Iterative Broadening

Matthew L. Ginsberg
William D. Harvey

Computer Science Department
Stanford University
Stanford, California 94305
ginsberg@cs.stanford.edu

Iterative Broadening

Abstract

Conventional blind search techniques generally assume that the goal nodes for a given problem are distributed randomly along the fringe of the search tree. We argue that this is often invalid in practice and suggest that a more reasonable assumption is that decisions made at each point in the search carry equal weight. We go on to show that a new search technique called *iterative broadening* leads to orders-of-magnitude savings in the time needed to search a space satisfying this assumption; the basic idea is to search the space using artificial breadth cutoffs that are gradually increased until a goal is found. Both theoretical and experimental results are presented.

1 Introduction

Imagine that we are searching a tree of uniform depth d using conventional depth-first search. We work our way to the fringe of the tree and check to see if we are at a goal node. Assuming that we are not, we back up a minimal amount, generate a new node at depth d, and check to see if *that* is a goal node. This process continues until we finally succeed in solving the problem at hand.

The process of depth-first search generates the fringe nodes (the nodes at depth d) in a particular order – from left to right, if we were to draw the tree in its entirety. Assuming that the goal nodes (of which there may be more than one) are randomly placed along the fringe of the search tree, this left-to-right order is a reasonable way to search the tree.

Now imagine that we are trying to solve some hard problem, say buying a birthday present for a friend. We decide to buy her a book on Tennessee walking horses, and visit a few book and tack stores looking for one but without success.

At this point, rather than continue looking for this particular gift, *we may well decide to try something else.* Even if we have good reason to believe that a book of the kind we want exists, we view our continued failure as an indication that we would be better off looking for a different gift, and end up buying her a saddle pad.

If all of the inference steps we might take while solving the problem were equally likely to lead to a solution, this would make no sense. But in practice, we view the fact that we have been unable to solve the problem of finding the Tennessee-walker book as evidence that the whole *idea* of getting the book is misguided. Does this idea have an analog in search generally?

It does. The reason is that it is possible to make a mistake. Going back to our tree of depth d, it is quite possible that one of the nodes at depth 1 (for example) has committed

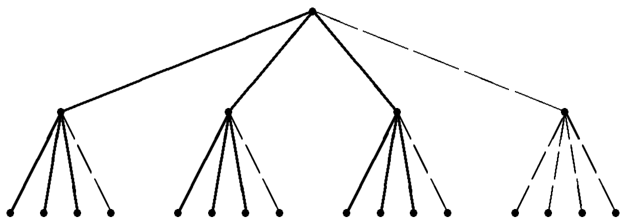

Figure 1: Search with a breadth cutoff

us to a course from which we will need to backtrack because this particular node has no goal node underneath it.

The way we deal with this in practice is by imposing an artificial breadth limit on our search. Thus we may try three different book stores before giving up and getting our friend something else for her birthday. What we are proposing in this paper is that blind search techniques do the same thing.

Specifically, we will suggest that depth-first search be augmented with an artificial breadth cutoff c. What this means is that when we have had to backtrack to a particular node n in the tree c times, we continue to backtrack to the previous choice point, *even if there are still unexplored children of the node n.* An example is depicted in Figure 1, where we have shown a tree with breadth 4 but a breadth cutoff of 3; the dashed lines indicate paths that have been pruned as a result.

Of course, the search with an artificial breadth cutoff will never search the entire tree; we need some way to recover if we fail to find an answer in our search to depth d. We will suggest that the most practical way to deal with this problem is to gradually increase the breadth cutoff. So we first start by searching the tree with an artificial breadth cutoff of 2, then try with a breadth cutoff of 3, and so on, until an answer is found. We will refer to this technique as "iterative broadening," borrowing terminology from Korf's notion of iterative *deepening* [6].

We will show in this paper that given reasonable assumptions about the distribution of the goal nodes along the fringe, this technique leads to tremendous expected savings in the amount of time needed to search the tree. The reason, roughly speaking, is that the searches with limited breadth retain a reasonable chance of finding a goal node while pruning large fractions of the complete space and enabling us to backtrack past early mistakes more effectively. Even if the preliminary searches fail to find a goal node, the time spent on them is often sufficiently small that overall performance is not much affected.

The outline of this paper is as follows: In the next section, we present a mathematical analysis of the technique, computing the utility of a search with breadth cutoff c in a space of branching factor b. In Section 3, we discuss the ratio of the time needed by our approach to that needed by the conventional one for various choices of b and d, and for various densities of goal nodes at the fringe of the tree. We present both theoretical values (which incorporate some approximations made in Section 2) and experimental results.

In Section 4, we show that the inclusion of heuristic information makes our method still more powerful, and also that it remains viable when used in combination with other search techniques (dependency-directed backtracking, etc.). Related work, limitations and extensions to our approach are discussed in Section 5.

2 Formal analysis

Suppose that our tree, of uniform branching factor b and depth d, has g successful goal nodes distributed along the fringe. What we will do is assume that decisions made at each point in the search are equally important – in other words, that there is some constant s such that if n is a node under which there is at least one goal node, then n has exactly s successful children. (We are calling a node successful if it has at least one goal node somewhere under it.) The fact that s is independent of the depth of n is the formal analog of our claim that decisions made at different depths are equally important. We assume that the s successful children are randomly distributed among the b children of n.

Since the root node is successful, it is not hard to see that there are s^d goal nodes in the tree as a whole, so that we must have $s = g^{1/d}$. In what follows, we will generally assume that s is an integer (so that we can evaluate the expressions that follow), and that $s > 1$ (so that there is more than a single goal node on the fringe).

Now suppose that we fix b, s and a breadth cutoff c. For a given depth d, we will denote the probability that the restricted-breadth search finds at least one goal node by $p(d)$, and the number of nodes examined by this search by $t(d)$. We begin by computing $\overline{p}(d) = 1 - p(d)$. What is the probability that the restricted search *fails* to find a goal node at depth d?

For $d = 0$ (the root node), we clearly have $\overline{p}(0) = 0$. For larger d, suppose that we denote by $\mathtt{pr}(i)$ the probability that of the s successful children of a node n, exactly i of them are included in the restricted search of breadth c. Now the probability that the search to depth $d + 1$ fails is given by

$$\overline{p}(d+1) = \sum_i \mathtt{pr}(i)\overline{p}(d)^i \tag{1}$$

since in order for the search to fail, the depth d search under each of the i successful children of the root node must also fail. We sum over the various i and weight the contributions by the probability that the i successful nodes at depth 1 all actually fail.

We can evaluate $\mathtt{pr}(i)$ by noting that in order for there to be exactly i successful nodes in the restricted search, we much choose i successful nodes from the s that are available, and also $c - i$ unsuccessful nodes from the $b - s$ available. Since the total number of ways to choose the c nodes in the restricted search from the b present in the entire tree is given by $\binom{b}{c}$, it is not hard to see that

$$\mathtt{pr}(i) = \frac{\binom{s}{i}\binom{b-s}{c-i}}{\binom{b}{c}}$$

This expression, together with (1), allows us to compute $p(d)$ for various d.

What about $t(d)$, the number of nodes examined by the restricted search to depth d? We begin by defining $t_s(d)$ to be the number of nodes examined on average by the restricted search to depth d *provided that the restricted search is successful.* We immediately have

$$t(d) = [1 - \overline{p}(d)]t_s(d) + \overline{p}(d)\left(\frac{c^{d+1} - 1}{c - 1}\right) \tag{2}$$

since the complete tree of breadth c and depth d must be searched if the restricted search fails.

We also need $\mathtt{pr}_s(i)$, the probability that exactly i of the depth 1 nodes are successful given that the restricted search succeeds. Bayes' rule gives us

$$\mathtt{pr}_s(i) = \mathtt{pr}(i|s) = \frac{\mathtt{pr}(s|i)\mathtt{pr}(i)}{\mathtt{pr}(s)} = \frac{[1 - \overline{p}(d-1)^i]\mathtt{pr}(i)}{1 - \overline{p}(d)}$$

To complete the calculation, we need to know that if we have c nodes at depth 1, of which i are successful, then the average number of these c nodes we need to examine before finding j of the successful ones is given by

$$\frac{j(c+1)}{i+1}$$

Given that the restricted search is successful, how many of the nodes do we need to examine before finding one that is successful with the breadth cutoff c? If i of the nodes at depth 1 are successful, and each successful node has a probability of failure given by $\overline{p}$, then the number of the c nodes we can expect to examine is given by

$$f(i, \overline{p}) = \sum_{j=0}^{i-1} \frac{(j+1)(c+1)}{i+1} \frac{\overline{p}^j(1 - \overline{p})}{1 - \overline{p}^i} \tag{3}$$

The rationale for this expression is as follows: Each term represents the number of nodes examined assuming that the first j successful nodes fail (which happens with probability $\overline{p}$ for each) and the $j + 1$st succeeds. The terms are weighted by $1/(1 - \overline{p}^i)$ because $\overline{p}^i$ is the probability that none of the i nodes succeeds, and this is eliminated by our assumption that the restricted search is successful.

Given this result, the number of failing nodes examined at depth 1 is $f(i, \overline{p}) - 1$ and $t_s(d+1)$ is therefore:

$$1 + \left[\sum_i \mathtt{pr}_s(i) f(i, \overline{p}(d)) - 1\right]\left(\frac{c^{d+1} - 1}{c - 1}\right) + t_s(d) \tag{4}$$

The first term corresponds to the fact that the root node is always examined; the final term $t_s(d)$ is the number of nodes examined below the depth 1 node that eventually succeeds. The summation is, as usual, over the number of successful children at depth 1; for each value of i, we compute the expected number of failing nodes examined at depth 1 and realize that each of these failing nodes leads to a complete search of breadth c and depth d.

Using (1) and the expression for $t_s(d+1)$ in (4), we can easily compute the probability of success and expected time taken for the first pass through the search tree, with breadth cutoff $c = 2$.

Unfortunately, we cannot use these expressions to evaluate the probabilities and time taken for subsequent searches with $c = 3$ and higher. The reason is that when we do the search with $c = 3$, we know that the search with $c = 2$ has already failed. Assuming that we do not reorder the children of the various nodes when we iterate and search the tree again, we need to take the fact that the $c = 2$ search has failed into account when analyzing $c = 3$.

For $f < c$, let us denote by $\overline{p}(c, f)$ the probability that the search with cutoff c fails *given that the search with cutoff f is known to fail.* Now it is obvious that

$$\overline{p}(c) = \overline{p}(c, f)\overline{p}(f)$$

so that we immediately have

$$\overline{p}(c, f) = \frac{\overline{p}(c)}{\overline{p}(f)} \tag{5}$$

We also need to do something similar for t. We will assume (wrongly) that the expected number of nodes examined during a *successful* search to depth d is unchanged by the fact that the search with breadth cutoff f has failed. In practice, it is possible to reorder the nodes so that the expected number is less than this. However, this may not be desirable because it will abandon heuristic information present in the original order; it is because of these competing reasons that we are taking the number to be unchanged. As in (2), this leads to

$$t(c, f) = [1 - \overline{p}(c, f)]t_s(d) + \overline{p}(c, f)\left(\frac{c^{d+1} - 1}{c - 1}\right).$$

The hard work is now done. The expected time taken to solve the problem using iterative broadening is given by

$$\sum_i \overline{p}(i-1)t(i, i-1) \tag{6}$$

where the term being summed corresponds to the time taken by the search of breadth i, given that the previous search has failed to solve the problem.

3 Results

3.1 Theoretical

Given the results of the previous section, it is straightforward to compute the expected time taken by an iterative-broadening search and compare it to the expected time taken by a simple depth-first search (i.e., $c = b$). The tables in Figures 2 and 3 do this for $s = 2$ and $s = 3$ respectively and for various choices of b (branching factor) and d (depth of tree). The numbers appearing in the table are the ratios of these two times and indicate the factor saved by the new approach. (Thus the 15.5 appearing in Figure 3 for $b = 15$ and $d = 7$ indicates

branching factor	depth 4	7	11	15
4	1.9	5.9	19.8	59.9
6	1.7	5.8	20.2	55.2
9	1.4	4.9	17.4	46.7
12	1.2	4.1	14.6	38.7
15	1.0	3.5	12.4	32.6

Figure 2: Performance improvement for $s = 2$

branching factor	depth 4	7	11	15
4	2.9	26.2	451.3	7276.6
6	3.9	30.7	292.3	1990.3
9	3.4	24.3	186.0	928.1
12	2.9	19.0	129.0	559.9
15	2.4	15.5	97.3	390.2

Figure 3: Performance improvement for $s = 3$

that iterative broadening will typically solve the problem 15.5 times faster than depth-first search.)

It can be seen from these figures that iterative broadening outperforms conventional search for $s \geq 2$ and $d \geq 4$. (For shallower searches, this was not always the case. The worst case examined was $s = 2, b = 15, d = 2$, when iterative broadening was 0.4 times as fast as depth-first search.) Furthermore, as the depth of search increases, so does the time saved. For large depths and branching factors, it appears that the iterative broadening technique reduces the time needed to solve the problem by an overall factor approximately linear in b and additionally reduces the effective branching factor by $s - 1$. This reduction in effective branching factor makes sense, since the search with breadth cutoff $s - 1$ is guaranteed to find a goal node at some point, since this breadth cutoff will never prune all of the goal nodes from the search space..

Note that if s were known in advance, we could achieve the same effect by working with a breadth cutoff of $b - s + 1$ directly. Iterative broadening achieves this savings without advance information about the size of s. Additional savings are also obtained because of the possible success of the searches with narrower breadth cutoffs.

3.1.1 The large depth limit

In the large d limit, it is possible to use our analysis to determine general conditions under which iterative broadening leads to computational speedups.

Suppose that we denote by $t_{df}(d)$ the time needed by conventional depth-first search to depth d and by $t(d)$ the time needed by the penultimate iteration of iterative broadening. We also denote by $p(d)$ the probability that this penultimate iteration succeeds in finding a goal node. It is clear by induction that if the technique of searching the graph with breadth cutoff $b-1$ before attempting the full depth-search technique leads to a performance improvement, then iterative broadening will as well. Of course, the time needed by this simple two-iteration approach is given by

$$t(d) + \overline{p}(d) t_{df}(d)$$

since we always incur the cost of the search with breadth cutoff $b - 1$ and incur the cost of the full search when the restricted search fails. We therefore need to find conditions under which

$$t(d) + \overline{p}(d) t_{df}(d) < t_{df}(d)$$

or

$$t(d) < p(d) t_{df}(d) \tag{7}$$

As we will see, iterative broadening leads to performance improvements for very small values of s, so that we will take $s = 1 + \epsilon$ for some small ϵ. To make things a bit more precise, we will suppose that a successful node in the tree has probability $1 - \epsilon$ of having a single successful child and probability ϵ of having two successful children, so that $1 + \epsilon$ successful children are expected on average.

We begin by using (4) to find the expected time needed by conventional depth-first search for large d. Note that we cannot use existing results here because of our unconventional assumption regarding distribution of the goal nodes.

With a full-width search, the cutoff c is equal to b and the search below any successful node is guaranteed to succeed, so that $\overline{p} = 0$ in (3). It follows that the only term contributing to the sum in (3) is $j = 0$, and

$$f(i, \overline{p}) = \frac{b+1}{i+1}$$

When we insert this in (4), the only terms that will contribute are $i = 1$ and $i = 2$, since we are assuming that a successful node has at most two successful children. This gives us

$$\begin{aligned}
t_{df}(d+1) &= 1 + [(1-\epsilon)f(1,\overline{p}) + \epsilon f(2,\overline{p}) - 1]\left(\frac{b^{d+1}-1}{b-1}\right) + t_{df}(d) \\
&= 1 + \left[(b+1)\left(\frac{1-\epsilon}{2} + \frac{\epsilon}{3}\right) - 1\right]\left(\frac{b^{d+1}-1}{b-1}\right) + t_{df}(d) \\
&\approx \left(\frac{b-1}{2}\right)\left(\frac{b^{d+1}-1}{b-1}\right) + t_{df}(d) \\
&\approx \frac{b^{d+1}}{2} + t_{df}(d)
\end{aligned}$$

Solving for $t_{df}(d)$ recursively gives us

$$t_{df}(d) \approx \frac{b^{d+1}}{2(b-1)} \tag{8}$$

which is the same amount of time needed to search the tree if only a single solution exists. In some sense, this should not be a surprise – the multiple solutions will be clustered because of our assumptions regarding their distribution, and it will be little easier for depth-first search to find one of them than to find them all.

What about the next-to-last iteration of iterative broadening? We need to compute both its probability of finding a goal node, $p(d)$ and the amount of time taken, $t(d)$. This latter expression is clearly bounded by the size of the entire search space with breadth cutoff $b-1$, which is given by

$$t(d) < \frac{(b-1)^{d+1}}{b-2} \tag{9}$$

For the probability of success, let n be a successful node. Then if n has only one successful child (probability $1-\epsilon$), the probability that the restricted search finds it is $\frac{b-1}{b}$, since $b-1$ of the b children will be examined. The probability that the search fails to examine the child is $\frac{1}{b}$.

If n has two successful children (probability ϵ), then at least one will be found by the search with breadth cutoff $b-1$. The probability that they will both be found is

$$\frac{(b-1)(b-2)}{b(b-1)} = \frac{b-2}{b}$$

since we need to choose the two successful nodes from among the $b-1$ that are examined. The probability that only one is examined is therefore $\frac{2}{b}$.

Combining these observations gives us:

$$\begin{aligned}
\mathtt{pr}(0) &= \frac{1-\epsilon}{b} \\
\mathtt{pr}(1) &= \frac{b-1}{b}(1-\epsilon) + \frac{2\epsilon}{b} = \frac{b - b\epsilon + 3\epsilon - 1}{b} \\
\mathtt{pr}(2) &= \frac{(b-2)\epsilon}{b}
\end{aligned}$$

Inserting this into (1) gives us:

$$\begin{aligned}
\overline{p}(d+1) &= \frac{1-\epsilon}{b} + \frac{b - b\epsilon + 3\epsilon - 1}{b}\overline{p}(d) + \frac{(b-2)\epsilon}{b}\overline{p}(d)^2 \\
&\approx \frac{1-\epsilon}{b} + \frac{b - b\epsilon + 3\epsilon - 1}{b}\overline{p}(d) + \frac{2b\epsilon - 4\epsilon}{b}\overline{p}(d) + \frac{2\epsilon - b\epsilon}{b} \\
&= \frac{1+\epsilon-b\epsilon}{b} + \frac{b-1+b\epsilon-\epsilon}{b}\overline{p}(d) \\
&= \frac{1-(b-1)\epsilon}{b} + \frac{(b-1)(1+\epsilon)}{b}\overline{p}(d)
\end{aligned}$$

where we have used the fact that $\overline{p} \approx 1$ to conclude that $\overline{p}^2 \approx 2\overline{p} - 1$. Suppose that we now set

$$x = \frac{1-(b-1)\epsilon}{b}$$

and

$$y = \frac{(b-1)(1+\epsilon)}{b}$$

Solving for $\overline{p}(d)$ recursively gives us

$$\overline{p}(d) = \frac{x}{1-y} - \frac{xy^{d+1}}{1-y}$$

We also have $x = 1-y$, so that $\frac{x}{1-y} = 1$ and the above expression simplifies to

$$\overline{p}(d) = 1 - \left[\frac{(b-1)(1+\epsilon)}{b}\right]^{d+1}$$

or

$$p(d) = \left[\frac{(b-1)(1+\epsilon)}{b}\right]^{d+1} \tag{10}$$

Inserting (8), (9) and (10) into (7) gives us

$$\left[\frac{(b-1)(1+\epsilon)}{b}\right]^{d+1} \frac{b^{d+1}}{2(b-1)} > \frac{(b-1)^{d+1}}{b-2}$$

which simplifies to

$$\frac{(1+\epsilon)^{d+1}}{2(b-1)} > \frac{1}{b-2} \tag{11}$$

But now note that $(1+\epsilon)^d$ is simply g, the number of goal nodes at the fringe. The inequality (11) therefore becomes

$$g > \frac{2(b-1)}{b-2} \tag{12}$$

Theorem 3.1 *In the large depth limit, iterative broadening will lead to computational savings whenever the number of goal nodes at the fringe exceeds*

$$\frac{2(b-1)}{b-2}$$

The expression appearing in Theorem 3.1 is typically quite small (three goal nodes suffice if $b > 3$) and we do not need for there to be more goal nodes as the depth increases.

3.1.2 Finding multiple solutions

The observation made in deriving (8) has interesting consequences if more than a single solution is needed for some search problem. The reason is that as we have seen, these solutions will tend to be clustered along the fringe of the search space.

It follows that if we are searching for multiple solutions, the most effective approach is probably to find a single solution quickly, and then to search outward from there. As we have seen, iterative broadening is the technique of choice for finding this single solution.

3.1.3 Optimal protocols

Armed with the theoretical results that we have presented, there is another approach that we might have taken. Instead of searching our space first with breadth cutoff 2, then 3 and so on, we could compute theoretically what sequence of breadth cutoffs would minimize the time needed to find a solution to the problem.

We have done this; we are not presenting detailed results here because they are of little interest. Roughly speaking, the optimal protocols differ only very slightly – both in terms of sequence of breadth cutoffs and in terms of computation time needed – from the simple protocol of increasing the breadth cutoff by 1 at each iteration.

3.2 Experimental

In order to ensure that the approximations made in Section 2 do not invalidate our theoretical results, we compared the iterative-broadening approach to conventional depth-first search on randomly generated problems. The experimental results appear in Figure 4 and are in overall agreement with our theoretical expectations.

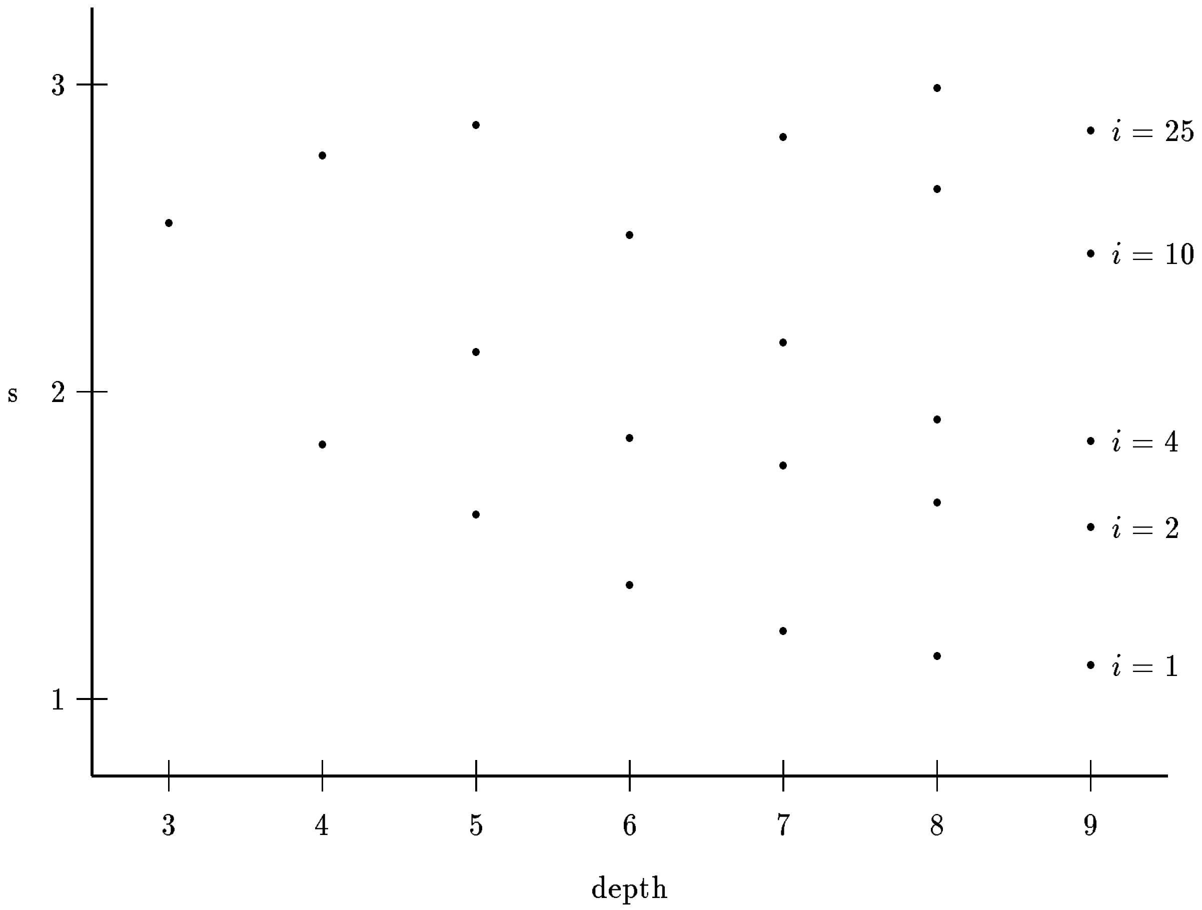

Figure 4: Performance improvement: blind search

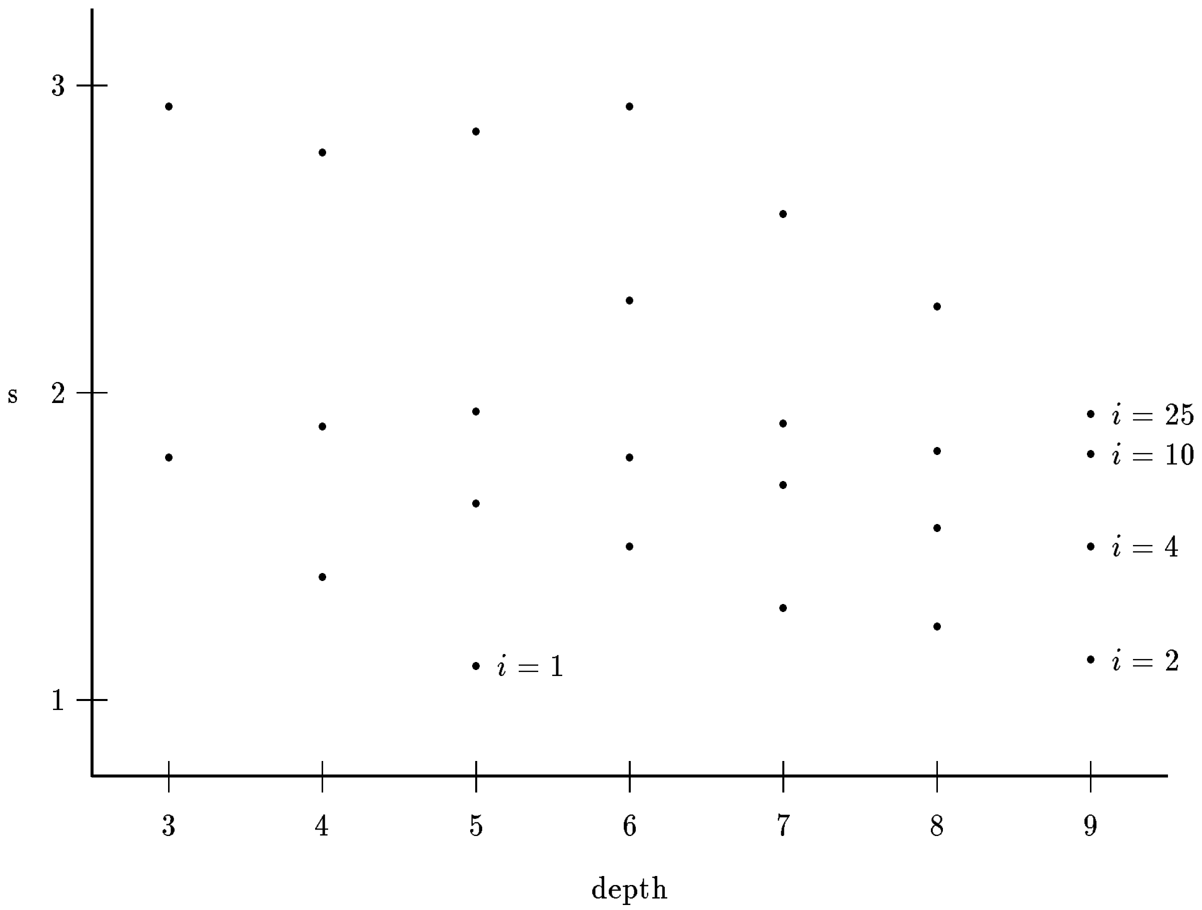

Figure 5: Performance improvement: heuristic search

This figure displays contour graphs where the contours reflect constant factor speedups for various values of s and d. The speedup observed was roughly independent of branching factor, and the points in the graph correspond to average performance improvements for 1000 trials for each branching factor from 5 to 8 inclusive. As an example, the data in Figure 4 show that for search to depth 7, the time needed to find a solution was halved using iterative broadening at $s = 1.76$, and the breakeven between the two methods occurred at $s = 1.22$. The experimental data includes nonintegral values for s since in practical applications a successful node may not have a fixed number of successful children.[1]

The case $s = 1$ is exceptional because it leads to exactly one goal node. Since this goal node is randomly located on the fringe of the search tree, the fastest search is the one that examines the fringe most rapidly, and we can therefore expect depth-first search to outperform iterative broadening in this case. At worst, iterative broadening might be a factor of b slower (since there are b possible iterations); in practice, we observed only a factor of 2.

[1] As in Section 3.1.1, nonintegral values of s were handled probabilistically.

4 Heuristic information

The experimental work described in Section 3 was extended to consider cases in which heuristic information was used to order the children of any particular node so that subnodes that are likely to lead to solutions are examined first. We simulated a fairly weak heuristic that scaled by a factor of $2(b+1-i)/(b+1)$ the probability that the ith child of a successful node was itself successful. Thus the probability of the first child's success is approximately doubled and the probabilities for subsequent children are scaled by a linearly decreasing amount. The results are shown in Figure 5 and indicate that heuristic information improves the relative performance of iterative broadening with respect to standard search techniques. (The same contours are shown in the two figures, but they are displaced down and to the left in Figure 5.) In fact, for $b > 5$ and $d > 6$, iterative broadening is the technique of choice even if $s = 1$.

These results can be understood by realizing that at any node, both search techniques examine the most likely children first. The consequences of making a mistake are different, however – if depth-first search examines an unsuccessful node near the top of the tree, it will go on to examine the entire subtree of that node at potentially devastating cost. Iterative broadening limits the fruitless search to a subtree of breadth c; the better the heuristic, the more likely it is that a goal will be encountered for small c. Even for the relatively weak heuristic used in our experiments, large relative performance improvements were obtained.

Combination with other techniques Finally, we tested the iterative-broadening technique by including it in a program designed to generate crossword puzzles by filling words into an empty frame [4]. This program uses a variety of techniques, including a heuristic ranking of the children of the node being examined, directional arc-consistency [1], backjumping (a simple form of dependency-directed backtracking) [3] and dynamic search rearrangement.

The results were as expected. We ran the program on 500 randomly-generated 5×5 crossword puzzles with an average of 2.5 blocked-out squares in each; the search was limited to 1000 nodes expanded per puzzle. Iterative broadening solved 497 puzzles expanding an average of 22.1 nodes per successful attempt; depth-first search solved 498 puzzles expanding an average of 42.8 nodes on each. Even in these relatively small search spaces, iterative broadening only expands approximately half as many nodes as conventional methods.

5 Related work, limitations and extensions

5.1 Related work

Rao and Kumar have recently suggested [7] that the most effective way to search a tree of the sort we have been examining is to interleave n parallel searches for the first n children of the root node, and have shown that this technique reduces expected search time if the distribution of the goal nodes is nonuniform along the fringe of the tree.

This approach works for exactly the same reason as iterative broadening – the cost of making a mistake is minimized. Iterative broadening can be expected to lead to still further improvements, since parallel search cannot address difficulties arising from mistakes made below the first level of the search tree or from damage done if the first n children are *all* failing nodes.

5.2 Limitations

In the problems that we have examined, there are always multiple solutions that appear at a fixed depth in the search space. It is interesting to consider the question of how likely this assumption is to be justified for the "prototypical" search problems typically considered by other AI researchers, and whether iterative broadening can be applied to these other problems.

The answer is typically negative, although the reasons vary.

Sliding tile puzzles Experiments with sliding tile puzzles show that iterative broadening does not lead to performance improvements in this domain. The reason is that the branching factor is approximately two, and it is effectively impossible to apply the approach in this case.

Rubik's cube Applying the method to the $2 \times 2 \times 2$ Rubik's cube also led to negative results. The reason is that there is typically only a single optimal path to the goal for this problem. Our approach in this case was to combine iterative broadening with iterative deepening [6]: We first searched the tree to depth 1, then to depth 2, and so on. At each iteration, we used iterative broadening as opposed to conventional depth-first search in our attempt to find a solution. As we have remarked, however, there is typically a unique shortest solution in this problem; when we used iterative broadening to find it, we were searching a space that contained only a single goal node.

Adversary search Finally, we considered the possibility of using iterative broadening in combination with α-β search to speed the examination of game trees. Our hope was that the early iterations would allow us to order the search in subsequent iterations, and that the restricted-breadth ordering would be more useful in pruning the subsequent search space than the restricted-depth search that is typically used when combining α-β search with iterative deepening.

Once again, we were disappointed. The reason this time was simply that the search performed by modern game-playing programs is already quite near optimal. Modern chess-playing programs, for example, typically examine only some 150% of the optimal number of nodes when looking for the best move [2].

The conclusion to be drawn from these negative results is not, of course, that the technique is without merit – although AI's classic chestnuts do not exhibit the features needed to make iterative broadening useful, other problems (such as constraint-satisfaction problems admitting multiple solutions) can be expected to.

5.3 Extensions

The argument that we presented in defense of iterative broadening in the introduction was simply that we expected decisions made at various points in the search space to be equally important in contributing to our attempt to find a solution, but there is a more fundamental feature at work here as well.

The reason iterative broadening works is that it takes failure of a node to be an indication that *siblings* of that node are also likely to fail. In the example of the introduction, we took the fact that three separate stores did not have a book on Tennessee walking horses to be evidence that other stores wouldn't, either.

In search where the solutions are randomly distributed along the fringe, the assumption that nodes affect their siblings in this way is obviously incorrect. In many practical problems, however, especially where heuristic information is being used, the assumption is likely to be valid because it is possible to make a mistake without realizing that you've done so.

It is natural to try to extend iterative broadening by exploiting this observation directly, using the failure of siblings to adjust the heuristic value associated to any particular node in a search space and then using an existing heuristic technique such as A^* to conduct the search using the modified heuristic. We are currently exploring this possibility.

6 Conclusion

Iterative broadening leads to computational speedups on search problems containing in excess of 40,000 nodes if either $s > 1.5$ (so that a successful node has, on average, at least 1.5 successful children) or there is heuristic information available indicating conditions under which a node is likely to lead to a solution. In the large depth limit, speedups can be expected whenever there are three or more solutions to the problem in question.

The speedups gained by our approach are often several orders of magnitude; this easily outweighs the cost, which appears to be at most a factor of 2. These theoretical results are confirmed by experimentation on both randomly generated small problems and the large toy problem of crossword puzzle generation.

Acknowledgement

This paper is an expanded version of [5]. The work has been supported by General Dynamics and by NSF under grant number DCR-8620059. We would like to thank Murray Campbell, Bob Floyd, Rich Korf and Mark Torrance for various helpful suggestions.

References

[1] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–38, 1988.

[2] C. Ebeling. *All the Right Moves: A VLSI Architechture for Chess.* MIT Press, Cambridge, MA, 1986.

[3] J. Gaschnig. Performance measurement and analysis of certain search algorithms. Technical Report CMU-CS-79-124, Carnegie-Mellon University, 1979.

[4] M. L. Ginsberg, M. Frank, M. P. Halpin, and M. C. Torrance. Search lessons learned from crossword puzzles. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 210–215, 1990.

[5] M. L. Ginsberg and W. D. Harvey. Iterative broadening. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 216–220, 1990.

[6] R. E. Korf. Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[7] V. Nageshwara Rao and V. Kumar. Superlinear speedup in state-space search. In *Proceedings of the 1988 Foundation of Software Technology and Theoretcal Computer Science*, December 1988. Lecture Notes in Computer Science #338, Springer Verlag.