Stratified Importance Sampling

Don Knuth, 31 January 2020

$$
\begin{aligned}
&\text{23 Feb 1975} \quad \Longrightarrow \quad \text{Lehmer} = 70 \\
&\text{10 Jan 2018} \quad \Longrightarrow \quad \text{Knuth} = 80 \\
&\text{31 Jan 2020} \quad \Longrightarrow \quad \text{Diaconis} = 75
\end{aligned}
$$

Mathematics of Computation

Volume 29 · Number 129

Special Issue ~ Dedicated to

DERRICK HENRY LEHMER

January 1975

Published by
American Mathematical Society
Providence · Rhode Island

Derrick Henry Lehmer

THE EDITORS of Mathematics of Computation are pleased to dedicate this issue to Derrick Henry Lehmer on the occasion of his seventieth birthday, February 23, 1975.

Lehmer played a role in the founding of MTAC and was an active member of the executive committee that produced the first issue in January, 1943, under the editorship of R. C. Archibald. In the 1946–1949 volumes, Lehmer joined Archibald as an editor, while in the 1950–1954 volumes, Lehmer was the chairman of an enlarged group of editors.

In an effort to keep this dedication a surprise and to keep the issue modest in size, the editors did not inform all of the people who would have liked to contribute articles to celebrate Lehmer's birthday. In fact, the editors have postponed the publication of a paper jointly authored by the dedicatee. To those who do not appear here, we apologize.

We must make mention of a special one, who was not informed about this issue—Emma Lehmer. She has also been associated with this journal from its beginning and continues to assist us to this day. We are in her debt too!

The editors are also pleased to acknowledge that this issue was made possible through the generous support that was received from friends of the Lehmers.

Eugene Isaacson
for the editors

MATHEMATICS OF COMPUTATION, VOLUME 29, NUMBER 129
JANUARY 1975, PAGES 121–136

Estimating the Efficiency of Backtrack Programs*

By Donald E. Knuth

To Derrick H. Lehmer on his 70th birthday, February 23, 1975

Abstract. One of the chief difficulties associated with the so-called backtracking technique for combinatorial problems has been our inability to predict the efficiency of a given algorithm, or to compare the efficiencies of different approaches, without actually writing and running the programs. This paper presents a simple method which produces reasonable estimates for most applications, requiring only a modest amount of hand calculation. The method should prove to be of considerable utility in connection with D. H. Lehmer's branch-and-bound approach to combinatorial optimization.

The majority of all combinatorial computing applications can apparently be handled only by what amounts to an exhaustive search through all possibilities. Such searches can readily be performed by using a well-known "depth-first" procedure which R. J. Walker [21] has aptly called *backtracking*. (See Lehmer [16], Golomb and Baumert [6], and Wells [22] for general discussions of this technique, together with numerous interesting examples.)

Sometimes a backtrack program will run to completion in less than a second, while other applications seem to go on forever. The author once waited all night for the output from such a program, only to discover that the answers would not be forthcoming for about 10^6 centuries. A "slight increase" in one of the parameters of a backtrack routine might slow down the total running time by a factor of a thousand; conversely, a "minor improvement" to the algorithm might cause a hundredfold improvement in speed; and a sophisticated "major improvement" might actually make the program ten times slower. These great discrepancies in execution time are characteristic of backtrack programs, yet it is usually not obvious what will happen until the algorithm has been coded and run on a machine.

Faced with these uncertainties, the author worked out a simple estimation procedure in 1962, designed to predict backtrack behavior in any given situation. This procedure was mentioned briefly in a survey article a few years later [8]; and during subsequent years, extensive computer experimentation has confirmed its utility. Several move solutions of Figure 5, stating that “il est probablement impossible de dénombrer la quantité de ces tours; . . . vraisemblablement, on ne peut effectuer plus de 35 coups.” Later [3, p. 20], [4, p. 35] he stated without proof that 35 is maximum.

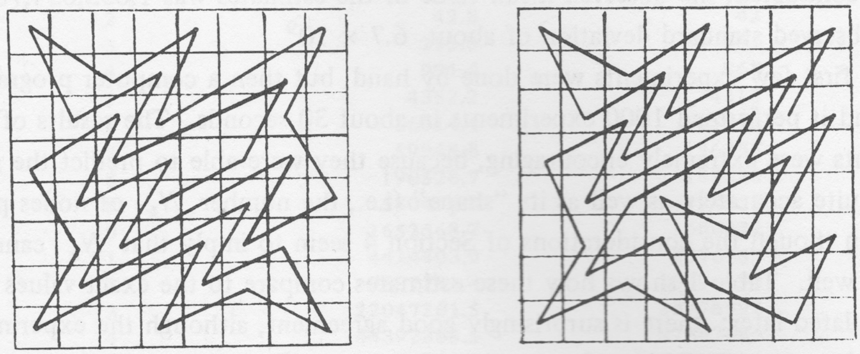


FIGURE 5. *Uncrossed knight’s tours*

The backtrack method provides a way to test his assertion; we may begin the tour in any of 10 essentially different squares, then continue by making knight’s moves that do not cross previous ones, until reaching an impasse. But backtrack trees that extend across 30 levels or more can be extremely large; even if we assume an average of only 3 consistent choices at every stage, out of at most 7 possible knight moves to new squares, we are faced with a tree of about $3^{30} = 205{,}891{,}132{,}094{,}649$ nodes, and we would never finish. Actually $3^{20} = 3{,}486{,}784{,}401$ is nearer the upper limit of feasibility, since it is not at all simple to test whether or not one move crosses another. It is certainly not clear *a priori* that an exhaustive backtrack search is economically feasible.

The simple procedure of Section 3 was therefore used to estimate the number of nodes in the tree, using $c(t) = 1$ for all t. Here are the estimated tree sizes found in the first ten independent experiments:

1571717091	209749511
315291281	58736818301
8231	311
1793651	259271
59761491	6071489081

The mean value is 6,696,688,822. The next sequence of ten experiments gave the estimates

567911	238413491
111	6697691
569585831	5848873631
111	161
411	140296511

for an average of only 680,443,586, although the four extremely low estimates make

TABLE 1. *Estimates after* 1000 *random walks*

k	Estimate, N'_k	True value, N_k
0	1.0	1
1	10.0	10
2	42.8	42
3	255.0	251
4	991.4	968
5	4352.2	4215
6	16014.4	15646
7	59948.8	56435
8	190528.7	182520
9	580450.8	574555
10	1652568.7	1606422
11	4424403.9	4376153
12	9897781.4	10396490
13	22047261.5	23978392
14	44392865.5	47667686
15	92464977.5	91377173
16	145815116.2	150084206
17	238608697.6	235901901
18	253061952.9	315123658
19	355460520.9	399772215
20	348542887.6	427209856
21	328849873.9	429189112
22	340682204.1	358868304
23	429508177.9	278831518
24	318416025.6	177916192
25	38610432.0	103894319
26	75769344.0	49302574
27	74317824.0	21049968
28	0.0	7153880
29	0.0	2129212
30	0.0	522186
31	0.0	109254
32	0.0	18862
33	0.0	2710
34	0.0	346
35	0.0	50
36	0.0	8
Total	3123375511.1	3137317290

7.2.2.9

Persi — Please excuse me for not having had time to study this closely for several months. I have a few comments to contribute to your "ongoing conversation," if you do intend to keep the topic alive. I hope that my lack of knowledge hasn't thrown me entirely off the track. But in any case these remarks are those of a motivated reader, so it is not impossible that other readers will share my misconceptions — don k

Sequential importance sampling for estimating the number of perfect matchings in bipartite graphs: An ongoing conversation with Laci

PERSI DIACONIS*

Departments of Mathematics and Statistics

Stanford University

his name is Bregman, not Brégman [in Cyrillic it's э not е]

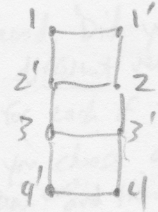
Abstract

Sequential importance sampling offers an alternative way to approximately evaluate the permanent. It is a stochastic algorithm which seems to work in practice but has eluded analysis. This paper offers examples where the analysis can be carried out and the first general bounds for the sample size required. This uses a novel importance sampling proof of Brégman's inequality due to Lovász.

1 Introduction

Let $G = (V, W, E)$ be a bipartite graph with $|V| = |W| = n$ and E a set of edges from V to W. Let $\mathcal{M}$ be the set of perfect matchings. Assume throughout that $\mathcal{M}$ is non-empty. There is a large literature on computing and approximating $M = |\mathcal{M}|$. See [4] for background and applications in statistics. The magisterial [12] covers every aspect of matching theory. This paper studies an importance sampling algorithm for Monte Carlo approximation of M.

is isomorphic to $P_2 \square P_4$



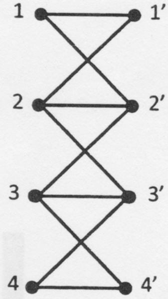


Figure 1: Fibonacci matchings

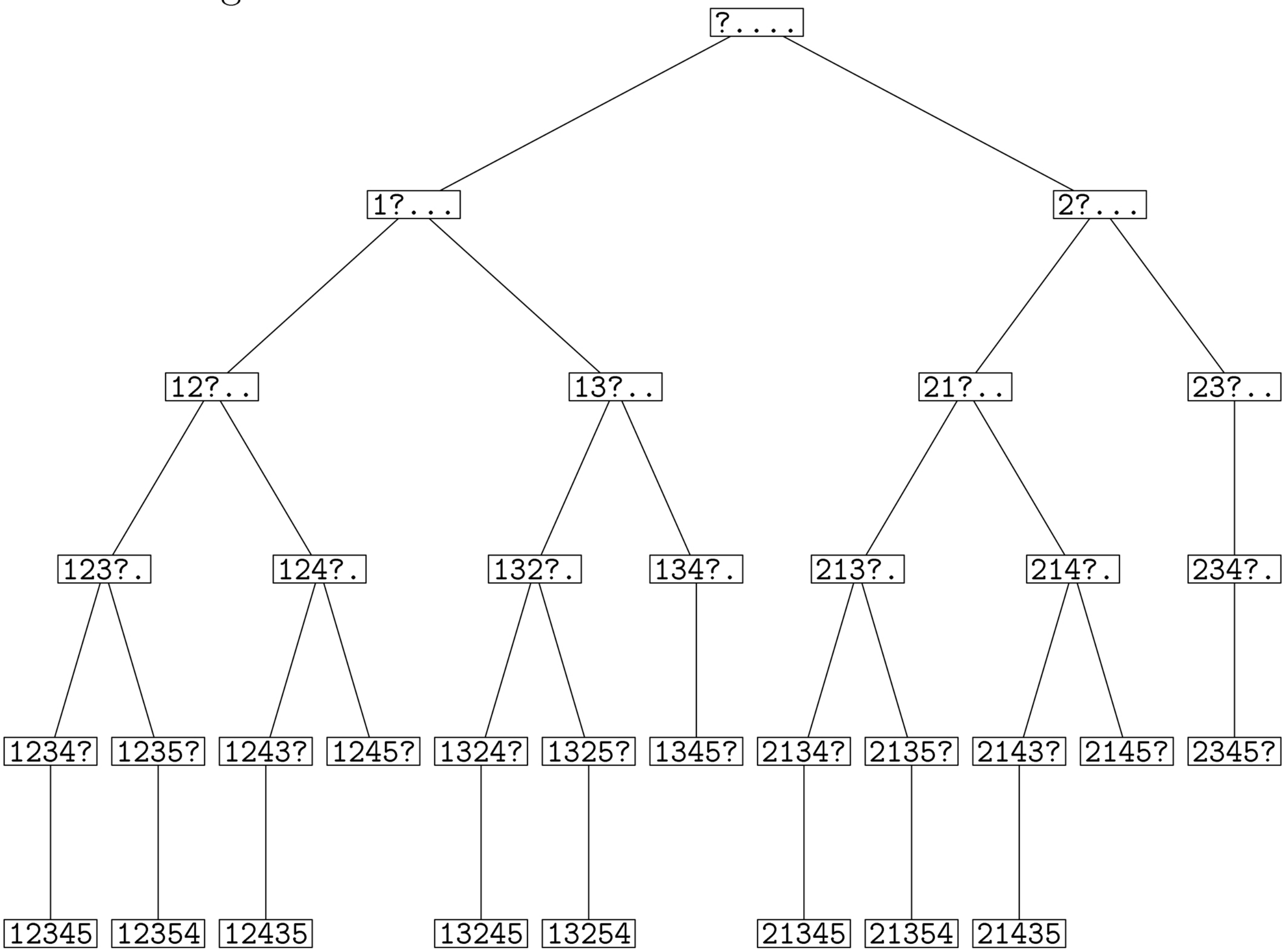
The number of perfect matchings is the Fibonacci number F_{n+1}. Indeed, 1 can only be matched to 1' or 2'. If it is matched to 1' the deleted graph is A_{n-1}. If it is matched to 2', then 2 must be matched to 1' and the deleted graph is A_{n-2}. These matchings are illustrated in Figure 1. Thus there are five perfect matchings consistent with A_4; these are shown in Table 1 along with their associated probabilities if the vertices are tried in order 1, 2, 3, 4 for $P_1(\pi)$ and 2, 3, 4, 1 for $P_2(\pi)$. For larger n the possible matching probabilities can be quite different. In what follows the vertex order $1, 2, \ldots, n$ is studied. By the way the permutation $P_3 = (1, 3, 2, 4)$ is also of interest

Table 1: Perfect matchings

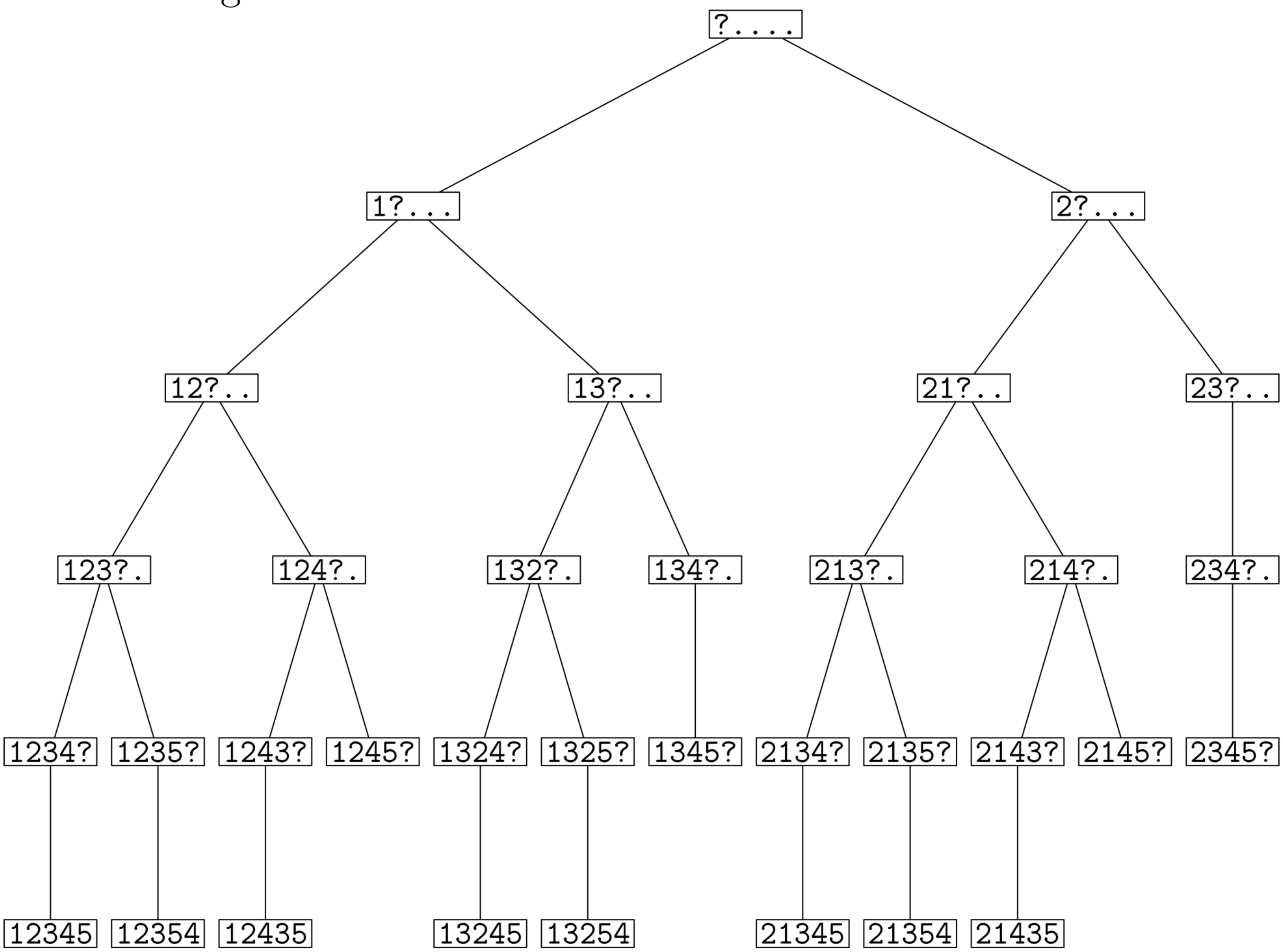
π	1234	2134	1324	1243	2143	
$P_1(\pi)$	1/8	1/4	1/4	1/8	1/4	variance = 3
$P_2(\pi)$	1/6	1/6	1/3	1/6	1/6	variance = 2
$P_3(\pi)$	1/6	1/4	1/6	1/6	1/4	variance = 1

Figure 2 shows a histogram of 1000 T_i when $n = 12$; then $F_{n+1} = 89$. The mean of 86.72 is reasonable but the minimum of 24, maximum of 288, and standard deviation of 45.3 give an indication of large variability. No, this must be $n = 10$. ($F_{13} = 233$, $F_{11} = 89$)

Left to right:



Left to right:

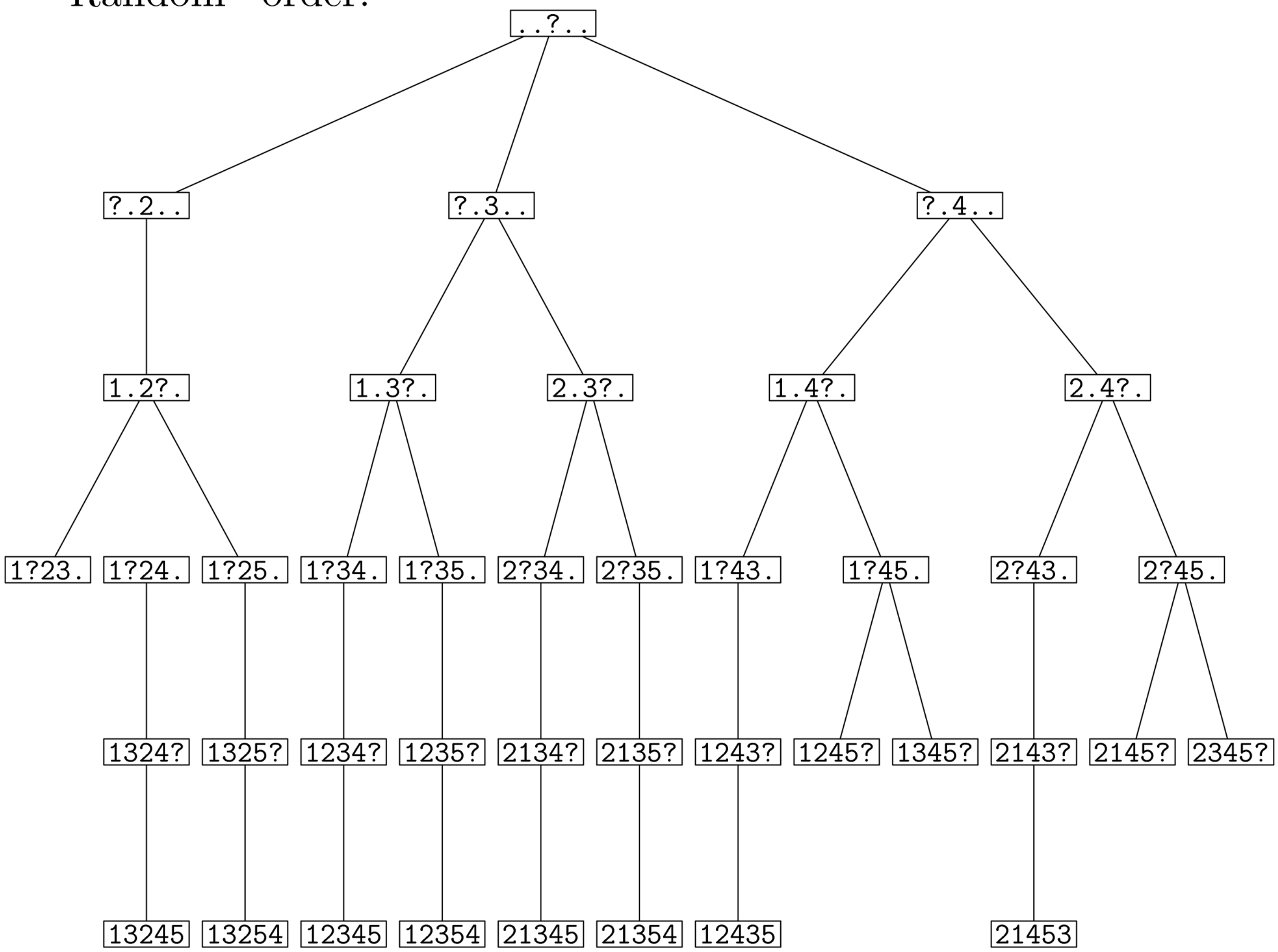


Easy to show:

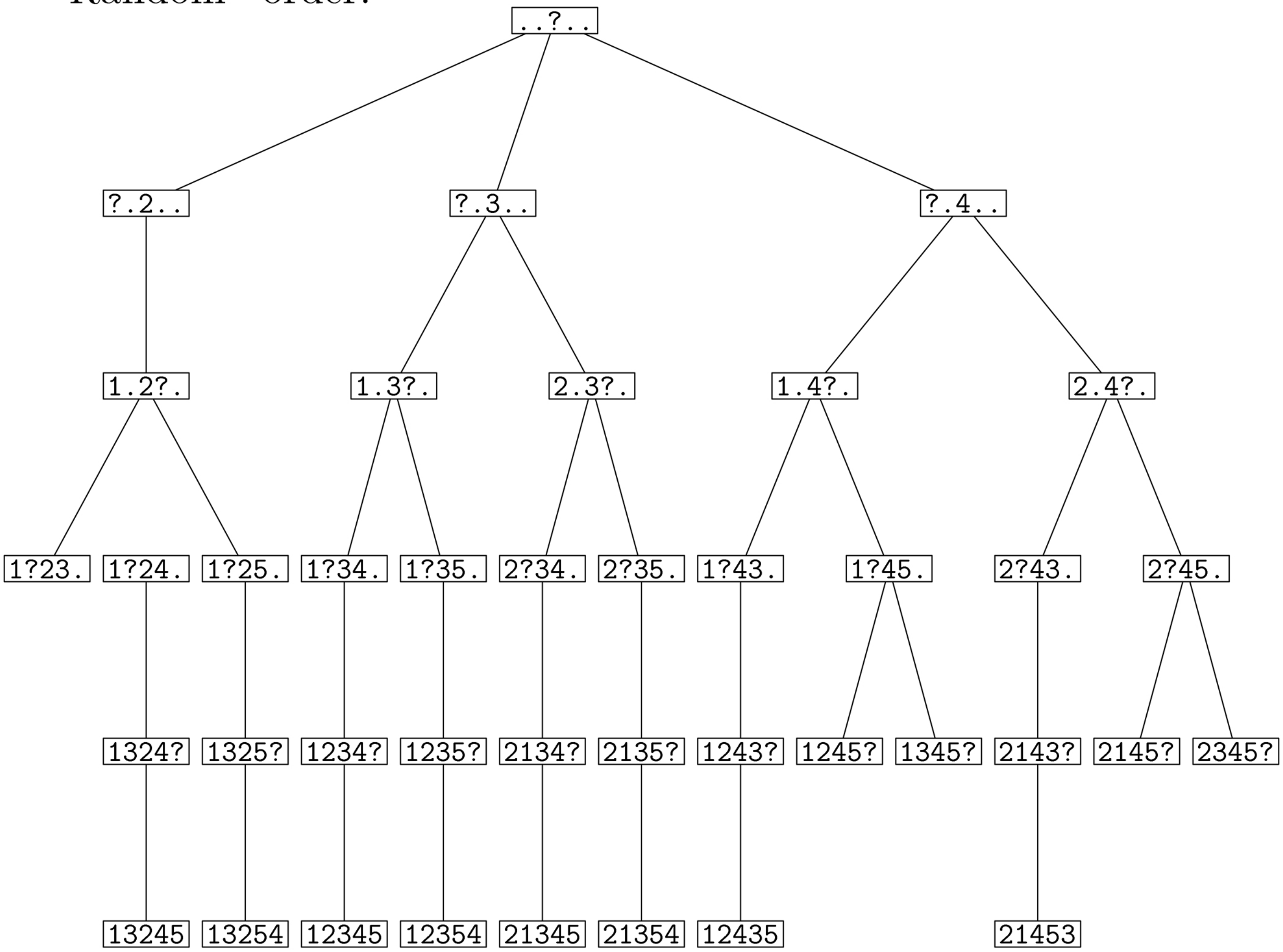
$$2F_{l+1} + F_l - 1 \text{ nodes on level } l \geq 0\,;$$

$$2F_{n+3} - n - 1 \text{ nodes total.}$$

“Random” order:



“Random” order:



Theorem (Ira Gessel, Philippe Jacquet, January 2020):

Average tree size, over all $n!$ orders,

is $\sim c\sqrt{n} \cdot \alpha^n$,

where $\alpha \approx 1.71995$ is the real root of $11x^3 - 18x^2 - x - 1 = 0$.

citations listed by ACM Digital Lib: Belov et al IJCAI 26(2011) 473-479
Franco et al IJCAI 26( ) 4302-4309
Lelis IJCAI 26(2017) 3735-3741
(shortest paths) Lelis et al AI 230 (2016) C 51-73
Lelis et al IJCAI 25 3185-3191
Pandel et al CS&SE 25 80-89
Pandel Amgrad "PACT" 23 (2011) 503-504
Lelis Zilles Holte Auton Agents (2013) 555-582
Lelis Otten Dechter IJCAI 23 594-600

Burns et al JAIR 47 (2013) 697-740
* Lelis Zilles Holte AI 196 (2013) 53-76
Özalin et al INFORMS 23 (2011) 392-403
* Kulkarni, Schröder Formal Concept Anal 4 (2006) 30-44
Tromp, Farnebäck Comput & Games 5 84-99
Kilby et al Natl Conf AI 21 1014-1019
Schröder AI 132 (2001) 105-117
Bailleux Chabrier Intell Data Anal 1 (1997) 263-294
Chen Algorithmica 12 (1994) 458-478

SIAM J. COMPUT.
Vol. 21, No. 2, pp. 295–315, April 1992

007

HEURISTIC SAMPLING: A METHOD FOR PREDICTING THE PERFORMANCE OF TREE SEARCHING PROGRAMS*

PANG C. CHEN†

Abstract. Determining the feasibility of a particular search program is important in practical situations, especially when the computation involved can easily require days, or even years. To help make such predictions, a simple procedure based on a stratified sampling approach is presented. This new method, which is called heuristic sampling, is a generalization of Knuth's original algorithm for estimating the efficiency of backtrack programs. With the aid of simple heuristics, this method can produce significantly more accurate cost estimates for commonly used tree search algorithms such as depth-first, breadth-first, best-first, and iterative-deepening.

Key words. heuristics, stratified sampling, search tree, feasibility testing, cost estimation, Monte Carlo method, analysis of algorithms

AMS(MOS) subject classifications. 68Q25, 65C05

1. Introduction. Tree searching [8] is a general, easily implemented problem-solving technique. Unfortunately, the efficiency of tree searching programs is usually difficult to analyze, even at a rudimentary level. Without analytic cost information, the typical course is to let the computer run until it either finishes the job or exhausts our patience. Switching to a more computational sampling approach provides a less haphazard alternative: We gain accuracy in understanding particular search programs and thus design better ones.

The sampling method that we will discuss generalizes an algorithm of Knuth [5] for estimating properties of a backtrack tree. Starting at the root, Knuth's algorithm extends a partial path by expanding the end node and picking a child according to a uniform distribution. It then forms an unbiased estimate of the tree property using the branching degrees along the randomly selected path. According to Knuth, this simple estimation procedure worked consistently well in his experiments. But as a refinement, Knuth also suggested the technique of importance sampling [2] in which the child is selected according to a weighted distribution, with the weight of each child being an estimate of the property of the corresponding subtree.

TABLE 1
Estimated tree profiles.

Level	h_0	h_1	h_2	Actual
0	1	1	1	1
1	10	10	10	10
2	43	42	34	42
3	257	196	204	251
4	1005	918	864	968
5	4267	3734	3604	4215
6	15684	14026	13724	15646
7	57443	46873	51326	56435
8	184017	126007	174680	182520
9	583197	445188	611950	574555
10	1608183	1613621	1832792	1606422
11	4371678	4679457	5106228	4376153
12	9920481	16793389	11767976	10396422
13	22507886	37750158	25507044	23978392
14	47054083	31161416	47574616	47667686
15	95846027	64365367	83428348	91377173
16	156085563	106268997	130208416	150084206
17	256650972	205655280	192462996	235901901
18	344261889	306859328	252526926	315123658
19	451703678	382647057	280613374	399772215
20	324806815	372852597	277536920	427209856
21	253992102	426672714	272541294	429189112
22	237855139	239446386	288235830	358868304
23	106973568	352833124	226878632	278831518
24	80018150	195532658	138579650	177916192
25	28304640	57001796	87525874	103894319
26	0	72444273	40045814	49302574
27	0	19732265	16732876	21049968
28	0	151248453	9984256	7153880
29	0	0	2985372	2129212
30	0	0	787150	522186
31	0	0	138532	109254
32	0	0	13210	18862
33	0	0	588	2710
34	0	0	588	346
35	0	0	0	50
36	0	0	0	8
total	2422806779	3046195329	2393871699	3137317290
std dev	10672087188	2331712470	878149488	
cost	11503	10875	11345	
trials	1000	40	5	

The Pi graph (an infinite graph on the nonnegative integers):

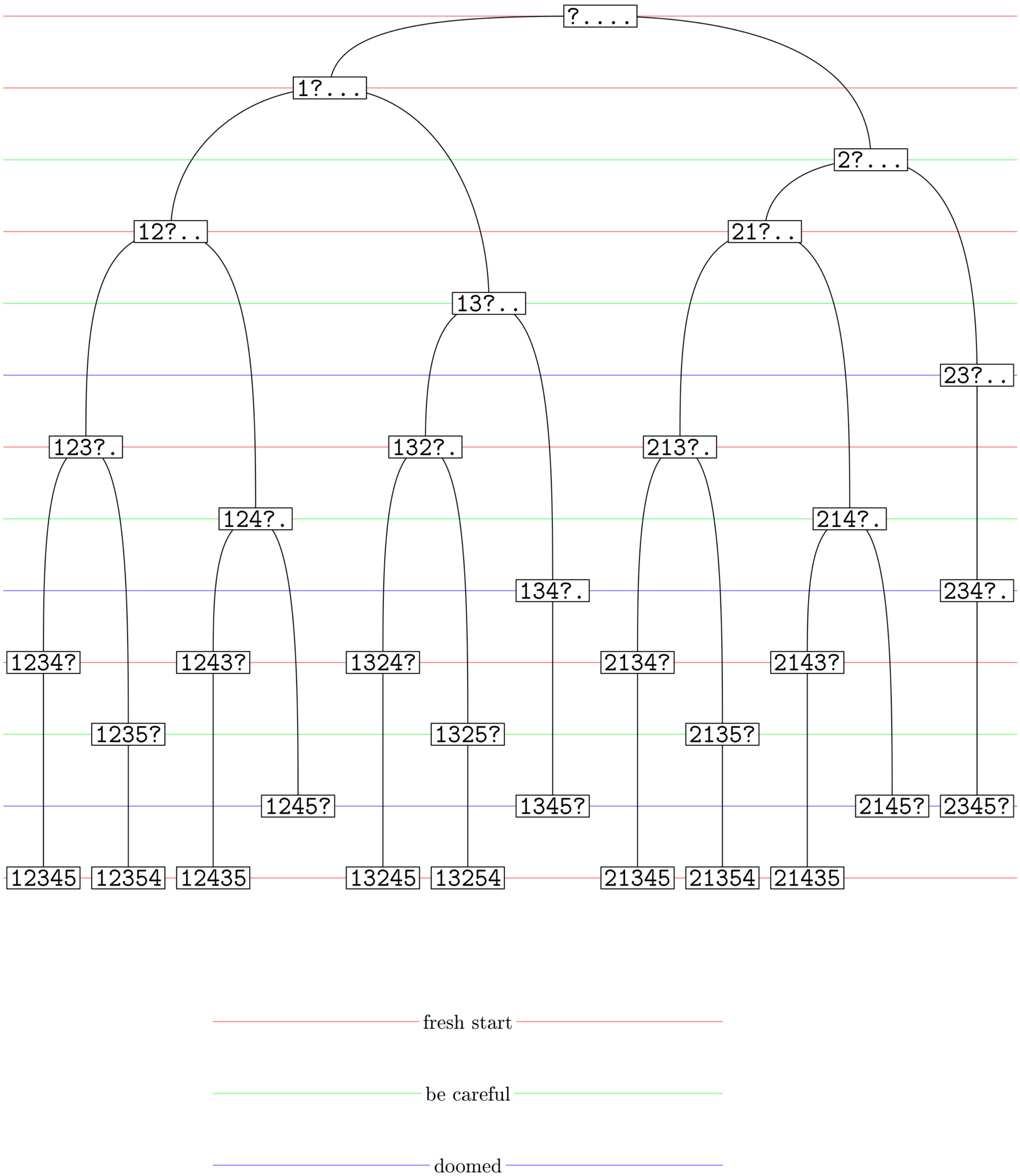
let π's binary digits be

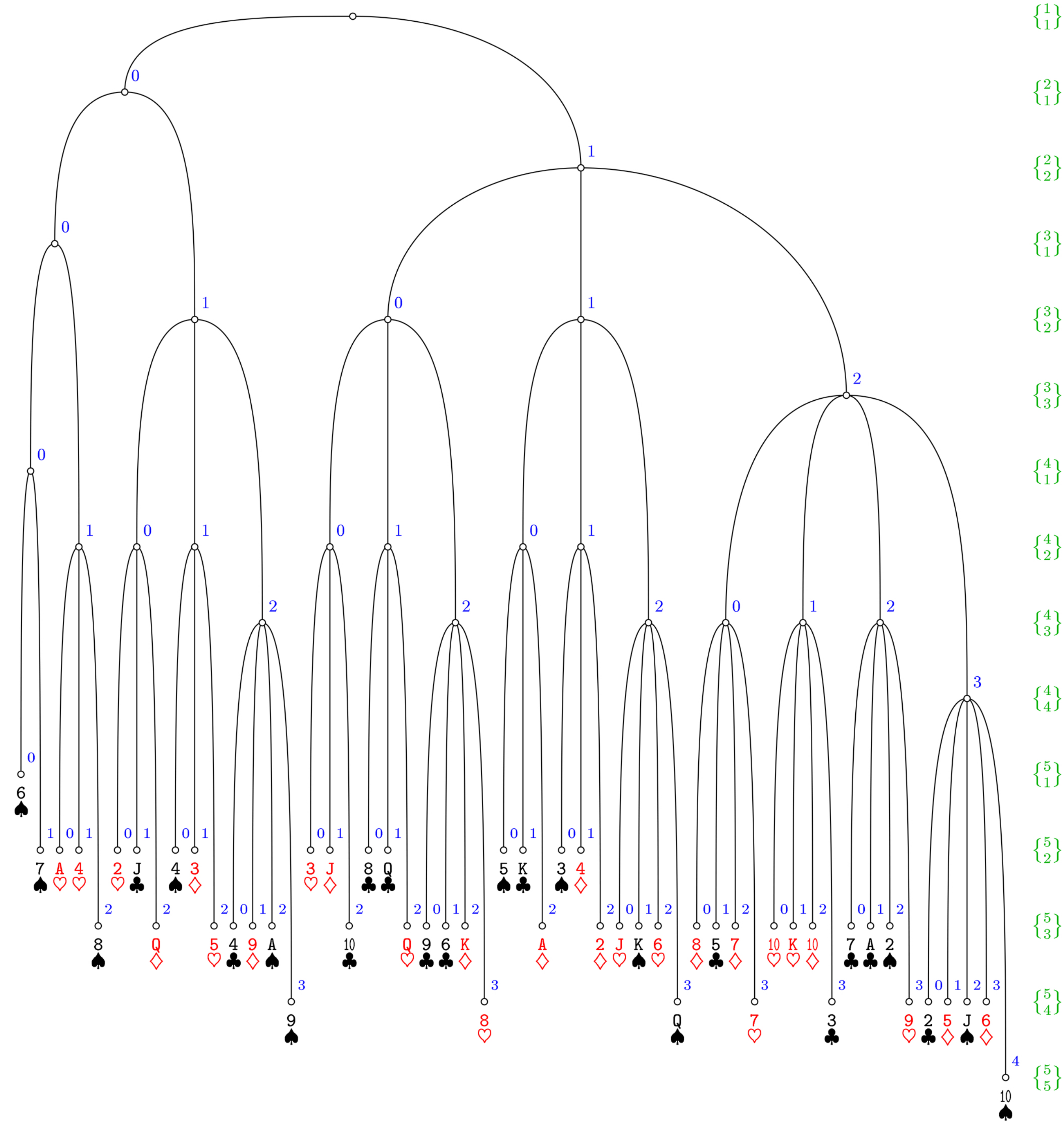
$\pi_0\pi_1\pi_2\pi_3\pi_4\pi_5\pi_6\pi_7\pi_8\pi_9\pi_{10}\pi_{11}\pi_{12}\pi_{13}\ldots =$

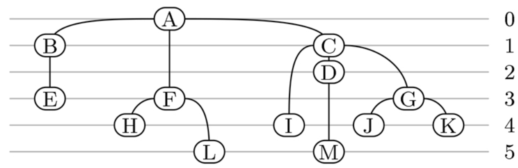
$11001001000011\ldots$; then

$$j \longrightarrow k \text{ and } j < k \quad \Longleftrightarrow \quad \pi_{j+k(k-1)/2} = 1.$$

$$\begin{array}{cccccccccc}
\color{red}{0} & 1 & 1 & 0 & 0 & 0 & 1 & 0 & & \\
1 & \color{red}{0} & 0 & 1 & 1 & 0 & 1 & 1 & & \\
1 & 0 & \color{red}{0} & 0 & 0 & 1 & 1 & 0 & & \\
0 & 1 & 0 & \color{red}{0} & 0 & 1 & 0 & 1 & & \cdots \\
0 & 1 & 0 & 0 & \color{red}{0} & 1 & 1 & 0 & & \\
0 & 0 & 1 & 1 & 1 & \color{red}{0} & 1 & 1 & & \\
1 & 1 & 1 & 0 & 1 & 1 & \color{red}{0} & 0 & & \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & \color{red}{0} & & \\
 & & & \vdots & & & & & & \ddots
\end{array}$$


?....
1?...
2?...
12?..
21?..
13?..
23?..
123?.
132?.
213?.
124?.
214?.
134?.
234?.
1234?
1243?
1324?
2134?
2143?
1235?
1325?
2135?
1245?
1345?
2145?
2345?
12345
12354
12435
13245
13254
21345
21354
21435
fresh start
be careful
doomed

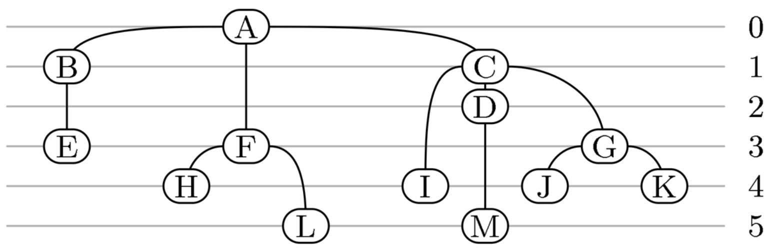



A
B
C
D
E
F
G
H
I
J
K
L
M
0
1
2
3
4
5

Chen subsets:

a subset S of the search tree T
is a *Chen subset* if

i) S is not empty.
ii) If $s \in S$, then $\widehat{s} \in S$. ($\widehat{s}$ denotes the parent of s.)
iii) If $\widehat{s} \in S$, then $s' \in S$ for some node s' with $h(s) = h(s')$.
iv) If $s \in S$ and $s' \in S$ and $s \neq s'$, then $h(s) \neq h(s')$.



a subset S of the search tree T
is a *Chen subset* if

i) S is not empty.
ii) If $s \in S$, then $\widehat{s} \in S$. ($\widehat{s}$ denotes the parent of s.)
iii) If $\widehat{s} \in S$, then $s' \in S$ for some node s' with $h(s) = h(s')$.
iv) If $s \in S$ and $s' \in S$ and $s \neq s'$, then $h(s) \neq h(s')$.