Note to readers:
Please ignore these
sidenotes; they're just
hints to myself for
preparing the index,
and they're often flaky!

KNUTH

THE ART OF COMPUTER PROGRAMMING

VOLUME 4 PRE-FASCICLE 9C

ESTIMATING BACKTRACK COSTS

(ridiculously preliminary draft)

DONALD E. KNUTH *Stanford University*

ADDISON–WESLEY

September 1, 2026

Internet
Stanford GraphBase
MMIX

Internet page `https://www-cs-faculty.stanford.edu/~knuth/taocp.html` contains current information about this book and related books.

See also `https://www-cs-faculty.stanford.edu/~knuth/sgb.html` for information about *The Stanford GraphBase*, including downloadable software for dealing with the graphs used in many of the examples in Chapter 7.

See also `https://www-cs-faculty.stanford.edu/~knuth/mmixware.html` for downloadable software to simulate the MMIX computer.

See also `https://www-cs-faculty.stanford.edu/~knuth/programs.html` for various experimental programs that I wrote while writing this material (and some data files).

Copyright © 2022 by Addison–Wesley

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher, except that the official electronic file may be used to print single copies for personal (not commercial) use.

Zeroth printing (revision -89), 31 August 2026

HIPPOCRATES
LONGFELLOW
Internet

PREFACE

Life is short, the art is long.
— HIPPOCRATES, *Aphorisms* (c. 415 B.C.)

Art is long, and Time is fleeting.
— HENRY WADSWORTH LONGFELLOW, *A Psalm of Life* (1838)

THIS BOOKLET contains draft material that I'm circulating to experts in the field, in hopes that they can help remove its most egregious errors before too many other people see it. I am also, however, posting it on the Internet for courageous and/or random readers who don't mind the risk of reading a few pages that have not yet reached a very mature state. *Beware:* This material has not yet been proofread as thoroughly as the manuscripts of Volumes 1, 2, 3, 4A, and 4B were at the time of their first printings. And alas, those carefully checked volumes were subsequently found to contain thousands of mistakes.

Given this caveat, I hope that my errors this time will not be so numerous and/or obtrusive that you will be discouraged from reading the material carefully. I did try to make the text both interesting and authoritative, as far as it goes. But the field is vast; I cannot hope to have surrounded it enough to corral it completely. So I beg you to let me know about any deficiencies that you discover.

To put the material in context, this portion of fascicle 9 previews Section 7.2.2.9 of *The Art of Computer Programming*, entitled "Estimating backtrack costs." I haven't had time to write much of it yet — not even this preface!

* * *

The explosion of research in combinatorial algorithms since the 1970s has meant that I cannot hope to be aware of all the important ideas in this field. I've tried my best to get the story right, yet I fear that in many respects I'm woefully ignorant. So I beg expert readers to steer me in appropriate directions.

Please look, for example, at the exercises that I've classed as research problems (rated with difficulty level 46 or higher), namely exercises . . . ; I've also implicitly mentioned or posed additional unsolved questions in the answers to exercises . . . . Are those problems still open? Please inform me if you know of a solution to any of these intriguing questions. And of course if no solution is known today but you do make progress on any of them in the future, I hope you'll let me know.

I urgently need your help also with respect to some exercises that I made up as I was preparing this material. I certainly don't like to receive credit for things that have already been published by others, and most of these results are quite natural "fruits" that were just waiting to be "plucked." Therefore please tell me if you know who deserves to be credited, with respect to the ideas found in exercises .... Furthermore I've credited exercises ... to unpublished work of .... Have any of those results ever appeared in print, to your knowledge?

Li
Rote
Wermuth
Knuth

* * *

Special thanks are due to Shisheng Li, Günter Rote, Udo Wermuth, ... for their detailed comments on my early attempts at exposition, as well as to numerous other correspondents who have contributed crucial corrections.

* * *

I happily offer a "finder's fee" of $2.56 for each error in this draft when it is first reported to me, whether that error be typographical, technical, or historical. The same reward holds for items that I forgot to put in the index. And valuable suggestions for improvements to the text are worth 32¢ each. (Furthermore, if you find a better solution to an exercise, I'll actually do my best to give you immortal glory, by publishing your name in the eventual book:–)

Cross references to yet-unwritten material sometimes appear as '00'; this impossible value is a placeholder for the actual numbers to be supplied later.

Happy reading!

Stanford, California D. E. K.
99 Umbruary 2020

MILTON
CARROLL
KNUTH
random sampling–
Monte Carlo–
search trees–
independent sets–
Π_8
pi graph of order 8
leaf node: a tree node that has no children

That which purifies us is triall,
and triall is by what is contrary.
— JOHN MILTON, *Areopagitica* (1644)

Does not my exhausted Reader ... feel
a certain glow of pride at having constructed so splendid a Tree—
such a veritable Monarch of the Forest?
— LEWIS CARROLL, in *Symbolic Logic, Part II* (1896)

Backtrack programs are full of surprises.
Sometimes they produce instant answers to a supposedly difficult problem.
But sometimes they spin their wheels endlessly,
trying to traverse an astronomically large search tree.
And sometimes they deliver results just about as fast as we might expect.
— DONALD E. KNUTH, while preparing
Combinatorial Algorithms, Part 2 (2015)

7.2.2.9. Estimating backtrack costs. Let's return now to a topic that was introduced near the beginning of Section 7.2.2, namely the fact that *random sampling* can often help us predict the approximate size of a search tree.

Suppose, for example, that we want to look at all of the independent sets of a given graph. Figure 400(a) shows Π_8, the pi graph of order 8, which was introduced in Section 7.2.2.5; and Fig. 400(b) shows a search tree that discovers each of its independent sets. There are 26 such sets, corresponding to the 26 leaves of that tree: $\emptyset$, $\{0\}$, $\{1\}$, $\{2\}$, $\{1,2\}$, $\{3\}$, $\{0,3\}$, ..., $\{0,4,7\}$, $\{2,4,7\}$, $\{6,7\}$.

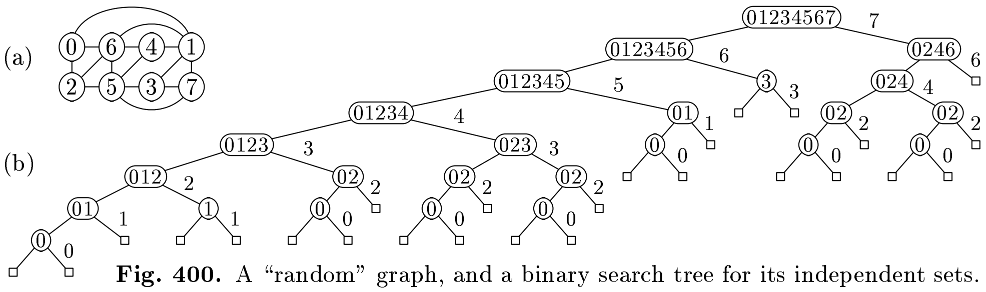


Fig. 400. A "random" graph, and a binary search tree for its independent sets.

Each node s of Fig. 400(b) is labeled with the subset S of vertices that might be in an independent set consistent with the downward path to s. More precisely, every nonleaf node is a binary branch: Its left child leads to the independent sets that do *not* include the largest candidate $v \in S$, while its right child leads to those that *do* include v. Thus, for example, the right subtree of the root represents all of the independent sets of $\Pi_8 \mid \{0,2,4,6\}$, because vertex 7 is adjacent to 1, 3, and 5. The left subtree represents all of the independent sets of Π_7.

In general, when a problem involves a potentially huge search tree T, we often want to estimate

$$\mathrm{cost}(T) \;=\; \sum_{s\in T} c(s), \tag{1}$$

the total cost of its nodes, where $c(s)$ is the cost associated with a particular node s. For example, if we want to count the total number of nodes in T, we simply define $c(s) = 1$ for each s. Or if we want to count only the number of leaves, we let $c(s) = [s \text{ is a leaf}]$. Or we might want the cost to be the amount of time that's spent when processing node s. Many useful scenarios arise, and we shall discuss methods that work for *any* function c for which (1) makes sense.

unbiased estimator
random path
degree of a tree node
$\widehat{s}$
parent
recurrence
variance–
mean

We learned long ago in Algorithm 7.2.2E about a surprisingly easy way to compute a random value X that is an "unbiased estimator" of the cost, in the sense that

$$\mathrm{E}\,X \;=\; \mathrm{cost}(T). \tag{2}$$

In fact, we obtain such an X by simply flipping coins and following a random path, starting at the root of T, until coming to a leaf. Here's that algorithm again, slightly reformulated:

Algorithm P (*Path sampling*). Given a tree T (or a way to generate T, top-down), and a cost function c on the nodes $s \in T$, this Monte Carlo algorithm computes a value X that satisfies (2).

P1. [Initialize.] Set $W \leftarrow 1$, $X \leftarrow 0$, and $s \leftarrow \mathrm{root}(T)$.

P2. [Augment X.] Set $X \leftarrow X + W \cdot c(s)$.

P3. [Done?] If s is a leaf, terminate the algorithm.

P4. [Branch.] Let s have children $\{s_1, s_2, \ldots, s_d\}$. Choose a uniformly random integer $J \in \{1, 2, \ldots, d\}$, and set $s \leftarrow s_J$, $W \leftarrow W \cdot d$. Return to step P2. ▮

The choices made in step P4 should, of course, be independent of each other.

For example, suppose we apply Algorithm P to Fig. 400(b); then it will sample one of 26 possible paths. If we define $c(s) = [s \text{ is a leaf}]$, so that $\mathrm{cost}(T)$ is the number of independent sets of Π_8, the final value of X will be the final value of W. And since that tree T is binary, the value of W will be a power of 2. Indeed, X will be 2^l when the path ends at a leaf that's l steps down from the root. So we'll get $X = 2^8 = 256$ if we happen to take the leftmost path; and we'll get $X = 2^2 = 4$ if we always branch to the right. But we're more likely to obtain $X = 16$ or $X = 32$ than either of those extreme values.

Notice that it's easy to calculate the probability $\Pr(s)$ that Algorithm P will visit any particular node s of any given tree T: Let $d(s)$ denote the degree of s (the number of s's children), and let $\widehat{s}$ be the parent of s. Then $\Pr(s) = 1/w(s)$, where

$$w(\mathrm{root}) = 1, \quad \text{and} \quad w(s) = d(\widehat{s})w(\widehat{s}) \ \text{ if } s \neq \mathrm{root}. \tag{3}$$

In fact, $w(s)$ is the value of W whenever Algorithm P visits node s. Therefore

$$\mathrm{E}\,X \;=\; \sum_{s\in T} \Pr(s)w(s)c(s) \;=\; \sum_{s\in T} \bigl(1/w(s)\bigr)w(s)c(s) \;=\; \sum_{s\in T} c(s) \;=\; \mathrm{cost}(T),$$

and we've verified (2).

Let's write $\mu = \mathrm{E}\,X$ and $\sigma^2 = \mathrm{var}(X)$ as a convenient shorthand for the mean and variance of X. If $X_1, X_2, \ldots, X_n$ are independent estimates, we know that the empirical "sample mean" $\widehat{\mathrm{E}}\,X = (X_1 + X_2 + \cdots + X_n)/n$ will almost surely be close to the true mean μ if n is large enough. Indeed, the second moment principle implies that

$$\Pr\Bigl(\widehat{\mathrm{E}}\,X \ge \mu + \frac{a\sigma}{\sqrt{n}}\Bigr) \le \frac{1}{a^2+1} \quad\text{and}\quad \Pr\Bigl(\widehat{\mathrm{E}}\,X \le \mu - \frac{a\sigma}{\sqrt{n}}\Bigr) \le \frac{1}{a^2+1}, \qquad (4)$$

for all $a \ge 0$. (See exercise MPR–48.) Our sample mean will therefore be quite reliable if the variance isn't too large.

On the other hand, a big variance can mean big trouble. Suppose, for example, that we try to estimate the number of independent sets of the *complete* graph K_n by the method above. (The answer is obviously $n+1$; but let's pretend that we need to estimate it.) Then we're applying Algorithm P to a maximally unbalanced binary tree with $n+1$ leaves: Every right child is a leaf. So X takes the value 2^l with probability $1/2^l$, for $1 \le l < n$, and the value 2^n occurs with probability $2/2^n$. In this case $\mu = n+1$; but $\sigma^2 = 2^{n+1} + 2^n - 2 - (n+1)^2$ is huge. There's clearly no practical way to distinguish between $n = 1000$ and $n = 1000000$, say, when X almost never exceeds 2^{100}.

We generally don't know whether or not the variance is suitably small. But there *is* a way to guess: After making $n > 1$ independent estimates X_1, X_2, …, X_n we can compute

$$\begin{aligned}\widehat{\mathrm{var}}(X) &= \frac{n}{n-1}\bigl(\widehat{\mathrm{E}}\,X^2 - (\widehat{\mathrm{E}}\,X)^2\bigr)\\ &= \frac{X_1^2 + X_2^2 + \cdots + X_n^2}{n-1} - \frac{(X_1 + X_2 + \cdots + X_n)^2}{n(n-1)},\end{aligned} \qquad (5)$$

which is an unbiased estimator of $\mathrm{var}(X)$. (See exercise 2.) We define $\widehat{\mathrm{var}}(X) = 0$ when $n = 1$. Floating point evaluation of $\widehat{\mathrm{var}}$ is best done with Eq. 4.2.2–(16).

Empirical estimates of the mean of a random variable X are often presented with "error bars" in the form $a \pm \delta$, where a and δ are approximations to $\widehat{\mathrm{E}}\,X$ and $\sqrt{\widehat{\mathrm{var}}(X)/n}$, suitably rounded. We might also compute the Chatterjee–Diaconis score

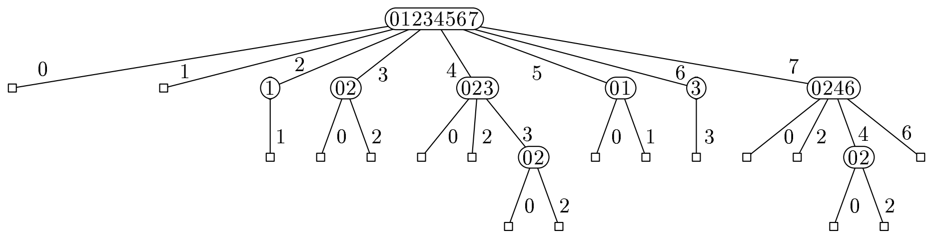
$$\widehat{\chi}(X) = \frac{\max(X_1, X_2, \ldots, X_n)}{n\,\widehat{\mathrm{E}}\,X} = \frac{\max(X_1, X_2, \ldots, X_n)}{X_1 + X_2 + \cdots + X_n}, \qquad (6)$$

which should ideally be small; see 7.2.2–(34). For example, after we apply Algorithm P to Fig. 400(b), $n = 10$ times, we typically get an error-bar estimate like 31 ± 12 for the number of leaves, and an accompanying score $\widehat{\chi} \approx 0.4$. But with $n = 1000$ we get sharper estimates such as 27 ± 1 and a score of 0.01.

The news isn't so good, however, when we try to estimate the independent sets of the larger graph Π_{64}. Here $n = 1000$ estimates typically give results like 11000 ± 2600 (and $\widehat{\chi} \approx 0.2$), which is too low: The actual number is 33607. We need to go up to $n = 1000000$ to get a decent approximation, and still the error-bars are rather large: 29000 ± 6000 (with score 0.15).

The situation is much worse with the really big graph Π_{4096}. A billion runs yield the estimate $(2.0 \pm 0.8) \times 10^{11}$ (with $\widehat{\chi} \approx 0.36$); we'll see later that this is *totally* off base. It's not even a decent ballpark estimate.

sample mean
second moment principle
complete graph
K_n
Floating point
error bars
Chatterjee–Diaconis score
pi graph



vectors
e_k
unit vector
bitwise techniques

Fig. 401. A multiway search tree for the independent sets of the graph Π_8 in Fig. 400(a).

But let's not give up hope. There's *another* way to examine all the independent sets of a graph, by using *multiway branching* instead of binary branching. For example, Fig. 401 shows the multiway search tree for Π_8; this one is more compact than Fig. 400(b): Every node s whose label lists k vertices $v_1 \ldots v_k$ now has exactly k children, corresponding to the choice of the largest vertex that will be included in the current independent set if we don't stop at node s. All independent sets of size l now appear on level l of the tree, for $l \geq 0$; therefore the relevant cost function is now $c(s) = 1$ for all s.

Algorithm P can take 17 paths in Fig. 401, and those paths are shorter than before. The smallest estimate is now 9, and the largest is now 105. Here are some typical results, together with corresponding statistics from the larger graph Π_{64}:

$$\Pi_8,\ n = 10\colon 23 \pm 3\ (\widehat{\chi} \approx 0.2) \qquad \Pi_{64},\ n = 10^3\colon 31000 \pm 5000\ (\widehat{\chi} \approx 0.13)$$
$$\Pi_8,\ n = 1000\colon 25.6 \pm 0.7\ (\widehat{\chi} \approx 0.004) \qquad \Pi_{64},\ n = 10^6\colon 33700 \pm 200\ (\widehat{\chi} \approx 0.001)$$

And a million samples with respect to Π_{4096} yield $(4.1 \pm 0.7) \times 10^{16}$ $(\widehat{\chi} \approx 0.1)$.

By looking a little closer at the tree while making these sample runs, we can actually make several estimates at once, because the individual costs $c(s)$ at each node can be *vectors* instead of scalars. (The random variable X will then be a vector too.) For example, suppose we want to estimate the number of independent sets of each size, as well as the total number. Every leaf of the binary search tree represents an independent set of size k when the path to that leaf takes k rightward branches; we can let $c(s) = e_k$ at that leaf, where e_k is the unit vector whose jth coordinate is $[j = k]$. Similarly, we can let $c(s) = e_l$ for every node s on level l of the multiway search tree. Then the expected value of X will be $(t_0, t_1, t_2, \ldots)$, when the graph has t_k independent k-element sets. And it's easy for Algorithm P to keep track of the relevant values of k or l as it moves down the tree. (For example, a million multiway samples of Π_{4096} suggest that $t_0 = 1$, $t_1 = 4096$, $t_2 \approx 4191000 \pm 2400$, $t_3 \approx 1429000000 \pm 1700000$, $\ldots$, $t_{12} \approx (1.454 \pm 2) \times 10^{14}$, and $t_{13} \stackrel{?}{=} 0$, although $\widehat{\chi} \approx 0.1$ is rather high.)

One of the great advantages of Algorithm P is that it's almost blindingly fast. Exercise 6 shows, for example, how bitwise techniques allow us to sample search trees for independent sets very quickly. Even with the large graph Π_{4096}, the running time per multiway sample is only about 550 mems, because independent sets in Π_{4096} rarely have more than 12 elements. (A binary sample takes longer, about 2700 mems.)

stratified trees–
heuristic score
stratum

Exercise 8 proves that there's a straightforward way to evaluate the variance of X *exactly*, if we're able to examine the whole tree T that is to be sampled by Algorithm P:

$$\operatorname{var}(X) = \sum_{s\in T} \sum \bigl\{ w(s)(\operatorname{cost}(s_j) - \operatorname{cost}(s_k))^2 \bigm| s_j \text{ and } s_k \text{ are children of } s \bigr\}. \quad (7)$$

Here $\operatorname{cost}(s)$ denotes the sum of $c(s')$ over all nodes s' in the subtree rooted at s. Of course, we only use Algorithm P in practice when T is too big for this calculation to actually be carried out (or when we're debugging a program to be used later with large T). But formula (7) makes it clear that the variance will be small if and only if the siblings of each family have roughly equal-cost subtrees. Using this formula we can show that the binary search trees for Π_8 and Π_{64} have variances 1440 and 262,248,355,250,929,018,767, respectively. The multiway search trees, by contrast, have variances 439 and 40,836,274,204.

Thus multiway trees are the winners in graphs like Π_n. But exercises 9 and 10 show that we get quite different outcomes in different kinds of graphs.

Stratified trees. The estimates of Algorithm P can be improved if we know more about the tree that we're trying to quantify. Indeed, there's often good reason to believe that many nodes of the tree represent subproblems that are similar to each other. We can take advantage of this knowledge when looking for a representative sample.

For instance, let's redraw Figs. 400(b) and 401 by positioning each node at a height that's based on the length of its label, rather than on its distance from the root:

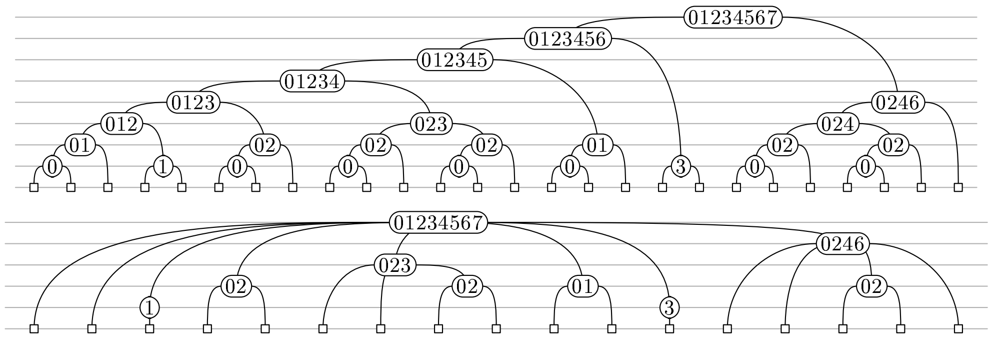


Fig. 402. Search trees with similar nodes placed into the same horizontal "stratum."

Nodes (0123) and (0246) are likely to have similar subtrees, because they both represent a search for independent sets in a 4-vertex graph. Figure 400(b), by contrast, gave equal prominence to the quite dissimilar nodes (0123456) and (0246).

In general, the idea is to assign a heuristic score $h = h(s)$, called the *stratum* of s, to every node s of the tree. This can be done in any way, provided only that

$$h(\text{root}) = 0; \quad \text{and} \quad h(s) > h(\widehat{s}) \text{ when } \widehat{s} \text{ is the parent of } s. \qquad (8)$$

But nodes of the same stratum should, ideally, have similar subtrees.

Chen
pun not resisted
Chen subset–
preorder

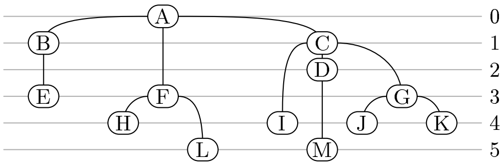


Fig. 403. A small, poorly stratified tree T. Here $h(\mathrm{A}) = 0$; $h(\mathrm{B}) = h(\mathrm{C}) = 1$; $\ldots$; $h(\mathrm{L}) = h(\mathrm{M}) = 5$.

Let's look at a somewhat weird example, in order to grasp the generality of this concept. Figure 403 illustrates a scenario in which nodes B and C have been assigned to the same stratum, although their subtrees are quite different; similarly, node E has little in common with F or G.

Given a large, stratified tree T, is there a fast way to obtain an unbiased estimate of its total cost, taking advantage of the strata when the heuristic scores $h(s)$ have been well chosen? Yes! Pang C. Chen has devised just such a "strategy," in *SICOMP* **21** (1992), 295–315. His elegant method, like Algorithm P, assigns weights $W = W(s)$ to the nodes s in a small subset S of the nodes, in such a way that the random variable $X = \sum_{s\in S} W(s)c(s)$ satisfies $\mathrm{E}\,X = \mathrm{cost}(T)$, regardless of the individual costs $c(s)$.

Every sample S produced by Chen's method will be a tree, contained in the big tree T; but it's not necessarily a path. More precisely, S will be what we shall call a *Chen subset*, defined by the following four properties:

i) The root of T is in S.
ii) If $s \in S$, then $\widehat{s} \in S$. (As above, $\widehat{s}$ denotes the parent of s.)
iii) If $s \in T$ and $\widehat{s} \in S$, then some $s' \in S$ satisfies $h(s') = h(s)$. (Perhaps $s' = s$.)
iv) If $s \in S$ and $s' \in S$ and $s \neq s'$, then $h(s) \neq h(s')$.

From (i) and (iii) and the monotonicity of h, we know that every Chen subset contains at least one leaf. And property (iv) is perhaps the most important of all: It states that *every stratum is represented by at most one node*. (Therefore, if there are fewer than 1000 strata, S will have fewer than 1000 nodes. That's what makes the method fast.)

In Fig. 403, for example, S must contain A. So it must contain either B or C, by (iii). But it cannot contain both B and C, by (iv). All told, that small tree turns out to have exactly nine Chen subsets:

$$\begin{aligned}&\mathrm{ABE, ABFHL, ACDFHL, ACDFHM, ACDFIL,}\\ &\qquad\qquad \mathrm{ACDFIM, ACDGIM, ACDGJM, ACDGKM.}\end{aligned} \tag{9}$$

At one extreme, we could assign every node of T to a distinct stratum all by itself; for example, we could let $h(s)$ be the position of s in preorder. Then every Chen subset would be the entire tree, and we'd be "estimating" $\mathrm{cost}(T)$ exactly — but very foolishly, by brute force. At another extreme, we could let $h(s)$ be the depth of s, namely its distance from the root. Then every Chen subset would be a path, and Chen's method would reduce to Algorithm P. In between those extremes there's often a happy medium, where Chen subsets provide efficient and reasonably sharp estimates.

Here then, without further ado, is Chen's method. We'll prove it valid later.

Algorithm C (*Chen subset sampling*). Given a stratified tree T (or a way to generate T, top-down), a cost function c on the nodes $s \in T$, and a heuristic function h satisfying (8) that defines the strata of T, this Monte Carlo algorithm computes a value X that satisfies (1) and (2). It uses auxiliary arrays `S` and `W`, having one entry for every possible value of $h(s)$. merge

C1. [Initialize.] Set $h \leftarrow h_{\max} \leftarrow 0$, $X \leftarrow 0$, $\mathtt{S[0]} \leftarrow \mathrm{root}(T)$, and $\mathtt{W[0]} \leftarrow 1$. Also set $\mathtt{W}[h] \leftarrow 0$ for all $h > 0$ that don't exceed every possible stratum.

C2. [Advance h.] If $\mathtt{W}[h] = 0$, set $h \leftarrow h+1$ and repeat this step.

C3. [Augment X.] Set $s \leftarrow \mathtt{S}[h]$, $W \leftarrow \mathtt{W}[h]$, and $X \leftarrow X + W \cdot c(s)$.

C4. [Done?] If s is a leaf and $h = h_{\max}$, terminate the algorithm.

C5. [Branch.] Let s have children $\{s_1, \ldots, s_d\}$. Do the operation merge(s_j, W) below, for $1 \le j \le d$. (This will ensure that $\mathtt{W}[h(s_j)] > 0$ for all j; and each of those strata $h(s_j)$ will have a representative in `S`.) Set $h \leftarrow h+1$ and return to C2. ▮

The set S of all nodes s that occur in step C3 clearly satisfies properties (i)–(iv) above; so it's a Chen subset. Conversely, we'll see below that every Chen subset has a positive probability of occurring.

The key operation needed in step C5 is quite straightforward:

$$\mathrm{merge}(s, W) \;=\; \begin{cases} \text{Set } h' \leftarrow h(s) \text{ and } W' \leftarrow \mathtt{W}[h']. \\ \text{If } h' > h_{\max}, \text{ set } h_{\max} \leftarrow h'. \\ \text{If } W' = 0, \text{ set } \mathtt{S}[h'] \leftarrow s \text{ and } \mathtt{W}[h'] \leftarrow W. \\ \text{Otherwise set } W' \leftarrow W' + W,\ \mathtt{W}[h'] \leftarrow W', \\ \quad \text{and let } U \text{ be uniform in } [0\,..\,1); \\ \quad \text{if } U < W/W', \text{ set } \mathtt{S}[h'] \leftarrow s. \end{cases} \tag{10}$$

This is where the randomization occurs. The net effect is that, after Algorithm C has seen nodes $s^{(1)}, \ldots, s^{(k)}$ on stratum h', having the respective weights $w^{(1)}, \ldots, w^{(k)}$, the current representative $\mathtt{S}[h']$ of that stratum will be $s^{(i)}$ with probability $w^{(i)}/\big(w^{(1)} + \cdots + w^{(k)}\big)$. (Exercise 14 illustrates an example run.)

How well does Algorithm C handle our independent set problem? The stratification by label size works very well indeed: The variance for the binary tree in Fig. 402 goes down from 1440 to just 2; and the variance for the multiway tree is zero! The corresponding variances for estimates of Π_{64} are also reduced dramatically, to 137,328,404 (binary) and 24,847,958 (multiway).

Of course the running time is also an issue here. When a node has high degree d, step C5 must look at all d of the children, to determine their strata and to merge them into the current sample. But step P4 looks at just one child and zooms ahead. Therefore we get a more meaningful comparison if we ask, "How reliable an estimate can I obtain by making one gigamem of computation?" — a second or so — instead of asking about the variance of a single estimate.

Consider, for example, the large graph Π_{4096}. One gigamem with Algorithm P on the binary tree brings us 372,646 independent samples; but this result, roughly $10^{10} \pm 10^{10}$, is terrible, as mentioned above. We can actually make 2,019,383 samples on the multiway tree in the same amount of time. And

oriented binary trees
binary trees
pun resisted
heat map

that result, $(3.85 \pm 0.41) \times 10^{16}$, isn't bad. Algorithm C has time to produce only five(!) samples with respect to that giant multiway tree; its estimate is then $(3.35 \pm 0.22) \times 10^{16}$, only slightly better. But Algorithm C wins with the *binary* tree: 672 samples, leading to the estimate $(3.58 \pm 0.05) \times 10^{16}$. If we allow ourselves a teramem of computation, instead of a gigamem, 671784 samples give the sharper result $(3.5882 \pm 0.0016) \times 10^{16}$. And it turns out that the true value is exactly 35856268459815803; see exercise 20.

Let's look at a different kind of tree, in order to get more experience with Algorithms P and C. There are 676,157 oriented binary trees that have 20 nodes (see exercise 2.3.4.4–6); and we can stratify them by assigning each node to stratum (l, d), where l is the distance from the root and d is the degree. (Well, Algorithm C actually wants the stratum to be an integer. So we define $h(s) = 3l + d$, plus a constant to make $h(\mathrm{root}) = 0$.)

Suppose we set $c(s) = 1$, and try to estimate the tree size. Each tree T will have a certain path variance $\mathrm{var_P}(T)$ with respect to Algorithm P, and a certain stratified variance $\mathrm{var_C}(T)$ with respect to Algorithm C. Figure 404 summarizes the joint distribution of these statistics.

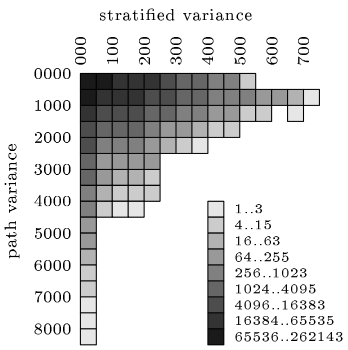
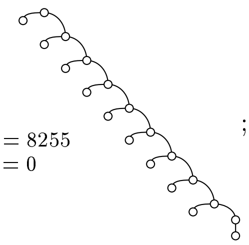
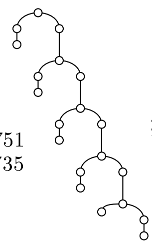
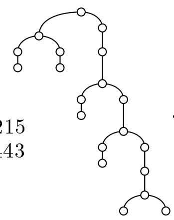


Fig. 404. A heat map for path variance versus stratified variance, based on the 676,157 oriented 20-node binary trees.

In general $\mathrm{var_P}$ tends to be roughly 10 times as large as $\mathrm{var_C}$, although there are many exceptions. Most of the trees — 416,889 of them (62%) — have $\mathrm{var_P} < 1000$ and $\mathrm{var_C} < 100$. In fact 35% have $\mathrm{var_P} < 500$ and $\mathrm{var_C} < 50$; 89% have $\mathrm{var_P} < 2000$ and $\mathrm{var_C} < 200$. But only 82 trees have $\mathrm{var_P}(T) = \mathrm{var_C}(T) = 0$.

Several extreme cases are noteworthy:

$\mathrm{var_P} = 8255$
$\mathrm{var_C} = 0$

;
$\mathrm{var_P} = 751$
$\mathrm{var_C} = 735$

;
$\mathrm{var_P} = 215$
$\mathrm{var_C} = 443$

.

The tree on the left maximizes $\mathrm{var_P}$; the tree in the middle maximizes $\mathrm{var_C}$; the curious tree on the right maximizes $\mathrm{var_C} - \mathrm{var_P}$.

crux
imitation tree
unbiased estimates

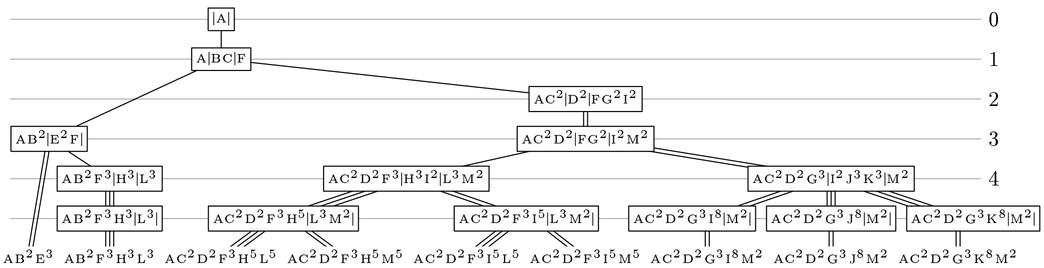


Fig. 405. Algorithm C will follow a path in this tree T', if given Fig. 403. (Every nonleaf has a label that includes a "crux" between vertical lines, as explained below.)

***Theoretical considerations.** When Algorithm C is invoked on the small example tree of Fig. 403, it essentially does a random walk in the branching tree depicted in Fig. 405. Its task is to choose a representative, unbiased Chen subset.

The top node, '|A|', means that the algorithm is poised to decide what node should represent the stratum containing A. There's only one choice; so the first move is to 'A|BC|F', which means that A has been chosen and that a choice must next be made between B and C on the next stratum. There also will come a time when a representative for the stratum containing F must be selected.

Two equally likely choices proceed from 'A|BC|F'. At the left is 'AB^2|E^{2}F|': The representative of stratum 1 is B; and we're going to pretend that A had *two* children exactly like B, each pulling a child exactly like E into the picture.

That makes a three-way tie for stratum 3, two branches of which lead to an assumed "imitation tree" whose six nodes are AB^{2}E^3. The other branch leads to 'AB^{2}F^3|H^3|L^3', because each F brings in an H and L on strata 4 and 5; and that leads inexorably to an imitation tree with twelve nodes AB^{2}F^{3}H^{3}L^3.

The right branch from 'A|BC|F' leads to similar adventures. Eventually the algorithm will choose among nine possible imitation trees, which are shown in Fig. 406. Each of them appears above the probability that it will be chosen.

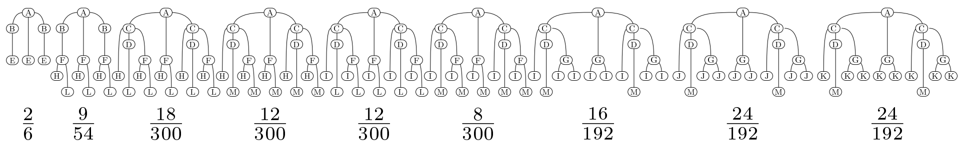


Fig. 406. The nine imitation trees that result from the paths in Fig. 405.

(The intermediate states shown in Fig. 405 don't actually match the precise behavior of Algorithm C. Instead of being in state 'AC^2|D^2|FG^{2}I^2', for instance, the algorithm will have already merged F with G^2, so that it never has to deal with more than one node from any stratum. In other words, Algorithm C makes its choices in a somewhat peculiar order. But the final outcomes are equivalent to those shown in Fig. 406.)

Figure 405 explains why Algorithm C produces unbiased estimates, because every step can be seen to preserve the expected cost. Let's write $a = c(\mathrm{A})$, $b = c(\mathrm{B})$, $\ldots$, and $A = \mathrm{cost}(\mathrm{A})$, $B = \mathrm{cost}(\mathrm{B})$, $\ldots$; then $A = a + B + C + F =$

martingale
variance
directed acyclic graph
dag

$a+(b+E)+(c+D+G+I)+(f+H+L)=\cdots=a+b+\cdots+m$. When the algorithm first reaches step C2, it's in state '|A|', with $X=0$ and A yet to be evaluated. The next time it gets to C2 it's in state 'A|BC|F', with $X=a$ and $B+C+F$ yet to be evaluated. Then it branches; we'll either have $X=a+2b$ and $2E+F$ pending, or $X=a+2c$ and $2D+F+2G+2I$ pending.

The main point is that *the expected value of X plus the pending costs hasn't changed*: We have $a+B+C+F=\frac{1}{2}(a+2b+2E+F)+\frac{1}{2}(a+2c+2D+F+2G+2I)$.

More formally, let's write for example

$$Z(\mathrm{AC}^2|\mathrm{D}^2|\mathrm{FG}^2\mathrm{I}^2)=a+2c+\mathrm{cost}(\mathrm{D}^2\mathrm{FG}^2\mathrm{I}^2)=a+2c+2D+F+2G+2I,$$

by "lowercasing" the nodes before the first vertical bar; this makes Z the sum of $a+2c$ (the estimated costs so far, namely the current value of X in this state) and $2D+F+2G+2I$ (the pending costs, which are to be estimated later).

It's not difficult to show that Z is a martingale, namely that *every random choice preserves* $\mathrm{E}\,Z$ — and by extension, that Algorithm C will produce an unbiased estimate for *any* given stratified tree. Exercise 30 illustrates the details for the 8-way split of '$|\mathrm{I}^2\mathrm{J}^3\mathrm{K}^3|$', which is the most complex branch in Fig. 405.

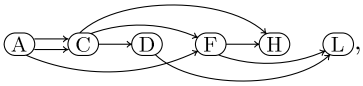
At the top, $Z=A$. At the bottom, $Z=X$ has one of nine values $a+2b+3e$, $\ldots$, $a+2c+2d+3g+8k+2m$, corresponding to the nine trees in Fig. 406, and with the probabilities shown there. The martingale property proves that

$$A=\tfrac{1}{3}(a+2b+3e)+\tfrac{1}{6}(a+2b+3f+3h+3l)+\tfrac{3}{50}(a+2c+2d+3f+5h+5l)+\\ \cdots+\tfrac{1}{8}(a+2c+2d+3g+8j+2m)+\tfrac{1}{8}(a+2c+2d+3g+8k+2m).$$

Our main goal, now that we know that X has the desired mean, is to study the *variance* of X, which is a quadratic polynomial in the individual node costs $\{a,b,\ldots,m\}$. To begin, we observe that there's a direct way to determine the final value $X(S)$ that arises in connection with any given Chen subset S of T, and its corresponding probability $p(S)$, by constructing a directed acyclic graph $\mathrm{dag}(S)$ on the nodes of S:

$$\mathrm{dag}(S)\text{ has an arc }\widehat{s}\to s'\text{ for every pair }(s,s')\text{ in property (iii).}\qquad(11)$$

In other words, there's an arc for every (s,s') with $h(s)=h(s')$ and $\widehat{s},s'\in S$. (We know that S satisfies property (iii), by the definition of Chen subsets.) For example,

dag(ACDFHL) =  , (12)

because the pairs (B, C), (C, C), (D, D), (F, F), (G, F), (H, H), (I, H), (L, L), (M, L) yield the arcs A → C, A → C, C → D, A → F, C → F, F → H, C → H, F → L, D → L. Notice that the out-degree of every node s in $\mathrm{dag}(S)$ will be the degree of s in the overall tree T.

Let $W_S(s)$ be the number of paths from the root r of T to node s in $\mathrm{dag}(S)$. In (12), for example, we have $S=\mathrm{ACDFHL}$, $W_S(\mathrm{A})=1$, $W_S(\mathrm{C})=2$, $W_S(\mathrm{D})=2$, $W_S(\mathrm{F})=3$, $W_S(\mathrm{H})=5$, and $W_S(\mathrm{L})=5$. An easy induction proves that $W_S(s)$ is, in fact, the value of W that occurs when step C3 adds $W\cdot c(s)$ to X.

imitation tree
partial Chen subset
clones
crux
critical pair

It's therefore the number of copies of node s in the imitation tree that's based on S (which for our example is the third imitation tree from the left in Fig. 406).

Furthermore, the merging mechanism of Algorithm C clearly causes S to be chosen as the Chen subset with probability

$$p_T(S) \;=\; p(S) \;=\; \prod_{s\in S\setminus r} \frac{W_S(\widehat{s})}{W_S(s)}. \tag{13}$$

(In example (12), $p(S)$ comes to $\frac{1}{2}\cdot\frac{2}{2}\cdot\frac{1}{3}\cdot\frac{3}{5}\cdot\frac{3}{5} = \frac{18}{300}$.) We've proved that, in general,

$$X(S) = \sum_{s\in S} W_S(s)\cdot c(s) \qquad \text{occurs with probability } p(S). \tag{14}$$

The variance, then, is $\sum_S p(S)X(S)^2 - \bigl(\sum_{s\in T} c(s)\bigr)^2$, where the first sum is over all Chen subtrees S in T. And here's where we get a lucky break: We've shown in Fig. 405 that the behavior of Algorithm C can actually be viewed as equivalent to the behavior of Algorithm P, on a special kind of tree T' — because Algorithm P takes random paths from the root to the leaves! Thus we can use Eq. (7) to evaluate the variance of X in Algorithm C, if we interpret (7) properly.

Let's say that a *partial Chen subset* is a subset S' that is obtained from a Chen subset S by removing all elements of strata $\geq h$, for some h. For example, the partial Chen subsets contained in the Chen subset ABFHL are $\emptyset$, A, AB, ABF, ABFH, and ABFHL. Notice now that *the tree T' in Fig. 405 is precisely the tree of partial Chen subsets of the tree in Fig. 403*, except that many of the nodes are present more than once (as indicated by parallel branch lines).

Indeed, Fig. 405 has not nine leaves but $2+9+18+12+12+8+16+24+24 = 125$ of them, because it has two copies of $\mathrm{AB^2E^3}$, 3×3 copies of $\mathrm{AB^2F^3H^3L^3}$, $2\times 3\times 3$ copies of $\mathrm{AC^2D^2F^3H^5L^5}$, and so on. For purposes of calculating the variance, it's best in fact to imagine that there are not nine Chen subsets but 125; *each S should be replicated $\prod_{s\in S\setminus r} W_S(\widehat{s})$ times.* This is the numerator of $p(S)$ in (13). Therefore, our new convention is that each of the clones of S occurs with probability $p'(S) = 1/\prod_{s\in S\setminus r} W_S(s)$ — retaining only the denominator of $p(S)$. (The leaf '$\mathrm{AB^2E^3}$', for instance, which corresponds to $X = a + 2b + 3e$, doesn't occur once with probability 1/3; it occurs twice, each with probability 1/6. Such splitting of cases with equal X does not affect the mean or variance.) And with this convention there's a perfect correspondence between the nodes of Fig. 405 and the (clones of) partial Chen subsets of Fig. 403.

Exercise 31 explains how to nail this correspondence down precisely, by showing for example that the partial Chen subset ACD has two clones, which correspond to the two nodes labeled $\mathrm{AC^2D^2|FG^2|I^2M^2}$.

The *crux* of a nonleaf label in T' is the portion between vertical lines. For example, the most complicated crux in Fig. 405 is '$\mathrm{|I^2J^3K^3|}$', on stratum 4. We say that two nodes $\{s, s'\}$ form a *critical pair* if they occur together in at least one crux. The critical pairs in Fig. 405 are therefore $\{\mathrm{B,C}\}$, $\{\mathrm{E,F}\}$, $\{\mathrm{F,G}\}$, $\{\mathrm{H,I}\}$, $\{\mathrm{I,J}\}$, $\{\mathrm{I,K}\}$, $\{\mathrm{J,K}\}$, and $\{\mathrm{L,M}\}$.

least common ancestor
lca
critical for s''
heft

Each critical pair has a *least common ancestor* (lca) in T. For example, $\mathrm{lca}(\mathrm{E},\mathrm{F}) = \mathrm{A}$; $\mathrm{lca}(\mathrm{I},\mathrm{J}) = \mathrm{C}$; $\mathrm{lca}(\mathrm{J},\mathrm{K}) = \mathrm{G}$. The critical pair $\{s, s'\}$ is said to be *critical for* s'' if $\mathrm{lca}(s, s') = s''$.

With these definitions, we're ready to state a sweeping generalization of (7):

Theorem V. *If Algorithm C computes X, given a stratified tree T, we have*

$$\mathrm{var}(X) = \sum_{s\in T} \sum \bigl\{ w(s)(\mathrm{cost}(s') - \mathrm{cost}(s''))^2 \bigm| \{s', s''\} \text{ is critical for } s \bigr\}. \qquad (15)$$

Here $w(s)$, called the *heft* of s, is defined by a recurrence that generalizes (3):

$$w(s) = \begin{cases} 1, & \text{if } s \text{ is the root;} \\ w(\widehat{s}) + \sum\{w(\mathrm{lca}(s,s')) \mid \{s,s'\} \text{ is a critical pair}\}, & \text{otherwise.} \end{cases} \qquad (16)$$

For example, $\bigl(w(\mathrm{A}), \ldots, w(\mathrm{M})\bigr) = (1, 2, 2, 2, 3, 3, 3, 4, 7, 8, 8, 4, 3)$ in Fig. 405.

Proof. Theorem V is proved in exercise 36. ▮

Corollary V. *If each individual node cost $c(s)$ is an integer, so is* $\mathrm{var}(X)$. ▮

* * *

Who knows what I might eventually say next?

* * *

EXERCISES

1. [*21*] Figure 400(b) is very similar to the ZDD for independent subsets of Π_8. Why?

▸ **2.** [*M20*] Let Σ_m denote the sum $X_1^m+\cdots+X_n^m$, where $X_1, \ldots, X_n$ are independent samples of the random variable X. Also let $\mu_m = \mathrm{E}\,X^m$. (Hence $\mathrm{var}(X) = \mu_2 - \mu_1^2$.)
 a) Prove that $\mathrm{E}\,\Sigma_1 = n\mu_1$ and $\mathrm{E}\,\Sigma_2 = n\mu_2$.
 b) Express $\mathrm{E}\,\Sigma_1^2$ in terms of n, μ_1, and μ_2.
 c) Consequently $\mathrm{E}\,\widehat{\mathrm{var}}(X) = \mathrm{var}(X)$, where $\widehat{\mathrm{var}}$ is defined for $n>1$ in Eq. (5).

3. [*HM21*] Continuing exercise 2, find an unbiased estimator for $\kappa_3 = \mathrm{E}(X-\mathrm{E}\,X)^3$, which is the third cumulant of X (see 1.2.10–(23)).

4. [*HM22*] What is the variance of $\widehat{\mathrm{var}}(X)$?

5. [*M21*] Suppose we obtain the successive estimates X_1, X_2, X_3, X_4, X_5, X_6, $\ldots$ $= 1, 2, 1, 4, 1, 2, \ldots$, where $X_k = 2^{\rho k}$ is defined by the ruler function ρ. What is the sample variance $\widehat{\mathrm{var}}(X)$ of Eq. (5) when (a) $n = 2^k - 1$? (b) $n = 2^k$?

▸ **6.** [*21*] Devise efficient data structures for using Algorithm P to estimate the number of independent sets in a given graph G, using (a) binary (b) multiway search trees.

8. [*M22*] Use the law of total variance (exercise MPR–138) to prove (7).

9. [*M22*] For each of the following graphs, does a binary or multiway search tree give a better estimate of the total number of independent sets, using Algorithm P?
 a) The complete graph K_n.
 b) The empty graph $\overline{K_n}$.
 c) The complete bipartite graph $K_{m,n}$.

10. [*M30*] Continuing exercise 9, consider the n-vertex path graph P_n.

▸ **12.** [*21*] Given a stratified tree T, formulate an XCC problem whose solutions are the Chen subsets of T.

13. [*20*] A certain tree T with eight strata has 15 nodes, seven of which have degree 2; the other eight nodes are leaves. How many of those leaves can be in a Chen subset S?

14. [*17*] Play through Algorithm C by hand with respect to the stratified tree of Fig. 403. Which of the nine Chen subsets in (9) does it implicitly construct, if the uniform deviate U in (10) is always exactly $2/5$?

15. [*16*] How many Chen subsets does the multiway tree in Fig. 402 have?

16. [*16*] Why does Algorithm C reduce to Algorithm P when $h(s) = h(\hat{s})+1$ for all s?

▸ **17.** [*21*] Let T be a search tree for the independent sets of a graph. Explain how to use vector-valued costs $c(s)$ in Algorithm C, in order to obtain unbiased estimates of the number of independent sets that have sizes $0, 1, \ldots,$ when T is (a) binary (b) multiway.

▸ **18.** [*M24*] How well does Algorithm C do, when estimating the (a) binary and (b) multiway search trees for the families of graphs in exercises 9 and 10?

19. [*M20*] How many independent sets does a *random* graph on n vertices have, if each edge is present with probability 1/2? Evaluate the expected number when $n = 4096$.

20. [*40*] Determine the exact number of independent sets of size k in the graph Π_{4096}, for $k = 0, 1, \ldots, 19$.

▸ **22.** [*24*] Use Algorithms P and C to estimate the number of independent sets in the *queen* graphs Q_n, namely the number of ways to place k nonattacking queens on an $n\times n$ chessboard, for $0\le k\le n$. Consider the counts for individual k as well as the total over all k, using both binary and multiway search trees. Approximately how many placements of nonattacking queens are possible on a 64×64 board?

ZDD
independent subsets
Π_8
unbiased estimator
cumulant
variance
ruler function
sample variance
binary
multiway
complete graph
K_n
empty graph
$\overline{K_n}$
complete bipartite graph
$K_{m,n}$
stratified tree
XCC problem
Chen subsets
vector-valued costs
random graph
queen graphs Q_n

oriented binary tree
binary tree
variance
partial Chen subset
directed acyclic graph
clones
Partial imitation trees
imitation trees
similar
heft

25. [*21*] Find an oriented binary tree on 11 vertices for which $\mathrm{var}_{\mathrm{P}}(T) = 0 < \mathrm{var}_{\mathrm{C}}(T)$.

30. [*M19*] Write out $Z(\mathrm{AC^2D^2G^3|I^2J^3K^3|M^2})$ and the Z equivalents of the three states just below that state in Fig. 405. Verify that the 8-way branch preserves $\mathrm{E}\,Z$.

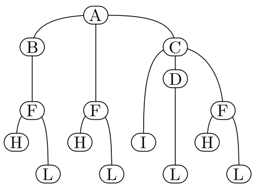
31. [*M23*] Given a partial Chen subset S' of T, define a directed acyclic graph $\mathrm{dag}(S')$ from which we can readily determine the label of the node that corresponds to S' in a tree T' such as Fig. 405. How many clones of that node appear in T'?

32. [*M21*] Nodes E and G both occur on stratum 3 of Fig. 403, but (E,G) isn't a critical pair. Why not? What is a necessary and sufficient condition that two nodes of the same stratum make a critical pair?

33. [*M20*] Why is Eq. (7) a special case of (15), and Eq. (3) a special case of (16)?

▶ **34.** [*M27*] Given the tree T' constructed in exercise 31, what costs $c'(s')$ on its nodes will make Algorithm P emulate the behavior of Algorithm C on T? And what is $\mathrm{cost}'(s')$, the sum of all costs c' in the subtree of T' rooted at s'?

▶ **35.** [*M28*] (*Partial imitation trees.*) Given a stratified tree T with root r, define a tree $I(s)$ for each $s \in T$ as follows: $I(r) = T$. Otherwise $I(s)$ is obtained from $I(\widehat{s})$ by surgically removing the subtrees of every node $s' \neq s$ with $h(s') = h(s)$, and replacing them with copies of s and *its* subtree. (These "partial imitation" trees, which encapsulate key information about T, interpolate between T and the "full imitation" trees illustrated in Fig. 406.)



a) If T is the tree in Fig. 403, its partial imitation $I(\mathrm{L})$ is shown above. Draw $I(\mathrm{I})$.

b) Prove that nodes s and t can both be present in the same Chen subset S if and only if t appears in $I(s)$.

c) What are the Chen subsets of $I(\mathrm{L})$ in the example above? With what probabilities do they arise, if Algorithm C is applied to $I(\mathrm{L})$ (instead of to T)?

d) Every node s' of $I(s)$ is a clone of some particular node $[s']$ of T. If S is a Chen subset of $I(s)$, show that the set $[S] = \{[s'] \mid s' \in S\}$ is a Chen subset of T.

e) Write $S \sim S'$, and say that S is *similar* to S', if S and S' are Chen subsets of $I(s)$ with $[S] = [S']$. In such cases $\mathrm{dag}([S])$ is essentially the same as $\mathrm{dag}(S)$ and $\mathrm{dag}(S')$. Illustrate this fact with the answer to (c).

f) Prove that, for all Chen subsets S and $s \in S$, $\sum_{S' \sim S} p_{I(s)}(S') = p_T(S) W_S(s)$. (See (13).) *Hint:* Consider Algorithm C *conditioned* on choosing s.

g) Let s and t be any nodes of T, possibly equal. Prove that $\mathrm{E}\,W_S(s)W_S(t)$, over all Chen subsets S chosen by Algorithm C, is the number of clones of t in $I(s)$.

36. [*M25*] Exercise 34 defines the costs that enter into (7) when Algorithm P emulates Algorithm C. Therefore (7) evaluates the desired variance $\mathrm{var}(X)$ in that algorithm.

a) Prove that, whenever s'_j and s'_k are children of the same parent $s' \in T'$, the difference $\mathrm{cost}'(s'_j) - \mathrm{cost}'(s'_k)$ between their subtree costs is a multiple of the difference between the costs of two nodes s_j and s_k that form a critical pair in T.

b) Consequently $\mathrm{var}(X)$ is the sum of $w(s',s'')(\mathrm{cost}(s') - \mathrm{cost}(s''))^2$ over all critical $\{s',s''\}$, for *some* rational constants $w(s',s'')$. The remaining task is to prove that $w(s',s'')$ equals the "heft," $w(\mathrm{lca}(s',s''))$, defined in (16). Apply exercise 35(g).

▶ **37.** [*22*] Design an algorithm that evaluates $\mathrm{var}(X)$, the variance in Eq. (15), given a stratified tree T. *Hint:* See exercise 31.

99. [*00*] this is a temporary dummy exercise

999. [*M00*] this is a temporary exercise (for dummies)

After [this] way of Solving Questions, a man may steale a Nappe,
and fall to worke again afresh where he left off.
— JOHN AUBREY, *An Idea of Education of Young Gentlemen* (c. 1684)

SECTION 7.2.2.9

1. The ZDD for this family of sets (with elements ordered $7 > 6 > \cdots > 0$) is obtained by (i) combining all nodes with the same label into a single node; (ii) changing each label to its largest element; and (iii) using $\boxed{\top}$ for the leaves.

2. (a) Indeed, $\mathrm{E}\,\Sigma_m = \mathrm{E}\sum_{k=1}^n X_k^m = \sum_{k=1}^n \mathrm{E}\,X_k^m = n\mu_m$.

(b) $\mathrm{E}\bigl(\sum_{k=1}^n X_k\bigr)^2 = \mathrm{E}\bigl(\sum_{k=1}^n X_k^2 + \sum_{j=1}^n \sum_{k=1}^n X_j X_k[j \neq k]\bigr) = \sum_{k=1}^n \mathrm{E}\,X^2 + \sum_{j=1}^n \sum_{k=1}^n (\mathrm{E}\,X)^2[j \neq k] = n\mu_2 + n(n-1)\mu_1^2$.

(c) $(\mathrm{E}\,\Sigma_2/n) - (\mathrm{E}\,\Sigma_1/n)^2 = \mu_2 - \mu_2/n - (n-1)\mu_1^2/n = \frac{n-1}{n}\,\mathrm{var}(X)$.

3. We have $\mathrm{E}\,\Sigma_1^3 = n\mu_3 + 3n(n-1)\mu_2\mu_1 + n(n-1)(n-2)\mu_1^3$, and $\mathrm{E}\,\Sigma_2\Sigma_1 = n\mu_3 + n(n-1)\mu_2\mu_1$. Hence $(n^2\Sigma_3 - 3n\Sigma_2\Sigma_1 + 2\Sigma_1^3)/(n(n-1)(n-2))$ has the desired mean. [R. A. Fisher, *Proceedings of the London Math. Society* (2) **30** (1929), 199–238.]

4. $\kappa_4/n + 2\kappa_2^2/(n-1)$. (See *CMath*, exercise 8.54.)

5. (a) $\bigl(2^{3k+1} - (k^2+4)2^{2k} + 2^{k+1}\bigr)/\bigl(4(2^k-1)(2^k-2)\bigr) = 2^{k-1} - k^2/4 + O(1)$; (b) $\bigl(6 \cdot 2^{3k} - (k^2+4k+6)2^{2k}\bigr)/\bigl(2^{k+2}(2^k-1)\bigr) = 3 \cdot 2^{k-1} - k^2/4 - k + O(1)$.

6. If G has vertices $\{0, 1, \ldots, n-1\}$, represent it bitwise as an $n \times n$ matrix A, with n rows A_j of $\lceil n/64 \rceil$ octabytes each. We will use only the leftmost j bits of A_j; and in that row, we will assume that $a_{jk} = [j \not\!\!-\!\!-\, k]$. (Thus A is actually the *complement* of the usual adjacency matrix.) We work with a bitwise mask $M = m_0 m_1 m_2 \ldots$, represented in $\lceil n/64 \rceil$ octabytes $M_i = m_{64i} m_{64i+1} \ldots m_{64i+63}$. And we maintain an integer z such that $M_{z-1} \neq 0$ and $M_i = 0$ for all $i \geq z$. (Or $z = 0$ if M is entirely zero.) The presence of z allows us to do most of the necessary multiword operations in a single 64-bit register. Mask M represents the labels on the nodes of Figs. 400(b) and 401; initially $m_j = [j < n]$ for all j, and $z = \lceil n/64 \rceil$, representing the root of the tree.

(a) In step P4, let v be maximum with $m_v = 1$, and set $m_v \leftarrow 0$. (Bitwise, we set $x \leftarrow M_{z-1}$, $M_{z-1} \leftarrow x \mathbin{\&} (x-1)$, and $v \leftarrow 64z - 1 - \nu((x-1) \mathbin{\&} {\sim}x)$.) Set $d \leftarrow 2$; and with probability $1/2$, also set $M \leftarrow M \mathbin{\&} A_v$. While doing this, adjust z as necessary.

(b) In step P4, set $d \leftarrow \nu M$. After choosing J, zero out the rightmost J 1-bits of M; then set $M \leftarrow M \mathbin{\&} A_v$, where m_v was the Jth 1-bit removed.

8. Let T consist of the root r and d subtrees $T_1, \ldots, T_d$ rooted at $r_1, \ldots, r_d$. Then Algorithm P computes $X = X(T) = c(r) + d \cdot X(T_J)$, where $J \in \{1, \ldots, d\}$ is uniform.

Conditioning on J we have $\mathrm{E}\bigl(X(T) \mid J = j\bigr) = \mathrm{E}\bigl(c(r) + d \cdot X(T_j)\bigr) = c(r) + d \cdot \mathrm{cost}(r_j)$. Hence $\mathrm{var}\bigl(\mathrm{E}(X \mid J)\bigr) = \sum_{j=1}^d \bigl(d \cdot \mathrm{cost}(r_j)\bigr)^2/d - \bigl(\sum_{j=1}^d d \cdot \mathrm{cost}(r_j)/d\bigr)^2 = d\sum_{j=1}^d \mathrm{cost}(r_j)^2 - \bigl(\sum_{j=1}^d \mathrm{cost}(r_j)\bigr)^2 = \sum_{j=1}^d \sum_{k=j+1}^d \bigl(\mathrm{cost}(r_j) - \mathrm{cost}(r_k)\bigr)^2$.

Also $\mathrm{var}\bigl(X(T) \mid J = j\bigr) = \mathrm{var}\bigl(d \cdot X(T_j)\bigr) = d^2\,\mathrm{var}(X(T_j))$. Hence $\mathrm{E}\bigl(\mathrm{var}(X \mid J)\bigr) = \sum_{j=1}^d d^2\,\mathrm{var}(X(T_j))/d = d\sum_{j=1}^d \mathrm{var}(X(T_j))$.

The total variance satisfies the recurrence $\mathrm{var}(X) = \mathrm{var}\bigl(\mathrm{E}(X \mid J)\bigr) + \mathrm{E}\bigl(\mathrm{var}(X \mid J)\bigr)$, which is solved by (7). [A generalization to the case that nonuniform probabilities are used in step P4 was given by D. E. Knuth in *Math. Comp.* **29** (1975), 130.]

9. (a) This is a worst-case scenario for binary search trees, as discussed in the text. But the multiway search tree has variance 0.

(b) This is a best-case scenario for binary search trees, since each of the 2^n paths yields a perfect estimate. But the multiway search trees are the binomial trees, which

AUBREY
family of sets
Fisher
bitwise
adjacency matrix
sideways addition
recurrence
Knuth
binomial trees

Fibonacci tree
recurrence
generating functions
phi
unit vectors e_k

make Algorithm P behave horribly, as shown in exercise 7.2.2–52. Indeed, (7) shows that the variance satisfies $v_n = n\sum_{k=0}^{n-1} 4^k - \bigl(\sum_{k=0}^{n-1} 2^k\bigr)^2 + n\sum_{k=0}^{n-1} v_k$; it's easy to prove that $v_k > (k+1)!$ for all $k \ge 4$.

(c) There are $2^m + 2^n - 1$ independent sets. The search trees depend on an ordering of the vertices; let's assume that the m vertices of one part precede the n vertices of the other. With binary branching there are 2^m leaves on level $m+n$ and $2^n - 1$ leaves on level n; the variance turns out to be $(2^m-1)^2(2^n-1)$. But with multiway branching we're stuck with two binomial trees, one for m and one for n, joined at the root.

10. Let the path be $1 — 2 — \cdots — n$. In the binary case the search tree turns out to be the Fibonacci tree of order $n+1$, which has F_{n+2} leaves. (See Fig. 8 in Section 6.2.1.) The variance satisfies the interesting recurrence $v_n = (F_{n+1} - F_n)^2 + 2v_{n-1} + 2v_{n-2}$, which can be solved by working with generating functions because we have $\sum_{n\ge0} v_n z^n = (z^2 - z^3)/((1+z)(1-3z+z^2)(1-2z-2z^2))$. The solution is

$$v_n = \frac{(1+\sqrt{3})^{n+1} + (1-\sqrt{3})^{n+1}}{2} - \frac{4F_{2n} + 7F_{2n+1} - 2(-1)^n}{5};$$

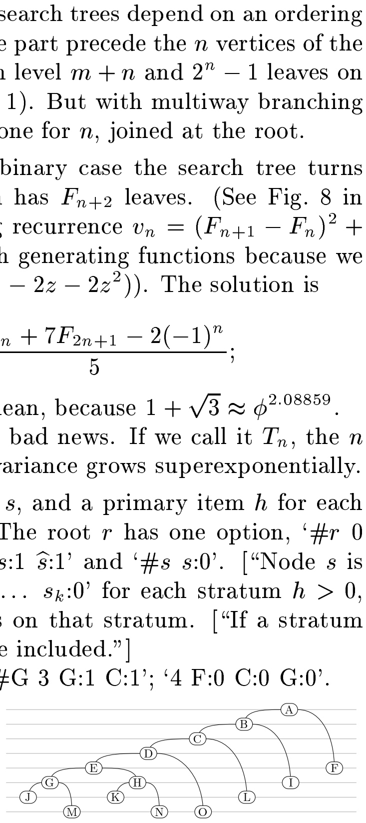
and it grows slightly faster than the square of the mean, because $1+\sqrt{3} \approx \phi^{2.08859}$.

On the other hand, the multiway search tree is bad news. If we call it T_n, the n children of the root are $(T_0, T_0, T_1, \ldots, T_{n-2})$. The variance grows superexponentially.

12. Let there be a primary item $\#s$ for each node s, and a primary item h for each stratum. Also a secondary item s for each node. The root r has one option, '$\#r$ 0 r:1'. Every node $s \ne r$ has two options, '$\#s$ $h(s)$ s:1 $\widehat{s}$:1' and '$\#s$ s:0'. ["Node s is either in or out."] Also append the option 'h s_1:0 … s_k:0' for each stratum $h > 0$, where $\{s_1, \ldots, s_k\}$ is the set of all parents of nodes on that stratum. ["If a stratum isn't represented, the parents of its nodes mustn't be included."]

For example, the options for Fig. 403 include '#G 3 G:1 C:1'; '4 F:0 C:0 G:0'.

13. There can't be five leaves in S, because at least four parents would be needed and there are only eight strata. But the tree T shown here has a four-leaf Chen subset ABCDFILO.



14. Examining the children in left-to-right order, we perform merge(B, 1), merge(F, 1), merge(C, 1) when $h = 0$; then merge(I, 2), merge(D, 2), merge(G, 2) when $h = 1$; etc. The Chen subset is ACDGJM (because $U = 2/5$ in merge(K, 3) retains `S[4]` = J). The final value of X is $c(\mathrm{A}) + 2c(\mathrm{C}) + 2c(\mathrm{D}) + 3c(\mathrm{G}) + 8c(\mathrm{J}) + 2c(\mathrm{M})$.

15. Choosing carefully on the "bottom stratum," the answer is $4 \cdot 2 \cdot 10 = 80$.

16. We'll always have $h' = h+1$ in (10). So merging $\{s_1, \ldots, s_d\}$ will leave a uniformly random s_j in `S[`$h+1$`]`, with `W[`$h+1$`]` $= d\cdot W$. (But Algorithm C does take a lot longer, of course, because each child is examined; Algorithm P peeks at only one of them.)

17. (a) As mentioned in the text, we want $c(s) = e_k$ when s is a leaf, otherwise $c(s) = (0, \ldots)$. Add a new array `K` to Algorithm C; set `K[0]` $\leftarrow 0$ in step C1, $k \leftarrow$ `K[`h`]` in step C3; and when setting `S[`h'`]` $\leftarrow s$ in (10), also set `K[`h'`]` $\leftarrow k$ if s was the left child, `K[`h'`]` $\leftarrow k+1$ if s was the right child. Ignore the value of X until termination — the final $h_{\max}$ will always be n, and there will be only one leaf. Then set $X \leftarrow W \cdot e_k$.

(b) Now $c(s) = e_l$ when we're l steps from the root. Add a new array `L` to Algorithm C; set `L[0]` $\leftarrow 0$ in step C1, $l \leftarrow$ `L[`h`]` in step C3; and when setting `S[`h'`]` $\leftarrow s$ in (10), also set `L[`h'`]` $\leftarrow l+1$. Notice that the Chen subset might contain several nodes from different strata with the same l. So X should be maintained as an array X_0, X_1,

..., initially zero, with $X_l \leftarrow X_l + W$ in step C3. (It's convenient to maintain a new variable $l_{\max}$, initially zero, the largest l seen so far in step C3.)

In practice we run Algorithm C many times, and we use Eqs. 4.2.2–(15), (16) to accumulate the overall statistics. Then it is important to realize that each run with a binary search tree estimates *zero* for all sizes $k' \neq k$; the current means and variances must be updated for *every* size less than or equal to $K_{\max}$, the largest k seen so far in different runs. Similarly, with multiway trees, we must update with estimates that are zero for all sizes with $l_{\max} < l \le L_{\max}$, where $L_{\max}$ is the largest $l_{\max}$ seen so far.

With another auxiliary array, for partial sums, we could estimate (say) the independent sets of size k whose sum is a prime number.

critical pair
heft
Fibonacci sequence
Östergård
ZDD
generating function

18. In seven of those eight cases, Algorithm C gives *perfect* results (variance 0), because any two nodes with labels of the same length have identical subtrees.

The exception is the binary tree for independent subsets of $K_{m,n}$, when $n > m+1$. If $n = m + t$ for $t > 0$, the stratum with labels of length $m + k$ has one node of cost $2^m + 2^k - 1$ and $2^{t-k} - 1$ nodes of cost 2^{m+k}, for $0 \le k < t$. Fortunately, that stratum has only one critical pair between nodes of different cost.

We must calculate the corresponding hefts using (16). Let $a_j = 1$ for $0 \le j \le m$ and $a_j = a_{j-1} + a_{j-1-m}$ for $j > m$. (This is the Fibonacci sequence when $m = 1$.) Then the variance comes to $\sum_{k=0}^{t-1} a_{t-1-k}(2^{m+k} - 2^m - 2^k + 1)^2$. (If $t \le m + 1$, all the hefts are 1, and the sum is $(2^m - 1)^2(2^{2t} - 6 \cdot 2^t + 3t + 5)/3$.)

19. The expected number of independent sets of size k is clearly $\binom{n}{k}/2^{k(k-1)/2}$. This quantity is maximized when $k \approx \lg n - \lg\ln n$. (More precisely, $k \approx \xi/\ln 2$, where $\xi e^{\xi} = n \ln 2$; see 7.2.1.5–(32).) Summing when $n = 4096$ gives $\approx 3.690196266499 \times 10^{16}$. (Of course, Π_n isn't exactly random.)

20. (Solution by P. Östergård, after 11929 "virtual core days" on a large computer cluster.) 1, 4096, 4190310, 1427544996, 182194979725, 9291979435922, 197255119600988, 1792775452108880, 7121172972180096, 12558590689855334, 9956022747349148, 3583838539300134, 590649524479701, 44880043687654, 1581582919448, 25981609725, 199845828, 722567, 1250, 0. (Multiway estimates with Algorithm P and binary estimates with Algorithm C are satisfactory only for $k \le 13$. The expected values for a *random* graph, as in exercise 19, are similar: 1, 4096, 4193280, ..., 802660.4, 1387.7, 1.1.)

22. Consider first the familiar 8×8 chessboard. Exercise 7.1.4–241(a) mentioned this family f of sets, and reported that the maximal ones are represented by a ZDD of size $Z(f^{\uparrow}) = 8577$. The entire family, with $Z(f) = 20244$, yields the generating function $t_0 + t_1 z + \cdots = 1 + 64z + 1288z^2 + 10320z^3 + 34568z^4 + 46736z^5 + 22708z^6 + 3192z^7 + 92z^8$.

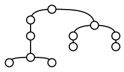
Using one gigamem of computation, Algorithm P estimates $(t_0, \ldots, t_8) = (0, 0, 0, 5700 \pm 1700, 26000 \pm 2000, 64000 \pm 21000, 24000 \pm 1000, 3170 \pm 40, 88 \pm 1)$, total 123000 ± 21000, with 6565984 samples of binary trees ($\widehat{\chi} \approx .2$); or $(1, 64, 1288, 10326 \pm 4, 34600 \pm 30, 46800 \pm 100, 22700 \pm 100, 3100 \pm 100, 110 \pm 30)$, total 119000 ± 300, with 15086912 samples of multiway trees ($\widehat{\chi} \approx 0.0003$). (The exact variances for the total, via (7), are 85112719542273883887 and 1253928499136 for binary and multiway.)

In the same amount of time, Algorithm C estimates $(1.7 \pm .4, 63 \pm 3, 1270 \pm 14, 10240 \pm 40, 34600 \pm 70, 46700 \pm 80, 22760 \pm 70, 3200 \pm 30, 93 \pm 6)$, total 118980 ± 40, with 581247 samples of binary trees ($\widehat{\chi} < 10^{-5}$); or $(1, 62 \pm 2, 1280 \pm 20, 10360 \pm 60, 34600 \pm 100, 46600 \pm 100, 22900 \pm 100, 3140 \pm 50, 88 \pm 8)$, total 119000 ± 40, with 108069 samples of multiway trees ($\widehat{\chi} \approx 2 \times 10^{-5}$). (The exact variances, by Theorem V, are 842451990 and 159706980. The exact total is 118969.)

The 64×64 board is a different story. Both versions of Algorithm P, again with a gigamem time limit, yield miserably inadequate estimates like $10^{48} \pm 10^{48}$ or $10^{38} \pm 10^{38}$, with Chatterjee–Diaconis scores exceeding .9. But Algorithm C is promising: The binary version estimates $(3.6 \pm .3) \times 10^{78}$, after generating 740 samples; the multiway version, with time to make only 2 samples, estimates $(4 \pm 2) \times 10^{78}$. A further binary run, this time allowing a teramem, suggests that the true value is $(3.79 \pm .02) \times 10^{78}$, with most of the placements having between 45 and 54 queens.

Chatterjee–Diaconis scores

25. One of two examples is shown; in both cases $\mathrm{var}_{\mathrm{C}}(T) = 8$.



30. $Z = a + 2c + 2d + 3g + 2I + 3J + 3K + 2M$; $Z_1 = a + 2c + 2d + 3g + 8i + 2M$; $Z_2 = a + 2c + 2d + 3g + 8j + 2M$; $Z_3 = a + 2c + 2d + 3g + 8k + 2M$. And $Z = (2Z_1 + 3Z_2 + 3Z_3)/8$, because $I = i$, $J = j$, $K = k$.

31. Let $\mathrm{dag}(\emptyset)$ be an isolated node, the root. Otherwise, as in (11), let $\widehat{s} \to s'$ whenever $h(s) = h(s')$ and $\widehat{s}, s' \in S'$. Also let $\widehat{s} \to s$ whenever $\widehat{s} \in S'$ but S' contains no node in stratum $h(s)$. For example, dag(ACD) is much like (12), but it lacks H and L, and has additional nodes I, G, M; arcs go $\mathrm{C} \to \mathrm{G}$, $\mathrm{C} \to \mathrm{I}$, $\mathrm{D} \to \mathrm{M}$ but not $\mathrm{C} \to \mathrm{F}$.

Put the nodes of $\mathrm{dag}(S')$, with multiplicities $W_{S'}(s)$, into a label, ordered by strata, with vertical bars surrounding the smallest unrepresented stratum. (For example, '$\mathrm{AC^2D^2|FG^2|I^2M^2}$'.) The number of clones is $\prod_{s \in S' \setminus r} W_{S'}(\widehat{s})$.

32. Nodes E and G can't be "between bars" with respect to $\mathrm{dag}(S')$ for any partial Chen subset S', because their parents (B and C) lie in the same stratum.

In general, *same-stratum nodes form a critical pair if and only if they have no two same-stratum ancestors, on the paths between them and their least common ancestor.*

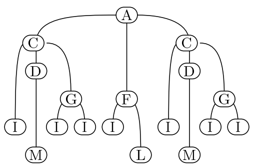
33. Two nodes on the same stratum, having the same parent, always form a critical pair for that parent. In the special case where each stratum is simply the distance from the root, those are the *only* critical pairs. And in that case if the parent $\widehat{s}$ of node s has degree d, the heft $w(s)$ is $w(\widehat{s}) + (d-1)w(\widehat{s})$, because s has $d-1$ siblings.

34. When Algorithm P reaches a node $s' \in T'$ that corresponds to the partial Chen subset S', it will have set $W \leftarrow \prod_{s \in S'} W_{S'}(s)$, which is $1/p(S')$. Let $s \in T$ be the node of S' with highest stratum. We want step P2 to set $X \leftarrow X + W_{S'}(s)c(s)$, because that's what Algorithm C would do. So we define $c'(s') = p(S')W_{S'}(s)c(s)$. [Incidentally, $p(S')W_{S'}(s) = p(S'')$, where s'' is $\widehat{s'}$, the parent of s' in T'.]

This definition of c' implies that $\mathrm{cost}'(S') = p(S')\bigl(c(s) + \sum W_{S'}(s''')\,\mathrm{cost}(s''')\bigr)$, summed over all s''' that are vertices of $\mathrm{dag}(S')$ but not elements of S'.

Of course those definitions cry out for an example. Suppose $S' = \mathrm{ACD}$. Then s' is one of the two nodes labeled $\mathrm{AC^2D^2|FG^2|I^2M^2}$. From the label we know that $s = \mathrm{D}$, $S'' = \mathrm{AC}$, $p(S') = 1/(1 \cdot 2 \cdot 2) = 1/4$, and $p(S'') = 1/(1 \cdot 2) = 1/2$. Also $\mathrm{cost}'(s') = \frac{1}{4}(\mathrm{cost(D)} + \mathrm{cost(F)} + 2\mathrm{cost(G)} + 2\mathrm{cost(I)} + 2\mathrm{cost(M)}) = \frac{1}{2}d + \frac{1}{4}(F + 2G + 2I + 2M)$, where $F = f + \mathrm{H} + \mathrm{L}$, $G = g + \mathrm{J} + \mathrm{K}$, $I = i$, $M = m$.

35. (a) (It's rather like the seventh full imitation tree in Fig. 406.)



(b) All surgically removed nodes are incompatible with s. Conversely, if $t \in I(s)$ we can do further surgery, if necessary, until obtaining a full imitation tree that includes both s and t.

(c) Using primes to distinguish same-name nodes, we find ten: ABFHL $\frac{18}{54}$; ABF′H′L′ $\frac{9}{54}$; ACDF′H′L′ $\frac{18}{300}$; ACDF′H′L″ $\frac{12}{300}$; ACDF″H″L″ $\frac{24}{300}$; ACDF″H″L‴ $\frac{36}{300}$; ACDF′IL′ $\frac{12}{300}$; ACDF′IL″ $\frac{8}{300}$; ACDF″IL″ $\frac{16}{300}$; ACDF″IL‴ $\frac{24}{300}$.

(d) Clearly $[S]$ satisfies properties (i), (ii), (iii), and (iv) of the definition.

(e) The first two Chen subsets in (c) have $[S]$ = ABFHL; the next four reduce to ACDFHL; and the last four to ACDFIL. These are the Chen subsets of T that contain L. Notice that $\mathrm{dag}(S) = \mathrm{dag}([S])$, when primes on the node names are removed, by definition of the dag; hence $W_S(s') = W_{[S]}(s')$. (Although similar Chen subsets have the same node names and the same dag, they often occur with different probabilities, because the parental structures in formula (13) differ.)

Chen
clones
left-sibling/right child links

(f) The behavior of Algorithm C on $I(s)$ is equivalent to its behavior on T, restricted to cases where s and all of its ancestors are part of the final sample. Hence the stated sum is the probability of obtaining S, divided by $1/W_S(s)$ — which is the probability of including s. (The factors for s and its ancestors in (13) are telescoping.)

(g) By (f), $\sum_S W_S(s)W_S(t)p_T(S) = \sum_{S',S} p_{I(s)}(S')[[S'] = S]W_S(t)$, which equals $\sum_{S'} p_{I(s)}(S') \sum_{s'\in S'} W_{S'}(s')[[s'] = t]$. And that's just $\sum_{S'} W_{S'}(s')p_{I(s)}(S') = 1$ times $\sum_{s'\in I(s)}[[s'] = t]$. (Surprise: By symmetry, it must also be the number of clones of s in $I(t)$(!). For example, there's one I in I(L), and one L in I(I). This theory was developed in Chen's Ph.D. thesis, *Heuristic Sampling on Backtrack Trees* (Stanford University, 1989), §3.3, where he noted that Corollary V is an immediate consequence.)

36. (a) This is almost true by definition, once we penetrate the definitions and the notations. Consider, for example, the children of node $\mathrm{AC^2D^2|FG^2|I^2M^2}$; that node corresponds to S' = ACD. The left child, corresponding to S'_1 = ACDF, has $\mathrm{cost}'(s'_1) = f/4 + (3H + 2I + 3L + 2M)/12$. The other two children, corresponding to S'_2 = ACDG, have $\mathrm{cost}'(s'_2) = g/4 + (2I + 3J + 3K + 2M)/12$. The difference is $(3f + 3H + 3L - 3g - 3J - 3K)/12 = (F - G)/4$. And (F, G) is critical.

(b) Indeed, *the heft $w(s)$ is the number of clones of s in $I(s)$*, because this quantity satisfies the recurrence (16). Consider, for example, s = I in exercise 35(a); then $\widehat{s}$ = C. There are two I's in I(C); also, on the same stratum, one H, two J's, and two K's. That's because lca(I, H) = A has heft 1, and lca(I, J) = lca(I, K) = C has heft 2.

(Notice furthermore that the coefficient of $c(s)^2$ in $\mathrm{var}(X)$ is $w(s) - 1$. If $s \neq t$, the coefficient of $c(s)c(t)$ is twice the number of t's in $I(s)$, minus 2.)

37. Let T have nodes $\{0, 1, \ldots, n-1\}$, with root 0, and with link fields `LCHILD`(p), `RSIB`(p) in each node, pointing to the leftmost child and right sibling. This algorithm also uses additional fields `PARENT`, `LINK`, `COST`, and `HEFT`. We assume that `LCHILD`$(p) > p$ and `RSIB`$(p) > p$, unless they're Λ. Furthermore there's an auxiliary array with entries `HEAD`$[h]$ for $0 \le h \le h_{\max}$, where $h_{\max}$ is the maximum stratum; it's initially zero.

V1. [Compute subtree costs.] For $p \leftarrow n-1, \ldots, 1, 0$, in decreasing order, do this: Set `COST`$(p) \leftarrow c(p)$, $q \leftarrow$ `LCHILD`(p); while $q \neq \Lambda$, set `COST`$(p) \leftarrow$ `COST`(p)+`COST`(q), `PARENT`$(q) \leftarrow p$, and $q \leftarrow$ `RSIB`(q).

V2. [Link same-stratum nodes.] For $p \leftarrow n-1, \ldots, 1, 0$ do this: Set $h \leftarrow h(p)$, `LINK`$(p) \leftarrow$ `HEAD`$[h]$, `HEAD`$[h] \leftarrow p$.

V3. [Loop on h.] Set `HEFT(0)` $\leftarrow 1$, $h \leftarrow 1$, $V \leftarrow 0$.

V4. [Initialize hefts for h.] Set $p \leftarrow$ `HEAD`$[h]$, and while $p \neq \Lambda$ set `HEFT`$(p) \leftarrow$ `HEFT(PARENT`(p)`)`, $p \leftarrow$ `LINK`(p).

V5. [Loop on p in h.] Set $p \leftarrow$ `HEAD`$[h]$. Go to V11 if $p = \Lambda$.

V6. [Loop on $q > p$ in h.] Set $q \leftarrow$ `LINK`(p). Go to V10 if $q = \Lambda$.

V7. [Test criticality.] Set $p' \leftarrow$ `PARENT`(p), $q' \leftarrow$ `PARENT`(q). While $h(p') \neq h(q')$, set $p' \leftarrow$ `PARENT`(p') if $h(p') > h(q')$, otherwise set $q' \leftarrow$ `PARENT`(q'). Then go to V9 if $p' \neq q'$. (In that case (p, q) isn't critical, by exercise 32.)

V8. [Augment V and hefts.] Set $V \leftarrow V + \mathtt{HEFT}(p') \cdot \bigl(\mathtt{COST}(p) - \mathtt{COST}(q)\bigr)^2$, $\mathtt{HEFT}(p) \leftarrow \mathtt{HEFT}(p) + \mathtt{HEFT}(p')$, $\mathtt{HEFT}(q) \leftarrow \mathtt{HEFT}(q) + \mathtt{HEFT}(p')$. (See Eqs. (15) and (16).)

V9. [End loop q.] Set $q \leftarrow \mathtt{LINK}(q)$. Return to V7 if $q \neq \Lambda$.

V10. [End loop p.] Set $p \leftarrow \mathtt{LINK}(p)$. Return to V6 if $p \neq \Lambda$.

V11. [End loop h.] Set $h \leftarrow h + 1$. Return to V4 if $h \leq h_{\max}$; otherwise terminate (the variance is V). ▌

99. ...

999. ...

INDEX AND GLOSSARY

I like indices, and the index to the NKS book — with its 14,967 entries — is my all-time favorite.

— STEPHEN WOLFRAM, The Making of *A New Kind of Science* (13 May 2022)

When an index entry refers to a page containing a relevant exercise, see also the *answer* to that exercise for further information. An answer page is not indexed here unless it refers to a topic not included in the statement of the exercise.

Preliminary notes for indexing appear in the upper right corner of most pages.

If I've mentioned somebody's name and forgotten to make such an index note, it's an error (worth \$2.56).