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

Received July 12, 1974.

AMS (MOS) subject classifications (1970). Primary 68A20, 05–04; Secondary 05A15, 65C05, 90B99.

Key words and phrases. Backtrack, analysis of algorithms, Monte Carlo method, Instant Insanity, color cubes, knight's tours, tree functions, branch and bound.

* This research was supported in part by the Office of Naval Research under contract NR 044-402 and by the National Science Foundation under grant number GJ 36473X. Reproduction in whole or in part is permitted for any purpose of the United States Government.

improvements on the original idea have also been developed during the last decade.

The estimation procedure we shall discuss is completely unsophisticated, and it probably has been used without fanfare by many people. Yet the idea works surprisingly well in practice, and some of its properties are not immediately obvious, hence the present paper might prove to be useful.

Section 1 presents a simple example problem, and Section 2 formulates backtracking in general, developing a convenient notational framework; this treatment is essentially self-contained, assuming no prior knowledge of the backtrack literature. Section 3 presents the estimation procedure in its simplest form, together with some theorems that describe the virtues of the method. Section 4 takes the opposite approach, by pointing out a number of flaws and things that can go wrong. Refinements of the original method, intended to counteract these difficulties, are presented in Section 5. Some computational experiments are recorded in Section 6, and Section 7 summarizes the practical experience obtained with the method to date.

1. Introduction to Backtrack. It is convenient to introduce the ideas of this paper by looking first at a small example. The problem we shall study is actually a rather frivolous puzzle, so it does not display the economic benefits of backtracking; but it does have the virtue of simplicity, since the complete solution can be displayed in a small diagram. Furthermore the puzzle itself seems to have been tantalizing people for at least sixty years (see [19]); it became extremely popular in the U.S.A. about 1967 under the name *Instant Insanity*.

Figure 1 shows four cubes whose faces are colored red (R), white (W), green (G), or blue (B); colors on the hidden faces are shown at the sides. The problem is to arrange the cubes in such a way that each of the four colors appears exactly once on the four back faces, once on the top, once in the front, and once on the bottom. Thus Figure 1 is not a solution, since there is no blue on the top nor white on the bottom; but a solution is obtained by rotating each cube 90°.

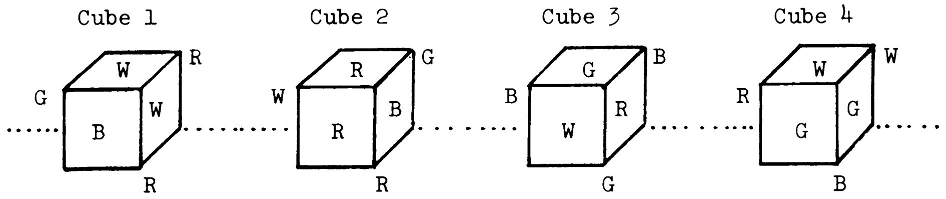


FIGURE 1. *Instant Insanity cubes*

We can assume that these four cubes retain their relative left-to-right order in all solutions. Each of the six faces of a given cube can be on the bottom, and there are four essentially different positions having a given bottom face, so each cube can be placed in 24 different ways; therefore the "brute force" approach to this problem is to try all of the $24^4 = 331776$ possible configurations. If done by hand, the brute force procedure might indeed lead to insanity, although not instantly.

It is not difficult to improve on the brute force approach by considering the effects of symmetry. Any solution clearly leads to seven other solutions, by simultaneously rotating the cubes about a horizontal axis parallel to the dotted line in Figure 1, and/or by rotating each cube 180° about a vertical axis. Therefore we can assume without loss of generality that Cube 1 is in one of three positions, instead of considering all 24 possibilities. Furthermore it turns out that Cube 2 has only 16 essentially different placements, since it has two opposite red faces; see Figure 2, which shows that two of its 24 positionings have the same colors on the front, top, back, and bottom faces. The same observation applies to Cube 3. Hence the total number of essentially different ways to position the four cubes is only $3 \cdot 16 \cdot 16 \cdot 24 = 18432$; this is substantially less than 331776, but it might still induce insanity.

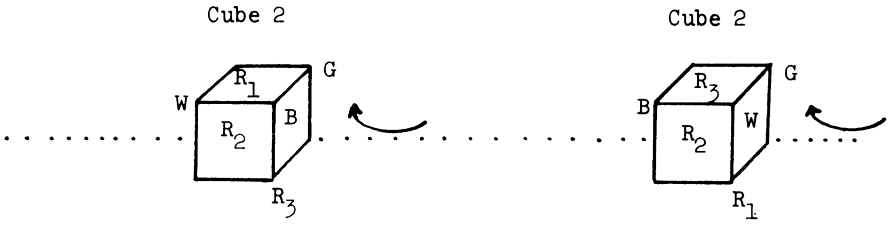


FIGURE 2. *Rotation by* 180° *in this case leaves the relevant colors unchanged*

A natural way to reduce the number of cases still further now suggests itself. Given one of the three placements for Cube 1, some of the 16 positionings of Cube 2 are obviously foolhardy since they cannot possibly lead to a solution. In Figure 1, for example, Cubes 1 and 2 both contain red on their bottom face, while a complete solution has no repeated colors on the bottom, nor on the front, top, or back; since this placement of Cube 2 is incompatible with the given position of Cube 1, we need not consider any of the $16 \cdot 24 = 384$ ways to place Cubes 3 and 4. Similarly, when Cubes 1 and 2 have been given a compatible placement, it makes sense to place Cube 3 so as to avoid duplicate colors on the relevant sides, before we even begin to consider Cube 4.

Such a sequential placement can be represented by a tree structure, as shown in Figure 3. The three nodes just below the root (top) of this tree stand for the three essentially different ways to place Cube 1. Below each such node are further nodes representing the possible placements of Cube 2 in a compatible position; and below the latter are the compatible placements of Cube 3 (if any), etc. Note that there is only one solution to the puzzle, represented by the single node on Level 4.

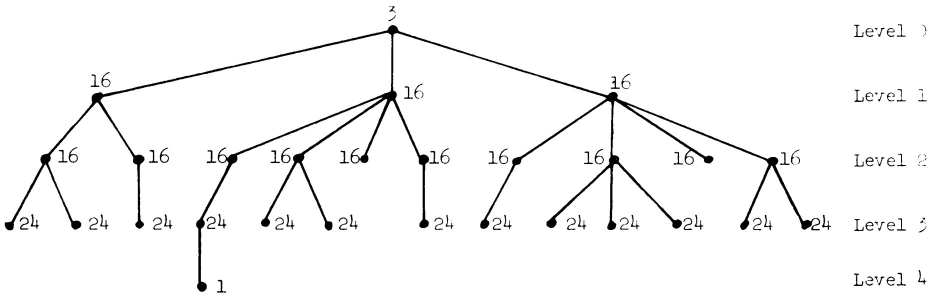


FIGURE 3. *The Instant Insanity tree*

This procedure cuts the number of cases examined to $3 + 3 \cdot 16 + 10 \cdot 16 + 13 \cdot 24 + 1 = 524$; for example, each of the 10 nodes on Level 2 of the tree involves the consideration of 16 ways to place Cube 3. It is reasonable to assume that a sane person can safely remain *compos mentis* while examining 524 cases; thus, we may conclude that systematic enumeration can cut the work by several orders of magnitude even in simple problems like this one. (Actually a further refinement, which may be called the technique of "homomorphism and lifting", can be applied to the Instant Insanity problem, reducing the total number of cases examined to about 50, as shown originally in [1]; see also [7] for further discussion and for a half-dozen recent references. But such techniques are beyond the scope of the present paper.)

The tree of Figure 3 can be explored in a systematic manner, requiring comparatively little memory of what has gone before. The idea is to start at the root and continually to move downward when possible, taking the leftmost branch whenever a decision is necessary; but if it is impossible to continue downward, "backtrack" by considering the next alternative on the previous level. This is a special case of the classical Trémaux procedure for exploring a maze [17, pp. 47–50], [13, Chapter 3].

2. The General Backtrack Procedure. Now that we understand the Instant Insanity example, let us consider backtracking in general. The problem we wish to solve can be expressed abstractly as the task of finding all sequences $(x_1, x_2, \cdots, x_n)$ which satisfy some property $P_n(x_1, x_2, \cdots, x_n)$. For example, in the case of Instant Insanity, $n = 4$; the symbol x_k denotes a placement of the kth cube; and $P_r(x_1,x_2,x_3,x_4)$ is the property that the four cubes exhibit all four colors on all four relevant sides.

The general backtrack approach consists of inventing intermediate properties $P_k(x_1, \cdots, x_k)$ such that

$$(1) \qquad P_{k+1}(x_1, \cdots, x_k, x_{k+1}) \quad \text{implies} \quad P_k(x_1, \cdots, x_k) \quad \text{for } 0 \leqslant k < n.$$

In other words, if $(x_1, \cdots, x_k)$ *does not* satisfy property P_k, then no extended sequence $(x_1, \cdots, x_k, x_{k+1})$ can possibly satisfy P_{k+1}; hence by induction, no extended sequence $(x_1, \cdots, x_k, \cdots, x_n)$ can solve the original condition P_n. The backtrack procedure systematically enumerates all solutions $(x_1, \cdots, x_n)$ to the original problem by considering all partial solutions $(x_1, \cdots, x_k)$ that satisfy P_k, using the following general algorithm:

Step B1. [Initialize.] Set k to 0.

Step B2. [Compute the successors.] (Now $P_k(x_1, \cdots, x_k)$ holds, and $0 \leqslant k < n$.) Set S_k to the set of all x_{k+1} such that $P_{k+1}(x_1, \cdots, x_k, x_{k+1})$ is true.

Step B3. [Have all successors been tried?] If S_k is empty, go to Step B6.

Step B4. [Advance.] Choose any element of S_k, call it x_{k+1}, and delete it from S_k. Increase k by 1.

Step B5. [Solution found?] (Now $P_k(x_1, \cdots, x_k)$ holds, and $0 < k \leqslant n$.) If $k < n$, return to Step B2. Otherwise output the solution $(x_1, \cdots, x_n)$ and go on to Step B6.

Step B6. [Backtrack.] (All extensions of $(x_1, \cdots, x_k)$ have now been explored.) Decrease k by 1. If $k \geqslant 0$, return to Step B3; otherwise the algorithm terminates. □

Condition (1) does not uniquely define the intermediate properties P_k, so we often have considerable latitude when we choose them. For example, we could simply let P_k be true for *all* $(x_1, \cdots, x_k)$, when $k < n$; this is the weakest possible property satisfying (1), and it corresponds to the brute force approach, where some 24^4 possibilities would be examined in the cube problem. On the other hand the strongest property is obtained when $P_k(x_1, \cdots, x_k)$ is true if and only if there exist $x_{k+1}, \cdots, x_n$ satisfying $P_n(x_1, \cdots, x_k, x_{k+1}, \cdots, x_n)$. In our example this strongest property would reduce the search to the examination of a trivial twig of a tree, but the decisions at each node would require considerable calculation. In general, stronger properties limit the search but require more computation, so we want to find a suitable trade-off. The solution adopted in our example (namely to use symmetry considerations when placing Cubes 1, 2, and 3, and to let $P_k(x_1, \cdots, x_k)$ mean that no colors are duplicated on the four relevant sides) is fairly obvious, but in other problems the choice of P_k is not always so self-evident.

3. A Simple Estimate of the Running Time.

For each $(x_1, \cdots, x_k)$ satisfying P_k with $0 < k < n$, the algorithm of Section 2 will execute Steps B2, B4, B5, and B6 once, and Step B3 twice. (To see this, note that it is true for Steps B2, B5, and B6, and apply Kirchhoff's law as in [12].) Let us call the associated running time the *cost* $c(x_1, \cdots, x_k)$. When $k = n$, the corresponding cost amounts to one execution of Steps B3, B4, B5, and B6. If we also let $c(\ )$ be the cost for $k = 0$ (i.e., one execution of Steps B1, B2, B3, and B6), the total running time of the algorithm comes to exactly

$$\sum_{k \geqslant 0} \sum_{P_k(x_1, \cdots, x_k)} c(x_1, \cdots, x_k). \tag{2}$$

This formula essentially distributes the total cost among the various nodes of the tree. Since the time to execute Step B2 can vary from node to node, and since the time to execute Step B5 depends on whether or not $k = n$, the running time is not simply proportional to the size of the tree except in simple cases.

Let T be the tree of all possibilities explored by the backtrack method; i.e., let

$$T = \{(x_1, \cdots, x_k) | k \geqslant 0 \text{ and } P_k(x_1, \cdots, x_k) \text{ holds}\}. \tag{3}$$

Then we can rewrite (2) as

$$\text{cost}(T) = \sum_{t \in T} c(t). \tag{4}$$

Our goal is to find some way of estimating $\text{cost}(T)$, without knowing a great deal about the properties P_k, since the example of Section 1 indicates that these properties might be very complex.

A natural solution to this estimation problem is to try a Monte Carlo approach, based on a random exploration of the tree; for each partial solution $(x_1, \cdots, x_k)$ for $0 \leqslant k < n$, we can choose x_{k+1} at random from among the set S_k of all continuations, as in the following algorithm. (A related procedure, but which is intrinsically different because it is oriented to different kinds of estimates, has been published by Hammersley and Morton [10], and it has been the subject of numerous papers in the literature of mathematical physics; see [5].)

Step E1. [Initialize.] Set $k \longleftarrow 0, D \longleftarrow 1$, and $C \longleftarrow c(\,)$. (Here C will be an estimate of (2), and D is an auxiliary variable used in the calculation of C, namely the product of all "degrees" encountered in the tree. An arrow " $\longleftarrow$ " denotes the assignment operation equivalent to Algol's " := "; and $c(\,)$ denotes the cost at the root of the tree, as in (2) when $k = 0$.)

Step E2. [Compute the successors.] Set S_k to the set of all x_{k+1} such that $P_{k+1}(x_1, \cdots, x_k, x_{k+1})$ is true, and let d_k be the number of elements of S_k. (If $k = n$, then S_k is empty and $d_k = 0$.)

Step E3. [Terminal position?] If $d_k = 0$, the algorithm terminates, with C an estimate of $\text{cost}(T)$.

Step E4. [Advance.] Choose an element $x_{k+1} \in S_k$ at random, each element being equally likely. (Thus, each choice occurs with probability $1/d_k$.) Set $D \longleftarrow d_k D$, then set $C \longleftarrow C + c(x_1, \cdots, x_{k+1})D$. Increase k by 1 and return to Step E2. □

This algorithm makes a random walk in the tree, without any backtracking, and computes the estimate

$$C = c(\,) + d_0 c(x_1) + d_0 d_1 c(x_1, x_2) + d_0 d_1 d_2 c(x_1, x_2, x_3) + \cdots, \tag{5}$$

where d_k is a function of $(x_1, \cdots, x_k)$, namely the number of x_{k+1} satisfying $P_{k+1}(x_1, \cdots, x_k, x_{k+1})$. We may define $d_k = 0$ for all large k, thereby regarding (5) as an infinite series although only finitely many terms are nonzero.

The validity of estimate (5) can be proved as follows.

THEOREM 1. *The expected value of* C, *as computed by the above algorithm, is* cost (T), *as defined in* (4).

PROOF. We shall consider two proofs, at least one of which should be convincing. First we can observe that for every $t = (x_1, \cdots, x_k) \in T$, the term

$$d_0 d_1 \cdots d_{k-1} c(x_1, \cdots, x_k) \tag{6}$$

occurs in (5) with probability $1/d_0 d_1 \cdots d_{k-1}$, since this is the chance that the algorithm will consider the partial solution $(x_1, \cdots x_k)$. Hence the sum of all the terms (6) has the expected value (4).

The second proof is based on a recursive definition of cost (T), namely

$$\text{cost}\,(T) = c(\ ) + \text{cost}\,(T_1) + \cdots + \text{cost}\,(T_d), \tag{7}$$

where $d = d_0$ is the degree of the root of the tree and $T_1, \cdots, T_d$ are the respective subtrees of the root, namely

$$T_j = \{(x_1, \cdots, x_k) \in T \mid x_1 \text{ is the } j\text{th element of } S_0\}.$$

We also have $C = c(\ ) + d_0 C'$, where $C' = c(x_1) + d_1 c(x_1, x_2) + d_1 d_2 c(x_1, x_2, x_3) + \cdots$ has the form of (5) and is an estimate of one of the T_j. Since each of the $d = d_0$ values of j is equally likely, the expected value of C is

$$E(C) = c(\ ) + d_0 E(C') = c(\ ) + d_0((E(C_1) + \cdots + E(C_d))/d),$$

where $E(C_j) = \text{cost}\,(T_j)$ by induction on the size of the tree. Hence $E(C) = \text{cost}\,(T)$. □

This theorem demonstrates that C is indeed an appropriate statistic to compute, based on one random walk down the tree. As an example of the theorem, let us consider Figure 3 in Section 1, using the costs shown there (since they represent the time to perform Step B2, which dominates the calculation). We have cost $(T) = 524$, and if the estimation algorithm is applied to the tree it is not difficult to determine that the result will be $C = 243$, or 291, or 435, or 531, or 543, or 819, or 1107, with respective probabilities 1/6, 1/6, 1/6, 1/6, 1/12, 1/6, and 1/12. Thus, a fairly reasonable approximation will nearly always be obtained; and we know that the mean of repeated estimates will approach 524, by the law of large numbers.

Since the proof of Theorem 1 applies to all functions $c(t)$ defined over trees, we can apply it to other functions in order to obtain further information:

COROLLARY 1. *The expected value of* D *at the end of the above algorithm is the number of terminal nodes in the tree.*

PROOF. Let $c(t) = 1$ if t is terminal, and $c(t) = 0$ otherwise; then $C = D$ at the end of the algorithm, hence $E(D) = E(C) = \Sigma c(t)$ is the number of terminal nodes by Theorem 1. □

COROLLARY 2. *The expected value of the product* $d_0 d_1 \cdots d_{k-1}$ *for fixed* k, *when the d_j's are computed by the above algorithm, is the number of nodes on level k of the tree.*

PROOF. Let $c(t) = 1$ for all nodes on level k, and $c(t) = 0$ otherwise; then $C = d_0 d_1 \cdots d_{k-1}$ at the end of the algorithm. (Note that $d_0 d_1 \cdots d_{k-1}$ is zero if the algorithm terminates before reaching level k.) □

Corollary 2 gives some insight into the "meaning" of the individual terms of our estimate (5); the term $d_0 d_1 \cdots d_{k-1} \cdot c(x_1, \cdots, x_k)$ represents the number of nodes on level k times the cost associated with a typical one of these nodes.

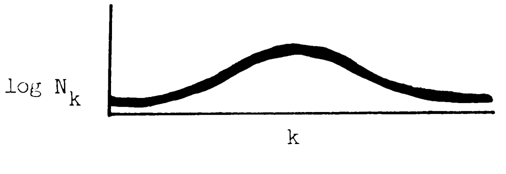
4. Some Cautionary Remarks. The algorithm of Section 3 seems too simple to work, and there are many intuitive grounds for skepticism, since we are trying to predict the characteristics of an entire tree based on the knowledge of only one branch! The combinatorial realities of most backtrack applications make it clear that different partial solutions can have drastically different behavior·patterns.

Just knowing that an experiment yields the right expected value is not much consolation in practice. For example, consider an experiment which produces a result of 1 with probability 0.999, while the result is 1,000,001 with probability 0.001; the expected value is 1001, but a limited sampling would almost always convince us that the true answer is 1.

There is reason to suspect that the estimation procedure of Section 3 will suffer from precisely this defect: It has the potential to produce huge values, but with very low probability, so that the expected value might be quite different from typical estimates.

Let N_k be the number of nodes on level k of the tree (cf. Corollary 2). In most backtrack applications, the vast majority of all nodes in the search tree are concentrated at only a few levels, so that in fact the *logarithm* of N_k (the number of digits in N_k) has a bell-shaped curve when plotted as a function of k:

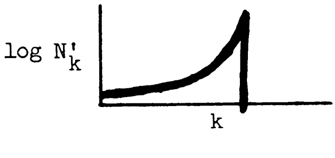
(8)



.

On the other hand our estimate (5) is composed of a series of estimates $N'_k = d_0 d_1 \cdots d_{k-1}$ which are never bell-shaped; since the d's are integers, the N'_k grow exponentially with k, until finally dropping to zero:

(9)



.

Although these two graphs have cqmpletely different characteristics, we are getting estimates which in the long run produce (8) as an average of curves like (9).

Consider also Figure 3, where we have somewhat arbitrarily assigned a cost of 1 to the lone solution node on Level 4. Perhaps our output routine is so slow that the solution node should really have a cost of 10^6; this now becomes the dominant portion of the total cost, but it will be considered only $1/12$ of the time, and then it will be multiplied by 12.

There is clearly a danger that our estimates will almost always be low, except for rare occasions when they will be much too high.

5. Refinements. Our estimation procedure can be modified in order to circumvent the difficulties sketched in Section 4. One idea is to introduce systematic bias into Step E4, so that the choice of x_{k+1} is not completely random; we can try to investigate the more interesting or more difficult parts of the tree.

The algorithm can be generalized by using the following selection procedure in place of Step E4.

Step E4′. [Generalized advance.] Determine, in any arbitrary fashion, a sequence of d_k positive numbers $p_k(1), p_k(2), \cdots, p_k(d_k)$ whose sum is unity. Then choose a random integer J_k in the range $1 \leqslant J_k \leqslant d_k$ in such a way that $J_k = j$ with probability $p_k(j)$. Let x_{k+1} be the J_kth element of S_k, and set $D \longleftarrow D/p_k(J_k)$, $C \longleftarrow C + c(x_1, \cdots, x_{k+1})D$. Increase k by 1 and return to Step E2. □

(Step E4 is the special case $p_k(j) = 1/d_k$ for all j.) Again we can prove that the expected value of C will be $\operatorname{cost}(T)$, no matter how strangely the probabilities $p_k(j)$ are biased in Step E4′; in fact, both proofs of Theorem 1 are readily extended to yield this result. It is interesting to note that the calculation of D involves *a posteriori* probabilities, so that it grows only slightly after a highly probable choice has been made. The technique embodied in Step E4′ is generally known as *importance sampling* [9, pp. 57–59].

Some choices of the $p_k(j)$ are much better than others, of course, and the most interesting fact is that one of the possible choices is actually perfect:

THEOREM 2. *If the probabilities* $p_k(j)$ *in Step* E4′ *are chosen appropriately, the estimate* C *will always be exactly equal to* $\operatorname{cost}(T)$.

PROOF. For $1 \leqslant j \leqslant d_k$, let $p_k(j)$ be

$$(10) \qquad p_k^*(j) = \frac{\operatorname{cost}(T(x_1, \cdots, x_k, x_{k+1}(j)))}{\operatorname{cost}(T(x_1, \cdots, x_k)) - c(x_1, \cdots, x_k)},$$

where $T(x_1, \cdots, x_k)$ is the set of all $t \in T$ having specified values $(x_1, \cdots, x_k)$ for the first k components, and where $x_{k+1}(j)$ is the jth element of S_k. Now we can prove that the relation

$$C + (\operatorname{cost}(T(x_1, \cdots, x_k)) - c(x_1, \cdots, x_k))D = \operatorname{cost}(T)$$

is invariant, in the sense that it always holds at the beginning and end of Step E4′.

Since $\operatorname{cost}(T(x_1,\cdots,x_k)) = c(x_1,\cdots,x_k)$ when $d_k = 0$, the algorithm terminates with $C = \operatorname{cost}(T)$.

Alternatively, using the notation in the second proof of Theorem 1, we have

$$C = c(\ ) + ((\operatorname{cost}(T) - c(\ ))/\operatorname{cost}(T_j))C_j$$

for some j, and $C_j = \operatorname{cost}(T_j)$ by induction, hence $C = \operatorname{cost}(T)$. □

Of course we generally need to know the cost of the tree before we know the exact values of these ideal probabilities $p_k^*(j)$, so we cannot achieve zero variance in practice. But the form of the $p_k^*(j)$ shows what kind of bias is likely to reduce the variance; any information or hunches that we have about relative subtree costs will be helpful. (In the case of Instant Insanity there is no simple *a priori* reason to prefer one cube position over another, so this idea does not apply; perhaps Instant Insanity is a mind-boggling puzzle for precisely this reason, since intuition is usually much more valuable.)

Theorem 2 can be extended considerably, in fact we can derive a general formula for the variance. The generating function for C satisfies

$$C(z) = z^{c(\ )} \sum_{1\leqslant j\leqslant d} p_j C_j(z^{1/p_j}) \tag{11}$$

and from this equation it follows by differentiation that

$$\begin{aligned}\operatorname{var}(C) &= C''(1) + C'(1) - C'(1)^2\\ &= \sum_{1\leqslant j\leqslant d} \operatorname{var}(C_j)/p_j + \sum_{1\leqslant i<j\leqslant d} p_i p_j \left(\frac{\operatorname{cost}(T_i)}{p_i} - \frac{\operatorname{cost}(T_j)}{p_i}\right)^2 .\end{aligned} \tag{12}$$

Iterating this recurrence shows that the variance can be expressed as

$$\operatorname{var}(C) = \sum_{t\in T} \frac{1}{P(t)} \sum_{1\leqslant i<j\leqslant d(t)} p^t(i)p^t(j)\left(\frac{\operatorname{cost}(T(t,i))}{p^t(i)} - \frac{\operatorname{cost}(T(t,j))}{p^t(j)}\right)^2 , \tag{13}$$

where $P(t)$ is the probability that node t is encountered, $d(t)$ is the degree of node t, $p^t(j)$ is the probability that we go from t to its jth successor, and $T(t,j)$ is the subtree rooted at that successor.

From this explicit formula we can get a bound on the variance, if the probabilities are reasonably good approximations to the relative subtree costs:

THEOREM 3. *If the probabilities* $p_k(j)$ *in Step* $E4'$ *satisfy*

$$\frac{\operatorname{cost}(T(x_1,\cdots,x_k,x_{k+1}(j)))}{p_k(j)} \leqslant \alpha \frac{\operatorname{cost}(T(x_1,\cdots,x_k,x_{k+1}(i)))}{p_k(i)}$$

for all i, j *and for some fixed constant* $\alpha \geqslant 1$, *the variance of* C *is at most*

$$\left(\left(\frac{\alpha^2 + 2\alpha + 1}{4\alpha}\right)^n - 1\right) \operatorname{cost}(T)^2 . \tag{14}$$

Proof. Let $q_j = \operatorname{cost}(T_j)/p_j$, and assume without loss of generality that $q_1 \leqslant q_2 \leqslant \cdots \leqslant q_d \leqslant \alpha q_1$. From elementary calculus we have, under the constraints $\operatorname{cost}(T_j) \geqslant 0$ and $\Sigma\, p_j = 1$,

$$\sum_{1\leqslant j\leqslant d} \frac{\operatorname{cost}(T_j)^2}{p_j} \leqslant \frac{\alpha^2 + 2\alpha + 1}{4\alpha}\left(\sum_{1\leqslant j\leqslant d} \operatorname{cost}(T_j)\right)^2,$$

equality occurring when $d = 2$ and $q_2 = \alpha q_1$. Furthermore

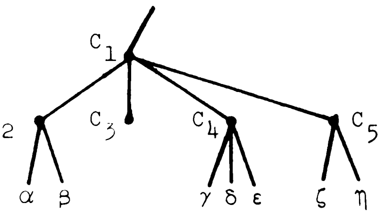
$$\sum_{1\leqslant i<j\leqslant d} p_i p_j (q_i - q_j)^2 = \sum_{1\leqslant j\leqslant d} \frac{\operatorname{cost}(T_j)^2}{p_j} - \left(\sum_{1\leqslant j\leqslant d} \operatorname{cost}(T_j)\right)^2.$$

Letting $\beta = (\alpha^2 + 2a + 1)/4\alpha$, we can prove (14) by induction since (12) now yields

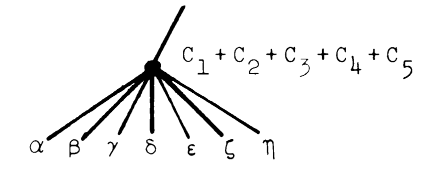
$$\begin{aligned}\operatorname{var}(C) &\leqslant \sum_{1\leqslant j\leqslant d} \operatorname{var}(C_j)/p_j + (\beta - 1)\operatorname{cost}(T)^2\\ &\leqslant \sum_{1\leqslant j\leqslant d} (\beta^{n-1} - 1)\operatorname{cost}(T_j)^2/p_j + (\beta - 1)\operatorname{cost}(T)^2\\ &\leqslant (\beta^n - \beta)\operatorname{cost}(T)^2 + (\beta - 1)\operatorname{cost}(T)^2 . \quad \square\end{aligned}$$

Theorem 3 implies Theorem 2 when $\alpha = 1$; for $\alpha > 1$ the bound in (14) is not especially comforting, but it does indicate that a few runs of the algorithm will probably predict $\operatorname{cost}(T)$ with the right order of magnitude.

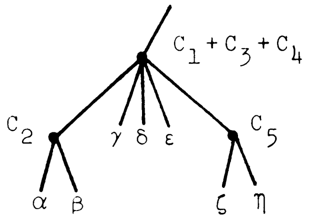
Another way to improve the estimates is to transform the tree into another one having the same total cost, and to apply the Monte Carlo procedure to the transformed tree. For example, the tree fragment



with costs $C_1, \cdots, C_5$ and subtrees $\alpha, \cdots, \eta$ can be replaced by


$C_1 + C_2 + C_3 + C_4 + C_5$
α β γ δ ε ζ η

by identifying five nodes. Intermediate condensations such as



are also possible.

One application of this idea, if the estimates are being made by a computer program, is to eliminate all nodes on levels 1, 3, 5, 7, · · · of the original tree, making the nodes formerly on levels $2k$ and $2k + 1$ into a new level k. For example, Figure 4 shows the tree that results when this idea is applied to Figure 3. The estimates in this collapsed tree are $C = 211$, or 451, or 461, or 691, or 931, with respective probabilities .2, .3, .1, .3, .1, so we have a slightly better distribution than before.

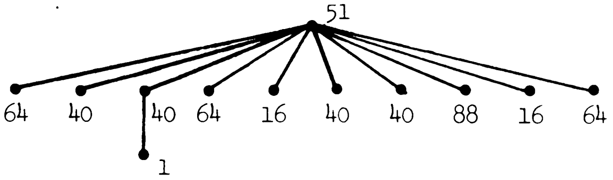


FIGURE 4. *Collapsed Instant Insanity tree*

Another use of this idea is to eliminate all terminal nodes having nonterminal "brothers". Then we can ensure that the algorithm never moves directly to a configuration having $d_k = 0$ unless all possible moves are to such a terminal situation; in other words, "stupid" moves can be avoided.

Still another improvement to the general estimation procedure can be achieved by "stratified sampling" [9, pp. 55–57]. We can reduce the variance of a series of estimates by insisting for example that each experiment chooses a different value of x_1.

6. Computational Experience. The method of Section 3 has been tested on dozens of applications; and despite the dire predictions made in Section 4 it has consistently performed amazingly well, even on problems which were intended to serve as bad examples. In virtually every case the right order of magnitude for the tree size was found after ten trials. Three or four of the ten trials would typically be gross underestimates, but they were generally counterbalanced by overestimates, in the right proportion.

We shall describe only the largest experiment here, since the method is of most critical importance on a large tree. Figure 5 illustrates the problem that was considered, the enumeration of uncrossed knight's tours; these are nonintersecting paths of a knight on the chessboard, where the object is to find the largest possible tour of this kind. T. R. Dawson first proposed the problem in 1930 [2], and he gave the two 35-

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.

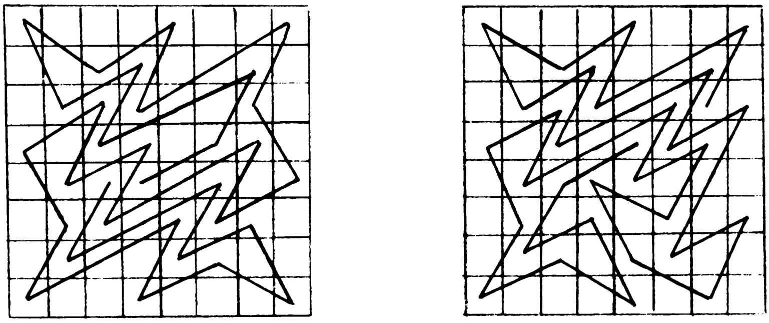


FIGURE 5. *Uncrossed knight's tours*

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
315291281
8231
1793651
59761491

209749511
58736818301
311
259271
6071489081

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

567911
111
569585831
111
411

238413491
6697691
5848873631
161
140296511

for an average of only 680,443,586, although the four extremely low estimates make this value look highly suspicious. (We could have avoided the "stupid moves" which lead to such low estimates, by using the technique explained at the end of Section 5, but the original method was being followed faithfully here.) After 100 experiments had been conducted, the observed mean value of the estimates was 1,653,634,783.8, with an observed standard deviation of about 6.7×10^9.

The first few experiments were done by hand, but then a computer program was written and it performed 1000 experiments in about 30 seconds. The results of these experiments were extremely encouraging, because they were able to predict the size of the tree quite accurately as well as its "shape" (i.e., the number N_k of nodes per level), even though the considerations of Section 4 seem to imply that N_k cannot be estimated well. Table 1 shows how these estimates compare to the exact values which were calculated later; there is surprisingly good agreement, although the experiment looked at less than 0.00001 of the nodes of the tree. Perhaps this was unusually good luck.

This knight's tour problem reflects the typical growth of backtrack trees; the same problem on a 7×7 board generates a tree with 10,874,674 nodes and on a 6×6 board there are only 88,467. On a 9×9 board we need another method; the longest known tour has 47 moves [18]. It can be shown that the longest reentrant tour on an $n \times n$ board has at least $n^2 - O(n)$ moves, see [11].

7. Use of the Method in Practice. There are two principal ways to apply this estimation method, namely by hand and by machine.

Hand calculation is especially recommended as the first step when embarking on any backtrack computations. For one thing, the algorithm is great fun to apply, especially when decimal dice [20] are used to guide the decisions. The reader is urged to try constructing a few random uncrossed knight's tours, recording the statistics d_k as the tours materialize; it is a captivating game that can lead to hours of enjoyment until the telephone rings.

Furthermore, the game is worthwhile, because it gives insight into the behavior of the algorithm, and such insight is of great use later when the algorithm is eventually programmed; good ideas about data structures, and about various improvements in the backtracking strategy, usually suggest themselves. The assignment of nonuniform probabilities as suggested in Section 5 seems to improve the quality of the estimates, and adds interest to the game. Usually about three estimates are enough to give a feeling for the amount of work that will be involved in a full backtrack search.

For large-scale experiments, expecially when considering the best procedure in some family of methods involving parameters that must be selected, the estimates can be done rapidly by machine. Experience indicates that most of the refinements suggested in Section 5 are unnecessary; for example, the idea of collapsing the tree into half as many levels does not improve the quality of the estimates sufficiently to justify the greatly increased computation. Only the partial collapsing technique which avoids "stupid moves" is worth the effort, and even this makes the program so much more

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

complex that it should probably be used only when provided as a system subroutine. (A collection of system routines or programming language features, that allow both the estimation algorithm and full backtracking to be driven by the same source language program, is useful.)

Perhaps the most important application of backtracking nowadays is to combinatorial optimization problems, as first suggested by D. H. Lehmer [15, pp. 168–169]. In this case the method is commonly called a *branch-and-bound* technique (see [14]). The estimation procedure of Section 3 does not apply directly to branch-and-bound algorithms; however, it is possible to estimate the amount of work needed to test any given bound for optimality. Thus we can get a good idea of the running time even in this case, provided that we can guess a reasonable bound. Again, hand calculations using a Monte Carlo approach are recommended as a first step in the approach to all branch-and-bound procedures, since the random experiments provide both insight and enjoyment.

Acknowledgments. I wish to thank Robert W. Floyd and George W. Soules for several stimulating conversations relating to this research. The knight's tour calculations were performed as "background computation" during a period of several weeks, on the computer at IDA-CRD in Princeton, New Jersey.

Computer Science Department
Stanford University
Stanford, California 94305

1. F. DE CARTEBLANCHE, "The colored cubes problem," *Eureka*, v. 9, 1947, pp. 9–11.

2. T. R. DAWSON, "Echecs Feeriques," problem 186, *L'Echiquier*, (2), v. 2, 1930, pp. 1085–1086; solution in *L'Echiquier*, (2), v. 3, 1931, p. 1150.

3. T. R. DAWSON, *Caissa's Wild Roses*, C. M. Fox, Surrey, England, 1935; reprinted in *Five Classics of Fairy Chess*, Dover, New York, 1973.

4. T. R. DAWSON, "Chess facts and figures," *Chess Pie III*, souvenir booklet of the International chess tournament, Nottingham, 1936, pp. 34–36.

5. PAUL J. GANS, "Self-avoiding random walks. I. Simple properties of intermediate-walks," *J. Chem. Phys.*, v. 42, 1965, pp. 4159–4163. MR **32** #3147.

6. SOLOMON W. GOLOMB & LEONARD D. BAUMERT, "Backtrack programming," *J. Assoc. Comput. Mach.*, v. 12, 1965, pp. 516–524. MR **33** #3783.

7. N. T. GRIDGEMAN, "The 23 colored cubes," *Math. Mag.*, v. 44, 1971, pp. 243–252.

8. MARSHALL HALL, JR. & D. E. KNUTH, "Combinatorial analysis and computers," *Amer. Math. Monthly*, v. 72, 1965, no. 2, part II, pp. 21–28. MR **30** #3030.

9. J. M. HAMMERSLEY & D. C. HANDSCOMB, *Monte Carlo Methods*, Methuen, London; Barnes & Noble, New York, 1965. MR **36** #6114.

10. J. M. HAMMERSLEY & K. W. MORTON, "Poor man's Monte Carlo," *J. Roy. Statist. Soc. Ser. B*, v. 16, 1954, pp. 23–28. MR **16**, 287.

11. DONALD E. KNUTH, "Uncrossed knight's tours," (letter to the editor), *J. Recreational Math.*, v. 2, 1969, pp. 155–157.

12. DONALD E. KNUTH, *Fundamental Algorithms*, The Art of Computer Programming, vol. 1, second edition Addison-Wesley, Reading, Mass., 1973.

13. DÉNES KÖNIG, *Theorie der endlichen und unendlichen Graphen*, Leipzig, 1936; reprint, Chelsea, New York, 1950.

14. E. L. LAWLER & D. E. WOOD, "Branch-and-bound methods: A survey," *Operations Research*, v. 14, 1966, pp. 699–719. MR **34** #2338.

15. DERRICK H. LEHMER, "Combinatorial problems with digital computers," *Proc. Fourth Canadian Math. Congress* (1957), Univ. of Toronto Press, Toronto, 1959, pp. 160–173. See also: *Proc. Sympos. Appl. Math.*, vol. 6, Amer. Math. Soc., Providence, R. I., 1956, pp. 115, 124, 201.

16. DERRICK H. LEHMER, "The machine tools of combinatorics," Chapter 1 in *Applied Combinatorial Mathematics*, edited by Edwin F. Beckenbach, Wiley, New York, 1964, pp. 5–31.

17. ÉDOUARD LUCAS, *Récréations Mathématiques*, v. 1, Paris, 1882.

18. MICHIO MATSUDA & S. KOBAYASHI, "Uncrossed knight's tours," (letter to the editor), *J. Recreational Math.*, v. 2, 1969, pp. 155–157.

19. T. H. O'BIERNE, *Puzzles and Paradoxes*, Oxford Univ. Press, New York and London, 1965.

20. C. B. TOMPKINS, review of "Random-number generating dice," by Japanese Standards Association, *Math. Comp.*, v. 15, 1961, pp. 94–95.

21. R. J. WALKER, "An enumerative technique for a class of combinatorial problems," *Proc. Sympos. Appl. Math.*, vol. 10, Amer. Math. Soc., Providence, R. I., 1960, pp. 91–94. MR **22** #12048.

22. MARK B. WELLS, *Elements of Combinatorial Computing*, Pergamon Press, Oxford and New York, 1971. MR 43 #2819.

23. L. D. YARBROUGH, "Uncrossed knight's tours," *J. Recreational Math.*, v. 1, 1968, pp. 140–142.