An Analysis of Consecutively Bounded Depth-First Search with Applications in Automated Deduction

Mark E. Stickel and W. Mabry Tyson

Artificial Intelligence Center
SRI International
Menlo Park, California 94025

Abstract

Consecutively bounded depth-first search involves repeatedly performing exhaustive depth-first search with increasing depth bounds of 1, 2, 3, and so on. The effect is similar to that of breadth-first search, but, instead of retaining the results at level n - 1 for use in computing level n, earlier results are recomputed. Consecutively bounded depth-first search is useful whenever a complete search strategy is needed and either it is desirable to minimize memory requirements or depth-first search can be implemented particularly efficiently. It is notably applicable to automated deduction, especially in logic-programming systems, such as PROLOG and EQLOG, and their extensions. Consecutively bounded depth-first search, unlike unbounded breadth-first search, can perform cutoffs by using heuristic estimates of the minimum number of steps remaining on a solution path. Even if the possibility of such cutoffs is disregarded, an analysis shows that, in general, consecutively bounded depth-first search requires only b/b-1 times as many operations as breadth-first search, where 6 is the branching factor.[1]

1 Introduction

In this paper, we investigate the properties of *consecutively bounded depth-first search.* In this method, exhaustive depth-first search is repeatedly performed with increasing depth bounds of 1, 2, 3, and so on. The effect is similar to that of breadth-first search, but, instead of retaining the results at level n — 1 for use in computing level n, earlier results are recomputed.[2]

Although this may appear to be a naive and costly search method, it is not necessarily so. It is sometimes advantageous to perform consecutively bounded depth-first search instead of the breadth-first search it imitates. One reason for this is that depth-first search requires much less memory.

Consecutively bounded depth-first search can also make use of heuristic information, in contrast to unbounded breadth- and depth-first search—the latter are uninformed search strategies that do not take into account heuristic estimates of the remaining distance to a solution. Informed search strategies such as the A* algorithm use such heuristic information to order the search space. Consecutively bounded depth-first search does not do that, but can use an estimate of the minimum number of remaining steps to a solution to perform cutoffs if the estimate exceeds the number of levels left before the depth bound is reached. If the number of remaining levels is uniformly exceeded by these estimates by more than one level, then one or more levels can be skipped when the next depth bound is set. As with the A* algorithm, admissability the guarantee of finding a shortest solution path first is preserved provided the heuristic estimate never exceeds the actual number of remaining steps to a solution

Another advantage of consecutively bounded depth-first search stems from the fact that, in some applications, depth-first search can be implemented with much higher efficiency than breadth-first search; consecutively bounded depth-first search combines this efficiency with the completeness of breadth-first search. In these applications, the greater efficiency of depth-first search more than compensates for the effort of recomputing earlier-level results in consecutively bounded depth-first search.

A specific instance of this is PROLOG-style automated deduction. PROLOGs use of depth-first search contributes significantly to its performance. If depth-first search were not used, more than one derived clause would have to be represented simultaneously and variables would have more than a single value simultaneously, i.e., different values in different clauses. This would imply the need for a more complex and less efficient representation for variable bindings than the one PROLOG currently uses.

One of our interests is in adapting PROLOG implementation technology to the design of high-performance general automated-deduction systems [6]. For general deduction, PROLOG'S depth-first search is incomplete and of limited utility. But to adopt breadth-first search would result in losing the efficiency advantages of PROLOG'S representation for variable bindings. Performing bounded depth-first search would preserve the depth-first character of the search while allowing exhaustive searching of the space to a specified level.

There is still the problem of selecting the depth bound. In an exponential search space, searching with a higher-than-necessary depth bound can waste an enormous amount of effort before the solution is found. This is because the cost of searching level *tt* in an exponential search space is generally large compared with the cost of searching earlier levels.

But this also makes it practical to perform consecutively bounded depth-first search. The depth bound is set successively at 1, 2, *'3,* etc., until a solution is found. If a uniform brandling factor 6 is assumed, this results in only about $b/b\text{-}1_t$ times as many operations as are necessary for breadth-first search to the same depth.

Another potential application in automated deduction and

'This research was supported by the Defenie Advanced Research Projects Agency under Contract N00039-84-K-0078 with the Naval Electronic Systems Command. The views and condusions contained in this document are those of the authors and should not be interpreted as representative of the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the United States government. Approved for public release. Distribution unlimited.

[2]We assume a basic familiarity with standard breadth-first, depth-first, and A* search strategies (e.g., see Nilsson [4]).

logic programming is in systems like EQLOG [I]. EQLOG extends PROLOG by replacing the standard unification algorithm with an algorithm based on narrowing that unifies terms in equationa) theories. Because the narrowing process is not necessarily finite, it may be necessary for completeness to interleave computation of unifiers by narrowing with the Horn-clause-resolution backtracking search. Here the use of consecutively bounded depth-first search would be beneficial both for its representational efficiency and for its low space consumption—the latter is particularly important because there may be a large number of unification attempts that are simultaneously active.

Consecutively bounded depth-first search is similar to the tree-searching strategy of *iterative deepening* used in chess [5]. In iterative deepening, search is repeatedly performed with increasing depth bounds until a time limit is reached. Insofar as these chess searches can be modeled by breadth-first search with a uniform branching factor, our analysis reveals that iterative deepening search in chess is only marginally more expensive than a single search to the maximum depth.

Despite this use of consecutively bounded depth-first search in chess and its obvious utility, it has surprisingly remained unanalyzed and unargued for—until now. Our proposal of a PROLOG technology theorem prover [6] included a description of this search strategy (which we implemented) and a very rough analysis on which this work builds. Korf [2,3] has independently come to similar conclusions on the value of this search strategy and has done his own analysis that emphasizes its asymptotic optimality in space and time among brute-force searches and, with the use of cutoffs, its optimality among admissable best-first searches.

2 Consecutively Bounded Depth-First Search Versus Breadth-First Search

We now present a comparison of the the effort required to perform consecutively bounded depth-first search and breadth-first search. The analysis ignores the effects of using heuristic information in consecutively bounded depth-first search, which would yield even more favorable results for that strategy.

Worst-case behavior for consecutively bounded depth-first search, as compared with breadth-first search, occurs when the branching factor is 1, i.e., when no searching is required. In this case, if the solution is found at depth n, the cost of breadth-first search with uniform branching factor 1 is $BFS_1(n) = n$, while the cost of consecutively bounded depth-first search is $CBDFS_1(n) = 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$.

For larger branching factors, consecutively bounded depth-first search will be shown to be the generally small constant factor $\frac{b}{b-1}$ times as expensive as breadth-first search (i.e., $\frac{1}{b-1}$ extra effort). This is true regardless of whether consecutively bounded depth-first search is used to search exhaustively to some depth or only until the first solution is found.

2.1 Exhaustive Search

The number of operations $BFS_b(n)$ performed in searching a space exhaustively with uniform branching factor b to depth n, using breadth-first search, is given by

$$BFS_b(n) = \sum_{i=1}^{n} b^i = \frac{b^{n+1}-1}{b-1} - 1 = \frac{b}{b-1}(b^n - 1). \quad (1)$$

We will also use the approximation

$$\frac{BFS_b(n)}{b^n} \approx \frac{b}{b-1}. \quad (2)$$

Because searching a space exhaustively with breadth-first or depth-first search differs only in the order of operations, the number of operations $DFS_b(n)$ for depth-first search is given by

$$DFS_b(n) = BFS_b(n) = \frac{b}{b-1}(b^n - 1). \quad (3)$$

The number of operations $CBDFS_b(n)$ performed in searching a space exhaustively with uniform branching factor b to depth n, using consecutively bounded depth-first search, i.e., searching exhaustively to depth $1, 2, \ldots, n$, is given by

$$\begin{aligned} CBDFS_b(n) &= \sum_{i=1}^{n} DFS_b(i) = \frac{b}{b-1}\sum_{i=1}^{n}(b^i - 1) \\ &= \frac{b}{b-1}(DFS_b(n) - n). \end{aligned} \quad (4)$$

Thus, the ratio of the costs of searching exhaustively to depth n, using consecutively bounded depth-first search, compared with searching to depth n by using breadth-first search can be approximated by

$$\frac{CBDFS_b(n)}{BFS_b(n)} = \frac{CBDFS_b(n)}{DFS_b(n)} \approx \frac{b}{b-1}. \quad (5)$$

2.2 Search to First Solution

Assume that the there are no solutions below level n and that the first solution is found at fraction r of the way through searching level n—e.g., $r = 0.5$ if half of the level n operations have been done when the first solution is found.

We now define the approximate number of operations $BFS_b(n, r)$ and $CBDFS_b(n, r)$, corresponding to $BFS_b(n)$ and $CBDFS_b(n)$, performed in searching a space to the first solution found at point r on level n:

$$BFS_b(n, r) = BFS_b(n) - (1 - r)b^n \quad (6)$$

$$CBDFS_b(n, r) = CBDFS_b(n) - (1 - r)DFS_b(n). \quad (7)$$

Thus, the ratio of the costs of searching to the first solution at point r on level n, using consecutively bounded depth-first search compared with using breadth-first search, can be approximated, by use of Equations (5) and (2), by

$$\begin{aligned} \frac{CBDFS_b(n, r)}{BFS_b(n, r)} &= \frac{CBDFS_b(n) - (1 - r)DFS_b(n)}{BFS_b(n) - (1 - r)b^n} \\ &\approx \frac{CBDFS_b(n) - (1 - r)(\frac{b-1}{b})CBDFS_b(n)}{BFS_b(n) - (1 - r)(\frac{b-1}{b})BFS_b(n)} \\ &= \frac{CBDFS_b(n)}{BFS_b(n)} \approx \frac{b}{b-1}. \end{aligned} \quad (8)$$

2.3 Memory Requirements

An advantage of consecutively bounded depth-first search is that it needs only an amount of memory that is linear in the depth of the tree. When the search routine is at some node in the tree, it needs to remember only the node's ancestors or, sometimes for convenience, the node's and ancestors' siblings as well. Even in the latter case, the memory required is only bn nodes where b is the branching factor and n is the depth. For ordinary breadth-first search, the memory required is $O(b^n)$, since each node in the tree at the current depth must be stored. Consequently, much less memory is needed for consecutively bounded depth-first search.

3 Evenly Bounded Depth-First Search

At first it appears that consecutively bounded depth-first search wastes effort by redoing the same steps too often. In fact only $\frac{1}{b}$ of the effort is wasted, but for small branching factors, when the waste is greatest, the amount of wasted effort can be decreased by modifying the strategy.

Consider using even bounds $(2, 4, \ldots)$ instead of consecutive bounds $(1, 2, \ldots)$ when searching for the first solution.

If the solution occurs on one of the bounding levels (i.e., at an even depth), evenly bounded depth-first search is a clear winner. But if the solution occurs on another level, it may be worse. Since we probably do not know the parity of the depth of the first solution any more than we know the depth itself, a comparison of evenly bounded with consecutively bounded depth-first search is difficult for any set of problems. One way to compare them would be to determine under what conditions the savings (when a solution is found at an even depth) outweigh the extra effort (when the solution could have been found at the preceding odd depth). For this analysis, we will assume that the solution that could have been found at the previous odd depth will still be the first solution found when searching to the next greater even depth. A contrary assumption would make evenly bounded depth-first search more favorable. We shall first compute the number of operations for evenly bounded depth-first search when the solution is found at even or odd depths.

Similarly to Equation (4),

$$EBDFS_b(2q) = \sum_{i=1}^{q} DFS_b(2i). \tag{9}$$

As a result of the even bounds,

$$EBDFS_b(2q-1) = EBDFS_b(2q). \tag{10}$$

Likewise, similarly to Equation (7),

$$EBDFS_b(2q, r) = EBDFS_b(2q-2) + rDFS_b(2q) \tag{11}$$

and again

$$EBDFS_b(2q-1, r) = EBDFS_b(2q, r). \tag{12}$$

For solutions at even levels, the difference between finding the first solution by consecutively bounded and evenly bounded depth-first search is

$$\begin{aligned} &CBDFS_b(2q, r) - EBDFS_b(2q, r) \\ &= \sum_{i=1}^{2q-1} DFS_b(i) - \sum_{i=1}^{q-1} DFS_b(2i) = \sum_{i=1}^{q} DFS_b(2i-1). \end{aligned} \tag{13}$$

For solutions at odd levels,

$$\begin{aligned} &CBDFS_b(2q-1, r) - EBDFS_b(2q-1, r) \\ &\quad = \sum_{i=1}^{2q-2} DFS_b(i) + rDFS_b(2q-1) - \\ &\qquad \left(\sum_{i=1}^{q-1} DFS_b(2i) + rDFS_b(2q) \right) \\ &\quad = \sum_{i=1}^{q-1} DFS_b(2i-1) + r\sum_{i=1}^{2q-1} b^i - r\sum_{i=1}^{2q} b^i \\ &\quad = \sum_{i=1}^{q-1} DFS_b(2i-1) - rb^{2q}. \end{aligned} \tag{14}$$

If the sum of the differences is positive, then the added efficiency in dealing with problems whose first solution is on an even level outweighs any extra overhead expended upon problems whose first solution is on an odd level. If that is so, evenly bounded depth-first search is clearly more efficient than consecutively bounded depth-first search. Analysis reveals that evenly bounded depth-first search is always preferable for a branching factor of 2 (or less), while consecutively bounded depth-first search is preferable for a branching factor of 4 or more For a branching factor of .3, the advantages of finding solutions on even levels are approximately equal to the disadvantages of finding solutions on odd levels.

4 Conclusion

We have analyzed the behavior of consecutively bounded depth-first search. This strategy is useful whenever a complete search strategy is needed, and either it is desirable to minimize memory requirements or depth-first search can be implemented particularly efficiently. Moreover, consecutively bounded depth-first search, in contrast to the unbounded breadth-first search it almost emulates, can take advantage of heuristic estimates of the minimum number of steps remaining on a solution path to perform cutoffs if that number exceeds the number of levels left before the depth bound is reached. Even if the possibility of such cutoffs is disregarded, we have found the performance penalty resulting from the use of consecutively bounded depth-first search to be small when compared with breadth-first search the former performs only b/b1 times as many operations as the latter, where *b* is the branching factor.

References

[l] Goguen, J. and .1. Mcscgucr. Equality, types and generics for logic programming. *Proceedings of the 1984 Logic Programming Symposium,* Uppsala, Sweden, 1984, 115 125.

[2] Korf, R.E. Depth-first iterative-deepening: an optimal admissable tree search. To appear in *Artificial Intelligence Journal.*

[3] Korf, HE. Iterative-deepening-A*: an optimal admissable tree search. *Proceeding* of the Ninth International Joint Conference on Artificial Intelligence,* Los Angeles, (California, August 1985.

[4] Nilsson, N..I. *Principles of Artificial Intelligence.* Tioga Publishing Co., Palo Alto, California, 1980.

[5] Slate, D..I. and L.R. Atkin. CHESS 15 The Northwestern University chess program. In Frey, P.W. (ed), *Chess Skill in Man and Machine,* Springer-Verlag, New York, New York, 1977, 82- 118.

[6] Stickel, ME. A PROLOG technology theorem prover. *Proceedings of the 1984 International Symposium on Logic Programming,* Atlantic City, New Jersey, February 1984, 211 217. Revised version appeared in *New Generation Computing 2, 4* (1984), 371-383.