time. Here we provide an improved result which again provides a solution in expected polynomial time.

Theorem 6.18: *Suppose that every clause in a k-SAT formula shares one or more variables with at most $2^{k-3} - 1$ other clauses. Then a solution for the formula exists and can be found in expected time that is polynomial in the number of clauses m.*

Before starting the proof, we informally describe our algorithm. As before, let $x_1, x_2, \ldots, x_\ell$ be the ℓ variables and $C_1, C_2, \ldots, C_m$ be the m clauses in the formula. We begin by choosing a random truth assignment (uniformly at random). We then look for a clause C_i that is unsatisfied; if no such clause exists we are done. If such a clause exists, we look specifically at the variables in the clause, and randomly choose a new truth assignment for those variables. Doing so will hopefully "fix" the clause C_i so that it is satisfied, but it may not; even worse, it may end up causing a clause C_j that shares a variable with C_i to become unsatisfied. We recursively fix these neighboring clauses, so that when the recursion is finished, we have that C_i is satisfied and we have not damaged any clause by making it become unsatisfied. We therefore have improved the situation by satisfying at least one previously unsatisfied clause. We then continue to the next unsatisfied clause; we have to do this at most m times.

The underlying question that we need to answer to show that this algorithm works is how we know that the recursion we have described stops successfully. Perhaps it simply goes on forever, or for an exponential amount of time. The proof we provide shows that this cannot be the case through a new type of argument. Specifically, we show that if such bad recursions occur with non-trivial probability, then one could compress a random string of n independent, unbiased flips into much fewer than n bits. That should seem impossible, and it is. While compression is a theme we cover in much more detail in Chapter 10, we explain the compression result we need here in the proof of the theorem. All we need is that a string of r random bits, where each bit is chosen independently and uniformly at random, cannot be compressed so that the average length of the representation over all choices of the r random bits is less than $r - 2$.

To see that this must be true, assume the best possible setting for us, where we don't have to worry about the "end" of our compressed sequence, but can use each string of bits of length less than r to represent one of the 2^r possible strings we aim to compress. That is, we won't worry that one compressed string might be "0" and another one might be "00", in which case it might be hard to distinguish whether "00" was meant to represent a single compressed string, or two copies of the string represented by "0". (Essentially, a compressed string can be terminated for free; this allowance can only

hurt us in our argument.) Still, each string of $s < r$ bits can only represent a single possible string of length r. Hence we have available one string of length 0 (the empty string), two strings of length 1, and so on. There are only $2^r - 1$ strings of length less than r; even if we count only those in computing the average length of the compressed string, which again can only hurt us, the average length would be at least

$$\sum_{i=1}^{r-1} \frac{1}{2^{r-i}} \cdot i \geq r - 2.$$

The same compression fact naturally holds true for any collection of 2^r equally likely strings; they do not have to be limited to strings of r random bits.

Given this fact, our proof proceeds as follows.

Proof of Theorem 6.18: The algorithm we use is explicitly provided as the k-Satisfiability Algorithm below.

k-Satisfiability Algorithm (Using Algorithmic LLL):

Input: A collection $C_1, C_2, \ldots, C_m$ of clauses for a k-SAT formula over n variables.

Output: A truth assignment for these variables.

Main routine:

1. Start with a random assignment.
2. While some C_i is not satisfied:
   (a) choose the unsatisfied C_i with the smallest value of i;
   (b) enter i in binary using $\lceil \log_2 m \rceil$ bits in the history;
   (c) use localcorrect on clause C_i.

localcorrect(C):

1. Resample new values for every variable in clause C.
2. While some C_j that shares a variable with C (including possibly C itself) is not satisfied
   (a) choose the unsatisfied C_j sharing a variable with C with the smallest value of j;
   (b) enter "0" followed by j in binary using $k - 3$ bits in the history;
   (c) use localcorrect on clause C_j.
3. Enter "1" in the history.

We note the algorithm produces a history, which we use in the analysis of the algorithm.

It is important to realize that while a clause can become satisfied and unsatisfied again multiple times through the recursive process, when we return to the main routine and complete the call to localcorrect, we have satisfied the clause C_i that localcorrect was called on from the main routine, and further any clause that was previously satisfied has stayed satisfied because of the recursion. What we wish to show is that the recursive process has to stop.

Our analysis makes use of the fact that our algorithm makes use of a random string. We provide two different ways to describe how our algorithm runs.

We can think of our algorithm as being described by the random string of bits it uses. It takes n bits to initially assign random truth values to each of the variables. After that, it takes k bits to resample values for a clause each time localcorrect is called. Let us refer to each time localcorrect is called as a round. Then one way to describe our algorithm's actions for j rounds is with the random string of $n + jk$ bits used by the algorithm.

But here is another way of describing how our algorithm works. We keep track of the "history" of the algorithm as shown in the algorithm. The history includes a list of the clauses that localcorrect is called on by the main routine. The history also includes a list of the recursive calls to localcorrect, in a slightly non-obvious way. First, we note that the algorithm uses a flag bit 0 and a flag bit 1 to mark the start and end of recursive calls, so the algorithm tracks the stack of recursive calls in a natural way. Second, instead of the natural approach of using $\lceil \log_2 m \rceil$ bits to represent the index of the clause in our recursive calls, the algorithm uses only $k - 3$ bits. We now explain why only $k - 3$ bits are needed. Since there are at most 2^{k-3} possible clauses that share a variable with the current clause (including the current clause itself) that could be the next one called, the clause can be represented by an index of $k - 3$ bits. (Imagine having an ordered list of the up to 2^{k-3} clauses that share a variable with each clause; we just need the index into that list.) Finally, our history will also include the current truth assignment of n bits. Note that the current truth assignment can be thought of as in an separate updatable storage area for the history; every time the truth assignment is updated, so is this part of the history.

We now show that when the algorithm has run j rounds, we can recover the random string of $n + jk$ bits that the algorithm has used from the history we have described. Start with the current truth assignment, and break the history up, using the flags that mark invocations of localcorrect. We can use the history to determine the sequence of recursive calls, and what clauses localcorrect was called on. Then, going backwards through the history, we know at each step which clause was being resampled. For that clause to have to be resampled, it must have been unsatisfied previously. But there is only *one* setting of the variables that makes a clause unsatisfied, and hence we know what the truth values for those variables were before the clause was resampled. We can therefore update the current truth assignment so that it represents the truth assignment before the resampling, and continue backwards through the process. Repeating this action, we can determine the original truth assignment, and since at each step we can determine what variable values were changed and what their values were on each resampling, we recover the whole string of $n + jk$ random bits.

Our history takes at most $n + m\lceil \log_2 m \rceil + j(k - 1)$ bits; here we use the fact that each resampling uses at most $k - 1$ bits, including the two bits that may be necessary as flags for the start and end of the recursion given by that resampling. For large enough j, our history yields a compressed form of the the random string used to run the algorithm, since only $k - 1$ bits are used to represent each resampling in the history instead of the k bits used by the algorithm.

Now suppose there were no truth assignment, in which case the algorithm would run forever. Then after a large enough number of rounds J, the history will be at most

$n + m\lceil \log_2 m \rceil + J(k-1)$ bits, while the random string running the algorithm would be $n + Jk$ bits. By our result on compressing random strings, we must have

$$n + m\lceil \log_2 m \rceil + J(k-1) \geq n + Jk - 2.$$

Hence

$$J \leq m\lceil \log_2 m \rceil + 2.$$

This contradicts that the algorithm can run forever, so there must be a truth assignment.

Similarly, the number of rounds J is more than $m\lceil \log_2 m \rceil + 2 + i$ with probability at most 2^{-i}. To see this, suppose the probability of lasting to this round is greater than 2^{-i}. Again consider the algorithm after $J = m\lceil \log_2 m \rceil + 2 + i$ rounds, so the history will be at most $n + m\lceil \log_2 m \rceil + J(k-1)$ bits. The algorithm can also be described by the $n + Jk$ random bits that led to the current state. As there are at least 2^{n+Jk} random bit strings of this length, and the probability of lasting at least this many rounds is greater than 2^{-i} by assumption, there are at least 2^{n+Jk-i} random bit strings associated with reaching this round. By our result on compressing random strings, it requires more than $n + Jk - i - 2$ bits on average to represent the at least 2^{n+Jk-i} random bit strings associated with reaching this round. But the history, as we have already argued, provides a representation of these random bit strings, in that we can reconstruct the algorithm's random bit string from the history. The number of bits the history uses is only

$$n + m\lceil \log_2 m \rceil + J(k-1) = n + Jk - i - 2,$$

a contradiction.

Since the probability of lasting more than $m\lceil \log_2 m \rceil + 2 + i$ is at most 2^{-i}, we can bound the expected number of rounds by

$$m\lceil \log_2 m \rceil + 2 + \sum_{i=1}^{\infty} i2^{-i}.$$

The expected number rounds used by the algorithm is thus at most $m\lceil \log_2 m \rceil + 4$.

The work done in each resampling round can easily be made to be polynomial in m, so the total expected time to find an assigment can be made polynomial in m as well. ■

While already surprising, the proof above can be improved slightly. A more careful encoding shows that the expected number of rounds required can be reduced to $O(m)$ instead of $O(m \log m)$. This is covered in the Exercise 6.21.

The algorithmic approach we have used for the satisfiability problem in the proof of Theorem 6.18 can be extended further to obtain an algorithmic version of the Lovász local lemma, which we now describe. Let us suppose that we have a collection of n events $E_1, E_2, \ldots, E_n$ that depend on a collection of ℓ mutually independent variables $y_1, y_2, \ldots, y_\ell$. The dependency graph on events has an edge between two events if they both depend on at least one shared variable y_i. The idea is that at each step if there is an event that is unsatisfied, we resample only the random variables on which that event depends. As with the k-Satisfiability Algorithm using the algorithmic Lovász

Local Lemma, this resampling process has to be ordered carefully to ensure progress. If the dependencies are not too great, then the right resampling algorithm terminates with a solution.

The symmetric version is easier to state.

Theorem 6.19: *Let $E_1, E_2, \ldots, E_n$ be a set of events in an arbitrary probability space that are determined by mutually independent random variables $y_1, y_2, \ldots, y_\ell$, and let $G = (V, E)$ be the dependency graph for these events. Suppose the following conditions hold for values d and p:*

1. *each event E_i is adjacent to at most d other events in the dependency graph, or equivalently, there are only d other events that also depend on one or more of the y_j that E_i depends on;*
2. $\Pr(E_i) \leq p$;
3. $ep(d+1) \leq 1$.

Then there exists an assignment of the y_i so that the event $\bigcap_{i=1}^{n} \bar{E}_i$ holds, and a resampling algorithm with the property that the expected number of times the algorithm resamples the event E_i in finding such an assignment is at most $1/d$. Hence the expected total number of resampling steps taken by the algorithm is at most n/d.

However, we also have a corresponding theorem for the asymmetric version.

Theorem 6.20: *Let $E_1, E_2, \ldots, E_n$ be a set of events in an arbitrary probability space that are determined by mutually independent random variables $y_1, y_2, \ldots, y_\ell$, and let $G = (V, E)$ be the dependency graph for these events. Assume there exist $x_1, x_2, \ldots, x_n \in [0, 1]$ such that, for all $1 \leq i \leq n$*

$$\Pr(E_i) \leq x_i \prod_{(i,j) \in E} (1 - x_j).$$

Then there exists an assignment of the y_i so that the event $\bigcap_{i=1}^{n} \bar{E}_i$ holds, and a resampling algorithm with the property that the expected number of times the algorithm resamples the event E_i in finding such an assignment is at most $x_i/(1 - x_i)$. Hence the expected total number of resampling steps taken by the algorithm is at most $\sum_{i=1}^{n} x_i/(1 - x_i)$.

The proofs of these theorems are beyond the scope of the book. Similar to the algorithm for satisfiability based on resampling given above, the proof relies on bounding the expected number of resamplings that occur over the course of the algorithm.

6.11. Exercises

Exercise 6.1: Consider an instance of SAT with m clauses, where every clause has exactly k literals.

(a) Give a Las Vegas algorithm that finds an assignment satisfying at least $m(1 - 2^{-k})$ clauses, and analyze its expected running time.

(b) Give a derandomization of the randomized algorithm using the method of conditional expectations.

Exercise 6.2:

(a) Prove that, for every integer n, there exists a coloring of the edges of the complete graph K_n by two colors so that the total number of monochromatic copies of K_4 is at most $\binom{n}{4} 2^{-5}$.
(b) Give a randomized algorithm for finding a coloring with at most $\binom{n}{4} 2^{-5}$ monochromatic copies of K_4 that runs in expected time polynomial in n.
(c) Show how to construct such a coloring deterministically in polynomial time using the method of conditional expectations.

Exercise 6.3: Given an n-vertex undirected graph $G = (V, E)$, consider the following method of generating an independent set. Given a permutation σ of the vertices, define a subset $S(\sigma)$ of the vertices as follows: for each vertex i, $i \in S(\sigma)$ if and only if no neighbor j of i precedes i in the permutation σ.

(a) Show that each $S(\sigma)$ is an independent set in G.
(b) Suggest a natural randomized algorithm to produce σ for which you can show that the expected cardinality of $S(\sigma)$ is

$$\sum_{i=1}^{n} \frac{1}{d_i + 1},$$

where d_i denotes the degree of vertex i.
(c) Prove that G has an independent set of size at least $\sum_{i=1}^{n} 1/(d_i + 1)$.

Exercise 6.4: Consider the following two-player game. The game begins with k tokens placed at the number 0 on the integer number line spanning $[0, n]$. Each round, one player, called the *chooser,* selects two disjoint and nonempty sets of tokens A and B. (The sets A and B need not cover all the remaining tokens; they only need to be disjoint.) The second player, called the *remover,* takes all the tokens from one of the sets off the board. The tokens from the other set all move up one space on the number line from their current position. The chooser wins if any token ever reaches n. The remover wins if the chooser finishes with one token that has not reached n.

(a) Give a winning strategy for the chooser when $k \geq 2^n$.
(b) Use the probabilistic method to show that there must exist a winning strategy for the remover when $k < 2^n$.
(c) Explain how to use the method of conditional expectations to derandomize the winning strategy for the remover when $k < 2^n$.

Exercise 6.5: We have shown using the probabilistic method that, if a graph G has n nodes and m edges, then there exists a partition of the n nodes into sets A and B such that at least $m/2$ edges cross the partition. Improve this result slightly: show that there exists a partition such that at least $mn/(2n - 1)$ edges cross the partition.

Exercise 6.6: We can generalize the problem of finding a large cut to finding a large k-cut. A k-cut is a partition of the vertices into k disjoint sets, and the value of a cut is the weight of all edges crossing from one of the k sets to another. In Section 6.2.1 we considered 2-cuts when all edges had the same weight 1, showing via the probabilistic method that any graph G with m edges has a cut with value at least $m/2$. Generalize this argument to show that any graph G with m edges has a k-cut with value at least $(k-1)m/k$. Show how to use derandomization (following the argument of Section 6.3) to give a deterministic algorithm for finding such a cut.

Exercise 6.7: A hypergraph H is a pair of sets (V, E), where V is the set of vertices and E is the set of hyperedges. Every hyperedge in E is a subset of V. In particular, an r-uniform hypergraph is one where the size of each edge is r. For example, a 2-uniform hypergraph is just a standard graph. A dominating set in a hypergraph H is a set of vertices $S \subset V$ such that $e \cap S \neq \emptyset$ for every edge $e \in E$. That is, S hits every edge of the hypergraph.

Let $H = (V, E)$ be an r-uniform hypergraph with n vertices and m edges. Show that there is a dominating set of size at most $np + (1-p)^r m$ for every real number $0 \leq p \leq 1$. Also, show that there is a dominating set of size at most $(m + n \ln r)/r$.

Exercise 6.8: Prove that, for every integer n, there exists a way to 2-color the edges of K_x so that there is no monochromatic clique of size k when

$$x = n - \binom{n}{k} 2^{1-\binom{k}{2}}.$$

(*Hint:* Start by 2-coloring the edges of K_n, then fix things up.)

Exercise 6.9: A *tournament* is a graph on n vertices with exactly one directed edge between each pair of vertices. If vertices represent players, then each edge can be thought of as the result of a match between the two players: the edge points to the winner. A *ranking* is an ordering of the n players from best to worst (ties are not allowed). Given the outcome of a tournament, one might wish to determine a ranking of the players. A ranking is said to *disagree* with a directed edge from y to x if y is ahead of x in the ranking (since x beat y in the tournament).

(a) Prove that, for every tournament, there exists a ranking that disagrees with at most 50% of the edges.

(b) Prove that, for sufficiently large n, there exists a tournament such that every ranking disagrees with at least 49% of the edges in the tournament.

Exercise 6.10: A family of subsets $\mathcal{F}$ of $\{1, 2, \ldots, n\}$ is called an *antichain* if there is no pair of sets A and B in $\mathcal{F}$ satisfying $A \subset B$.

(a) Give an example of $\mathcal{F}$ where $|\mathcal{F}| = \binom{n}{\lfloor n/2 \rfloor}$.

(b) Let f_k be the number of sets in $\mathcal{F}$ with size k. Show that

$$\sum_{k=0}^{n} \frac{f_k}{\binom{n}{k}} \leq 1.$$

(*Hint:* Choose a random permutation of the numbers from 1 to n, and let $X_k = 1$ if the first k numbers in your permutation yield a set in $\mathcal{F}$. If $X = \sum_{k=0}^{n} X_k$, what can you say about X?)

(c) Argue that $|\mathcal{F}| \leq \binom{n}{\lfloor n/2 \rfloor}$ for any antichain $\mathcal{F}$.

Exercise 6.11: Consider a graph in $G_{n,p}$ with n vertices and each pair of vertices independently connected by an edge with probability p. We prove a threshold for the existence of triangles in the graph.

Let $t_1, \ldots, t_{\binom{n}{3}}$ be an enumeration of all triplets of three vertices in the graph. Let $X_i = 1$ if the the three edges of the triplet t_i appear in the graph, so that t_i forms a triangle in the graph. Otherwise $X_i = 0$. Let $X = \sum_{i=1}^{\binom{n}{3}} X_i$.

(a) Compute $\mathbf{E}[X]$.

(b) Use (a) to show that if $pn \to 0$ then $\Pr(X > 0) \to 0$.

(c) Show that $\mathbf{Var}[X_i] \leq p^3$.

(d) Show that $\mathbf{Cov}(X_i, X_j) = p^5 - p^6$ for $O(n^4)$ pairs $i \neq j$, otherwise $\mathbf{Cov}(X_i, X_j) = 0$.

(e) Show that $\mathbf{Var}[X] = O(n^3 p^3 + n^4(p^5 - p^6))$.

(f) Conclude that if p is such that $pn \to \infty$ then $\Pr(X = 0) \to 0$.

Exercise 6.12: In Section 6.5.1, we bounded the variance of the number of 4-cliques in a random graph in order to demonstrate the second moment method. Show how to calculate the variance directly by using the equality from Exercise 3.9: for $X = \sum_{1=1}^{n} X_i$ the sum of Bernoulli random variables,

$$\mathbf{E}[X^2] = \sum_{i=1}^{n} \Pr(X_i = 1)\mathbf{E}[X \mid X_i = 1].$$

Exercise 6.13: Consider the problem of whether graphs in $G_{n,p}$ have cliques of constant size k. Suggest an appropriate threshold function for this property. Generalize the argument used for cliques of size 4, using either the second moment method or the conditional expectation inequality, to prove that your threshold function is correct for cliques of size 5.

Exercise 6.14: Consider a graph in $G_{n,p}$, with $p = c \ln n/n$. Use the second moment method or the conditional expectation inequality to prove that if $c < 1$ then, for any constant $\varepsilon > 0$ and for n sufficiently large, the graph has isolated vertices with probability at least $1 - \varepsilon$.

Exercise 6.15: Consider a graph in $G_{n,p}$, with $p = 1/n$. Let X be the number of triangles in the graph, where a triangle is a clique with three edges. Show that

$$\Pr(X \geq 1) \leq 1/6$$

and that

$$\lim_{n \to \infty} \Pr(X \geq 1) \geq 1/7.$$

(*Hint:* Use the conditional expectation inequality.)

Exercise 6.16: Consider the set-balancing problem of Section 4.4. We claim that there is an $n \times n$ matrix $\mathbf{A}$ for which $\|\mathbf{A}\bar{b}\|_\infty$ is $\Omega(\sqrt{n})$ for any choice of $\bar{b}$. For convenience here we assume that n is even.

(a) We have shown in Eqn. (5.5) that

$$n! \leq \mathrm{e}\sqrt{n}\left(\frac{n}{\mathrm{e}}\right)^n.$$

Using similar ideas, show that

$$n! \geq a\sqrt{n}\left(\frac{n}{\mathrm{e}}\right)^n$$

for some positive constant a.

(b) Let $b_1, b_2, \ldots, b_{m/2}$ all equal 1, and let $b_{m/2+1}, b_{m/2+2}, \ldots, b_m$ all equal -1. Let $Y_1, Y_2, \ldots, Y_m$ each be chosen independently and uniformly at random from $\{0, 1\}$. Show that there exists a positive constant c such that, for sufficiently large m,

$$\Pr\left(\left|\sum_{i=1}^{m} b_i Y_i\right| > c\sqrt{m}\right) > \frac{1}{2}.$$

(*Hint:* Condition on the number of Y_i that are equal to 1.)

(c) Let $b_1, b_2, \ldots, b_m$ each be equal to either 1 or -1. Let $Y_1, Y_2, \ldots, Y_m$ each be chosen independently and uniformly at random from $\{0, 1\}$. Show that there exists a positive constant c such that, for sufficiently large m,

$$\Pr\left(\left|\sum_{i=1}^{m} b_i Y_i\right| > c\sqrt{m}\right) > \frac{1}{2}.$$

(d) Prove that there exists a matrix $\mathbf{A}$ for which $\|\mathbf{A}\bar{b}\|_\infty$ is $\Omega(\sqrt{n})$ for any choice of $\bar{b}$.

Exercise 6.17: Use the Lovász Local Lemma to show that, if

$$4\binom{k}{2}\binom{n}{k-2}2^{1-\binom{k}{2}} \leq 1,$$

then it is possible to color the edges of K_n with two colors so that it has no monochromatic K_k subgraph.

Exercise 6.18: Use the general form of the Lovász Local Lemma to prove that the symmetric version of Theorem 6.11 can be improved by replacing the condition $4dp \leq 1$ by the weaker condition $\mathrm{e}p(d+1) \leq 1$.

Exercise 6.19: Let $G = (V, E)$ be an undirected graph and suppose each $v \in V$ is associated with a set $S(v)$ of $8r$ colors, where $r \geq 1$. Suppose, in addition, that for each $v \in V$ and $c \in S(v)$ there are at most r neighbors u of v such that c lies in $S(u)$. Prove that there is a proper coloring of G assigning to each vertex v a color from its class $S(v)$ such that, for any edge $(u, v) \in E$, the colors assigned to u and v are different.

You may want to let $A_{u,v,c}$ be the event that u and v are both colored with color c and then consider the family of such events.

Exercise 6.20: A k-uniform hypergraph is an ordered pair $G = (V, E)$, but edges consist of sets of k (distinct) vertices, instead of just 2. (So a 2-uniform hypergraph is just what we normally call a graph.) A hypergraph is k-regular if all vertices have degree k; that is, they are in k hypergraph edges.

Show that for sufficiently large k, the vertices of a k-uniform, k-regular hypergraph can be 2-colored so that no edge is monochromatic. What's the smallest value of k you can achieve?

Exercise 6.21: In our description of the k-Satisfiability Algorithm using the algorithmic Lovász local lemma, we used $\lceil \log_2 m \rceil$ bits in the history to represent each clause called in the main routine. Instead, however, we could simply record in the history which clauses are initially unsatisfied with an array of m bits. Explain any other changes you need to make in the algorithm in order to properly record a history that you can "reverse" to obtain the initial assignment, and explain how this allows one to modify the proof of Theorem 6.18 so that only $O(m)$ rounds are needed in expectation.

Exercise 6.22: Implement the algorithmic Lovász Local Lemma for the following scenario. Consider a 9-SAT formula where each variable appears in 8 clauses. Set up a formula with 112,500 variables and 100,000 clauses in the following manner: set up 8 copies of each of the 112,500 variables (900,000 total variables), permute them, and use the ordering to assign the variables to the 100,000 clauses. (If any clauses share a variable, which is likely to happen, try to locally correct for this by swapping one copy to another clause.) Then assign a random "sign" to each variable – with probability 1/2, use $\bar{x}$ instead of x. This gives a formula that satisfies the conditions of Theorem 6.18.

Your implementation of the algorithmic Lovász Local Lemma does not need to keep track of the history. However, you should track how many times the local correction procedure is required before termination. Repeat this experiment with 100 different formulas derived from the process above, and report on the distribution of the number of local corrections required. Note that you may want to take some care to make the local correction step efficient in order to have your program run effectively.