Depth-bounded Discrepancy Search

Toby Walsh*
APES Group, Department of Computer Science
University of Strathclyde, Glasgow G1 1XL. Scotland
tw@cs.strath.ac.uk

Abstract

Many search trees are impractically large to explore exhaustively. Recently, techniques like limited discrepancy search have been proposed for improving the chance of finding a goal in a limited amount of search. Depth-bounded discrepancy search offers such a hope. The motivation behind depth-bounded discrepancy search is that branching heuristics are more likely to be wrong at the top of the tree than at the bottom. We therefore combine one of the best features of limited discrepancy search – the ability to undo early mistakes – with the completeness of iterative deepening search. We show theoretically and experimentally that this novel combination outperforms existing techniques.

1 Introduction

On backtracking, depth-first search explores decisions made against the branching heuristic (or "discrepancies"), starting with decisions made deep in the search tree. However, branching heuristics are more likely to be wrong at the top of the tree than at the bottom. We would like therefore to direct search to discrepancies against the heuristic high in the tree – precisely the opposite of depth-first search. To achieve this aim, depth-bounded discrepancy search (or DDS) combines together limited discrepancy search [Harvey and Ginsberg, 1995] and iterative deepening search [Korf, 1985]. DDS appears to perform better than both limited discrepancy search, LDS and the improved version, ILDS [Korf, 1996]. Unlike ILDS, DDS does not need an upper limit on the maximum depth of the tree (that is, the maximum number of branching points down any branch). This will be an advantage in many domains as we often only have a loose limit on the maximum depth because of constraint propagation and pruning.

2 Discrepancies

A discrepancy is any decision point in a search tree where we go against the heuristic. For convenience, we assume that left branches follow the heuristic. Any other branch breaks the heuristic and is a discrepancy. For convenience, we call this a right branch. Limited discrepancy search (LDS) explores the leaf nodes in increasing order of the number of discrepancies taken to reach them. On the ith iteration, LDS visits all leaf nodes with up to i discrepancies. The motivation is that our branching heuristic has hopefully made only a few mistakes, and LDS allows a small number of mistakes to be corrected at little cost. By comparison, depth-first search (DFS) has to explore a significant fraction of the tree before it undoes an early mistake. Korf has proposed an improved

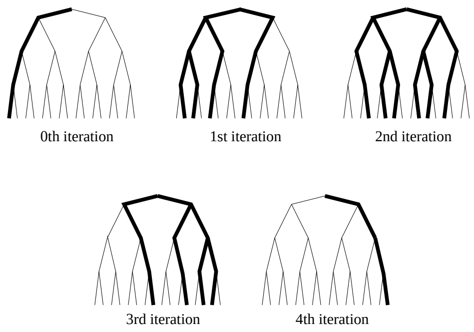

Figure 1: ILDS on a binary tree of depth 4.

version of LDS called ILDS that uses (an upper limit on) the maximum depth of the tree. On the ith iteration, ILDS visits leaf nodes at the depth limit with exactly i discrepancies. This avoids re-visiting leaf nodes at the depth limit with fewer discrepancies. When the tree is not balanced or the limit on the maximum depth is loose, ILDS re-visits all those leaf nodes above the limit.

Both LDS and ILDS treat all discrepancies alike, irrespective of their depth. However, heuristics tend to be less informed and make more mistakes at the top of the search tree. For instance, the Karmarkar-Karp heuristic for number partitioning [Korf, 1996] makes a fixed (and possibly incorrect) decision at the root of the tree. Near to the bottom of the tree, when there are 4 or less numbers left to partition, the heuristic always makes the optimal choice. Given a limited amount of search, it may pay to explore discrepancies at the top of the tree before those at the bottom. This is the motivation behind depth-bounded discrepancy search.

*The author is supported by EPSRC award GR/K/65706. I wish to thank members of the APES group for their help.

3 Depth-bounded discrepancy search

Depth-bounded discrepancy search (DDS) biases search to discrepancies high in the tree by means of an iteratively increasing depth bound. Discrepancies below this bound are prohibited. On the 0th iteration, DDS explores the leftmost branch. On the $i+1$th iteration, DDS explores those branches on which discrepancies occur at a depth of i or less. As with ILDS, we are careful not to re-visit leaf nodes visited on earlier iterations. This is surprisingly easy to enforce. At depth i on the $i+1$th iteration, we take only right branches since left branches would take us to leaf nodes visited on an earlier iteration. At lesser depths, we can take both left and right branches. And at greater depths, we always follow the heuristic left. Search is terminated when the increasing bound is greater than the depth of the deepest leaf node. As the $i+1$th iteration of DDS on a balanced tree with branching factor b explores $O(b^i)$ branches, the cost of each iteration grows by approximately a constant factor.

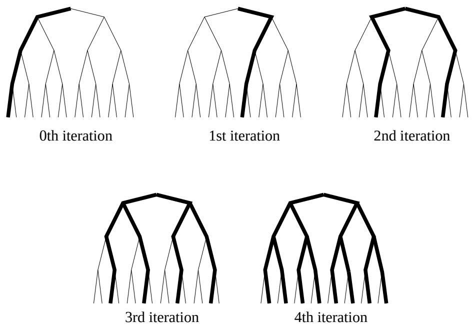

Figure 2: DDS on a binary tree of depth 4.

As an example, consider a binary tree of depth 4. On the 0th iteration, we branch left with the heuristic and visit the leftmost leaf node. On the 1st iteration, we branch right at the root and then follow the heuristic left. On the 2nd iteration, we branch either way at the root, just right at depth 1, and left thereafter. Branching left at depth 1 re-visits leaf nodes visited on earlier iterations. On the 3rd iteration, we branch either way at the root and depth 1, just right at depth 2, then left thereafter. On the 4th and final iteration, we branch either way up to and including depth 2, and just right at depth 3. This takes us to the remaining leaf nodes and completes the search of the tree. Note that we did not re-visit any leaf nodes. The following pseudo-code describes DDS for binary trees. Note that, unlike ILDS, the maximum depth does not need to be given as it is *computed* during search as the second argument returned by PROBE.

function DDS
  $k = 0$
  repeat
    $\langle goal, depth\rangle$ = PROBE($root$,k)
    $k = k + 1$
  until $goal$ **or** $k > depth$
  return $goal$

function PROBE($node$,k)
  if leaf($node$) **then return** $\langle$goal-p($node$), 0$\rangle$
  if $k = 0$ **then**
    $\langle goal, depth\rangle$ = PROBE($left(node)$,0)
    return $\langle goal, 1 + depth\rangle$
  if $k = 1$ **then**
    $\langle goal, depth\rangle$ = PROBE($right(node)$,0)
    return $\langle goal, 1 + depth\rangle$
  if $k > 1$ **then**
    $\langle goal_1, depth_1\rangle$ = PROBE($left(node)$,$k-1$)
    if $goal_1$ **then return** $\langle goal_1, 1 + depth_1\rangle$ **else**
      $\langle goal_2, depth_2\rangle$ = PROBE($right(node)$,$k-1$)
      return $\langle goal_2, 1 + \max(depth_1, depth_2)\rangle$

DDS visits all the leaf nodes of a search tree, but never re-visits leaf nodes at the maximum depth. To prove this, consider the depth j of the deepest right branch on the path to any leaf node at the maximum depth. Only the jth iteration of DDS will visit this leaf node. DDS re-visits leaf nodes above the maximum depth of the tree when the depth bound k exceeds the depth of the shallowest leaf node. This typically occurs late in search when much of the tree has been explored. Indeed, as search is usually limited to the first few iterations, we often do not re-visit any leaf nodes. ILDS also re-visits leaf nodes that occur above (the upper limit on) the maximum depth of the tree. However, ILDS can re-visit leaf nodes from the first iteration onwards. For example, if the leftmost leaf node is above the maximum depth of the tree, it is re-visited on every iteration.

Given a limited amount of search, DDS explores more discrepancies at the top of search tree than LDS or ILDS. At the end of the first iteration of ILDS on a binary tree of height d, we have explored branches with at most one discrepancy. By comparison, DDS is already on approximately its $\log_2(d)$-th iteration, exploring branches with up to $\log_2(d)$ discrepancies. And at the end of the second iteration of ILDS, we have explored branches with at most two discrepancies. DDS is now on approximately its $2\log_2(d)$-th iteration, exploring branches with up to $2\log_2(d)$ discrepancies.

DDS is a close cousin of iterative-deepening search [Korf, 1985]. There are, however, two significant differences. First, when the depth bound is reached, we do not immediately backtrack but follow the heuristic to a leaf node. Second, by always going right immediately above the depth bound, we avoid re-visiting any nodes at the depth bound.

4 Asymptotic complexity

For simplicity, we consider just balanced binary trees of depth d. The analysis would generalize with little difficulty to trees with larger branching rates. Korf has shown that in searching the tree completely, LDS visits $O(\frac{d+2}{2}2^d)$ leaf nodes [Korf, 1996]. By comparison, DDS, DFS and ILDS all visit the 2^d leaf nodes just once. ILDS and DDS are less efficient than DFS as they re-visit interior nodes multiple times. However, they have a similar overhead. DFS visits $2^{d+1}-1$ nodes in searching the tree exhaustively. By comparison, ILDS visits approximately 2^{d+2} nodes [Korf, 1996], as does DDS. Let DDS(d) be the number of nodes visited by DDS in searching a tree of depth d completely. On the 0th iteration, DDS explores the d nodes below the root ending at the leaf node on the leftmost branch. On the ith iteration, DDS explores completely a subtree of depth $i-1$. In addition, DDS branches right at the 2^{i-1} nodes at depth $i-1$, and then left down the $(d-i)$ nodes to a leaf node. Thus, for d large,

$$\begin{aligned}
\text{DDS}(d) &\approx d + \sum_{i=1}^{d} (2^i + 2^{i-1}(1+d-i)) \\
&= d + \sum_{i=1}^{d} 2^{i-1}(3+d) - \sum_{i=1}^{d} i2^{i-1} \\
&= d + (2^d - 1)(3+d) - 2^d\,(d-1) - 1 \\
&= 4\,2^d - 4 \\
&\approx 2^{d+2}
\end{aligned}$$

Expanding only right nodes at the depth bound contributes significantly to this result. On the last and most expensive iteration, the depth bound reaches the bottom of the tree, yet DDS explores just the 2^{d-1} out of the 2^d leaf nodes that have not yet been visited. Without this efficiency, DDS would explore approximately 2^{d+3} nodes.

5 A Small Improvement

By increasing the space complexity of DDS, we can avoid most of the interior node overhead. For instance, we could search to the depth bound using breadth-first search instead of iterative-deepening as present. We can then start successive iterations from the frontier at the depth bound, saving the need to re-visit all the interior nodes above the depth boumd. The only overhead now is that of re-visiting the approximately 2^d interior nodes that lie along the left paths followed beneath the depth bound. The dominating cost is that of breadth-first search which visits approximately 2^{d+1} nodes. Unfortunately, this improvement increases the space complexity at the kth iteration from $O(k)$ to $O(2^k)$. Provided search is limited to the first few iterations, this may be acceptable and offer improvements in runtimes. We can achieve lesser savings but at linear cost in the space complexity using recursive best-first search with the depth as the cost function [Korf, 1993]. This will perform breadth-first search using space that is only of size $O(k)$ but will re-visit some interior nodes. We might also consider searching up to depth bound using ILDS. Such a strategy would not explore the tree completely. For example, it would not explore the branch which goes left at the root, right at depth 1, and left thereafter. For completeness, we would need to search to the depth bound using LDS. But this search strategy would re-visit many more nodes than necessary.

6 A Simple Theoretical Model

To analyse iterative broadening and limited discrepancy search theoretically, Harvey and Ginsberg introduce a simple model of heuristic search with a fixed probability of a branching mistake [Ginsberg and Harvey, 1992; Harvey and Ginsberg, 1995]. Nodes in a search tree are labelled "good" and "bad". A good node has a goal beneath it. A bad node does not. The mistake probability, m is the probability that a randomly selected child of a good node is bad. This is assumed to be constant across the search tree. See [Harvey, 1995] for some experimental justification of this assumption.

The heuristic probability, p is the probability that at a good node, the heuristic choses a good child first. If the heuristic choses a bad child, then since the node is good, the other child must be good. If the heuristic makes choices at random then $p = 1 - m$. If the heuristic does better than random then $p > 1 - m$. And if the heuristic always makes the wrong choice, $p = 1 - 2m$. Figure 3, adapted slightly from Figure 3.3 in [Harvey, 1995], lists what can happen when we branch at a good node, and gives the probabilities with which each situation occurs.

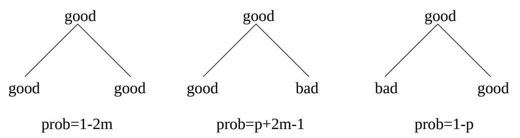

Figure 3: Branching at a good node. As the situations are disjoint and exhaustive, the probabilities sum to 1.

In order to simplify the analysis, Harvey and Ginsberg assume that p is constant throughout the search tree, though they note that it tends to increase with depth in practice. At the top of the tree, heuristics are less informed and have to guess more than towards the bottom of the tree. This simplifying assumption will bias results against DDS. Harvey and Ginsberg also restrict their analysis to the first iteration of LDS. The combinatorics involved in computing exactly the probability of success of later iterations are very complex and have defeated analysis. To study later iterations, we simply *estimate* probabilities by searching an ensemble of trees constructed to have a fixed depth and a given heuristic and mistake probability. Since search will be limited to the first few iterations of the algorithms, the nodes in the trees can be generated lazily on demand. Trees with billions of nodes can easily be analysed by this method.

As in Figure 4 of [Harvey and Ginsberg, 1995], Figure 4 gives the probability of finding a goal when searching trees of height 30 with a mistake probability of 0.2 computed from an ensemble of 100,000 trees. Such trees have approximately a billion leaf nodes, of which just over a million are goals. With about 1 in 1000 leaf nodes being goals, such problems are relatively easy. Nevertheless, the probability of success for DFS rises only slightly above p^d, the probability that the first branch ends in a goal. As in [Harvey and Ginsberg, 1995], DFS is given the highest heuristic probability in this and all subsequent experiments but still performs poorly. ILDS, which is omitted from the graphs for clarity, performs very similarly to LDS. On these small and easy problems, DDS offers a small advantage over LDS at each value of p. The discontinuities in the gradients of the graphs for DDS correspond to increases in the depth bound. They do not seem to disappear with even larger samples.

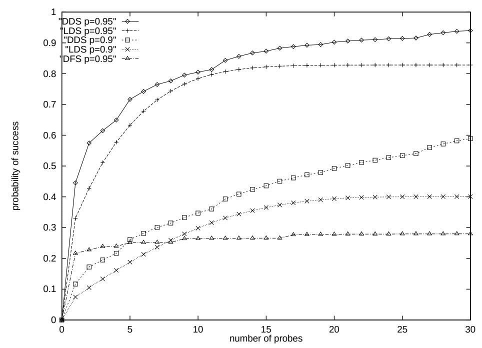

Figure 4: Probability of success on trees of height 30 and mistake probability, $m = 0.2$.

We next consider more difficult search problems. As in Figure 5 of [Harvey and Ginsberg, 1995], Figure 5 gives the probability of finding a goal when searching trees of height 100 with a mistake probability of 0.1 computed from an ensemble of 10,000 trees. Such trees have approximately 10^{30} leaf nodes, of which about 1 in 38,000 are goals. This graph covers the 0th, 1st and part of the 2nd iteration of LDS. Previously, Harvey and Ginsberg restricted their theoretical analysis to the 1st iteration of LDS as it is intractable to compute the probabilities exactly for later iterations. As predicted in [Harvey and Ginsberg, 1995], at the start of the 2nd iteration, the probability of success for LDS "rise[s] steadily once again" as we explore fresh paths at the top of the search tree. However, DDS appears to be more effective at exploring these paths and offers performance advantages over LDS at every value of heuristic probability.

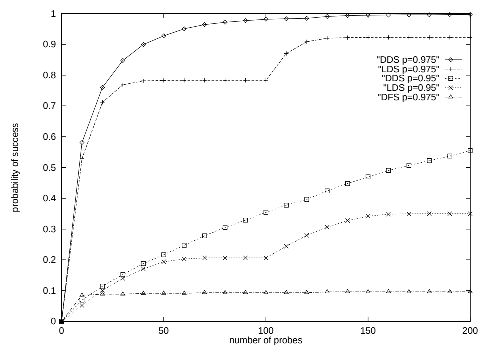

Figure 5: Probability of success on trees of height 100 and mistake probability, $m = 0.1$.

Finally, we vary the heuristic probability with depth. In Figure 6, p varies linearly from $1 - m$ at depth 0 (i.e. random choice) to 1 at depth d (i.e. always correct). This may provide a better model for the behavior of heuristics in practice. DDS appears to offer greater performance advantages over LDS in such situations.

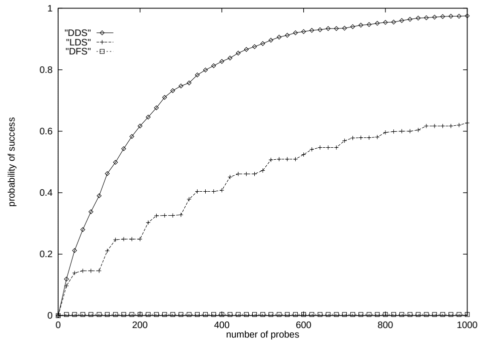

Figure 6: Probability of success on trees of height 100 and $m = 0.1$ with heuristic probability, p varying linearly from $1 - m$ (random choice) at the root to 1 (correct choice) at leaf nodes. At depth i, $p = (1 - m) + im/100$.

7 Early mistakes

We attribute the success of DDS to the ease with which it can undo early mistakes. Such mistakes can be very costly for DFS. For instance, Crawford and Baker identified early mistakes as the cause of poor performance for a Davis-Putnam procedure using DFS on Sadeh's scheduling benchmarks [Crawford and Baker, 1994]. On some problems, a solution was found almost immediately. On other problems, an initial bad guess would lead to searching a subtree with the order of 2^{70} nodes. Recent phase transition experiments have also identified early mistakes as a cause of occasional exceptionally hard problems (or ehps) in under-constrained regions. For example, Gent and Walsh found ehps in soluble propositional satisfiability problems [Gent and Walsh, 1994]. They report that ehps often occur after an early mistake when the heuristic branches into an insoluble and difficult subproblem. Similarly, Smith and Grant found ehps in soluble constraint satisfaction problems with sparse constraint graphs [Smith and Grant, 1995]. Again they report that they often occur after an early branching mistake into an insoluble and difficult subproblem.

DDS may undo early mistake at much less cost than DFS. To test this hypothesis, we generated satisfiable problems from the constant probability model using the same parameters as [Gent and Walsh, 1994]. Each problem has 200 variables and a ratio of clauses to variables of 2.6. Literals are included in a clause with probability 3/400, giving a mean clause length of approximately 3. Unit and empty clauses are discarded as they tend to make problems easier. This problem class is typically much easier than the random 3-SAT model but generates a greater frequency of ehps [Gent and Walsh, 1994]. To solve these problems, we use the Davis-Putnam procedure, branching on the first literal in the shortest clause. Table 1 demonstrates that DDS reduces the severity of ehps compared to DFS. Other methods to tackle ehps include sophisticated backtracking procedures like conflict-directed backjumping [Smith and Grant, 1995] and learning techniques like dependency-directed backtracking [Bayardo and Schrag, 1996] which identify quickly the insoluble subproblems after an early mistake.

	DDS	DFS
mean	7.54	$> 49,000$
50%	1	1
90%	5	2
99%	87	19
99.9%	859	761,095
99.99%	7,581	$> 200,000,000$

Table 1: Mean and percentiles in branches explored by the Davis-Putnam procedure on 10,000 satisfiable problems from the constant probability model with 200 variables. At the time of submission, DFS had searched over 200,000,000 branches on 2 problems without success. We hope to report the final result at the conference.

8 Adding bounded backtracking

Whilst early mistakes are not necessarily costly for DDS, mistakes deep in the tree will be very costly. Bounded backtrack search (BBS) allows quick recovery from mistakes deep in the tree for little additional cost [Harvey, 1995]. A backtrack bound of depth l adds at most a factor of 2^l to the search cost. For small l, this is likely to be cheaper than the cost of additional iterations of DDS. An alternative view is to consider bounded backtracking as strengthening the branching heuristic to avoid choices which fail using a fixed lookahead.

DDS combines very naturally with BBS to give the algorithm DDS-BBS. DDS explores early mistakes made at the top of the tree whilst BBS identifies failures rapidly at the bottom of the tree. The two search strategies therefore join together without overlap. Harvey has also combined LDS with BBS to give LDS-BBS. See the Appendix for details and for a correction of a small error in the specifications of the algorithm given in [Harvey, 1995; Harvey and Ginsberg, 1995]. On the ith iteration, LDS-BBS re-visits all those leaf nodes with less than i discrepancies. One improvement is to combine ILDS with BBS so that the discrepancy search does not re-visit leaf nodes. However, even with such a modification, bounded backtracking still re-visits leaf nodes visited on earlier iterations. By comparison, the bounded backtracking in DDS-BBS only visits leaf nodes with a greater discrepancy count that have not yet been visited. The following pseudo-code describes DDS-BBS for binary trees.

```
function DDS-BBS(l)
  k = 0
      repeat
      ⟨goal, depth⟩ = PROBE(root,k,l)
      k = k + 1
      until goal or k + l > depth
  return goal

function PROBE(node,k,l)
  if leaf(node) then return ⟨goal-p(node),0⟩
  if k = 0 then
      ⟨goal_1, depth_1⟩ = PROBE(left(node),0,l)
      if goal_1 or depth_1 ≥ l then return ⟨goal_1, 1 + depth_1⟩
          else ⟨goal_2, depth_2⟩ = PROBE(right(node),0,l)
          return ⟨goal_2, 1 + max(depth_1, depth_2)⟩
  if k = 1 then
      ⟨goal, depth⟩ = PROBE(right(node),0,l)
      return ⟨goal, 1 + depth⟩
  if k > 1 then
      ⟨goal_1, depth_1⟩ = PROBE(left(node),k − 1,l)
      if goal_1 then return ⟨goal_1, 1 + depth_1⟩ else
          ⟨goal_2, depth_2⟩ = PROBE(right(node),k − 1,l)
          return ⟨goal_2, 1 + max(depth_1, depth_2)⟩
```

9 Experiments

A phase transition is often seen when we vary the constrainedness of randomly generated problems and they go from being under-constrained and soluble to over-constrained and insoluble [Cheeseman *et al.*, 1991]. Discrepancy search strategies like DDS, LDS and ILDS are designed for under-constrained and soluble problems, where we expect to explore just a small fraction of the search tree. Harvey and Ginsberg report that such problems are common in areas like planning and scheduling [Harvey and Ginsberg, 1995]. Discrepancy search strategies offer no advantage on insoluble problems where the tree must be traversed completely. Indeed, for a balanced binary tree, our asymptotic results, along with those of [Korf, 1996], show that we explore approximately double the number of nodes of DFS. When the tree is unbalanced, as a result of constraint propagation and pruning, the overhead can be even greater. We therefore restricted the experiments in this section to soluble and under-constrained problems, on which discrepancy search strategies can offer advantages.

As in [Harvey, 1995], we use the random 3-SAT model. A phase transition occurs for this model at a clause to variable ratio, L/N of approximately 4.3 [Mitchell *et al.*, 1992]. We generated 10,000 satisfiable problems from the soluble region with $L/N = 3.5$ and from 50 to 250 variables. Each problem is solved using the Davis-Putnam procedure, branching as before on the first literal in the shortest clause. Results are given in Table 2. As expected, on such under-constrained and soluble problems, DDS and ILDS are superior to DFS, with DDS offering an advantage over ILDS especially on larger and harder problems. By comparison, on "critically constrained" problems at $L/N = 4.3$, DFS gave the best performance, with little to chose between ILDS and DDS.

N	DFS mean	DFS 99.9%	ILDS mean	ILDS 99.9%	DDS mean	DDS 99.9%
50	14.40	262	10.81	450	10.65	520
100	116.36	3,844	28.33	822	24.87	710
150	641.50	29,224	57.37	1,425	48.85	1,011
200	3,795.54	201,863	114.37	3,621	99.07	2,403
250	21,816.87	1,422,539	239.49	10,104	198.30	6,081

Table 2: Branches explored by the Davis-Putnam procedure on 10,000 satisfiable random 3-SAT problems at $L/N = 3.5$. Best results in each row are underlined.

10 Related work

LDS was proposed by Harvey and Ginsberg in [Harvey, 1995; Harvey and Ginsberg, 1995]. They showed that it outperformed existing strategies like DFS and iterative sampling on large, under-constrained scheduling problems. ILDS, Korf's improved version of LDS appeared in [Korf, 1996]. Ari Jonsson (personal communication reported on page 94 of [Harvey, 1995]) has proposed a related search strategy to DDS in which we perform iterative-deepening search followed by bounded backtracking from nodes at the depth bound. Like LDS, such a strategy re-visits leaf nodes multiple times. For example, the leftmost leaf node is visited on every iteration of the search. By comparison, DDS visits only half the nodes at the depth bound, and never re-visits any leaf node at the maximum search depth.

11 Conclusions

For many problems, the search tree is too large to explore completely. As a consequence, we often want to increase the chance of finding a solution in a limited amount of search. We have shown both theoretically and experimentally that depth-bounded discrepancy search is an effective means of exploring large and under-constrained search trees. By focusing on branching decisions at the top of the search tree, where heuristics are most likely to be wrong, depth-bounded discrepancy search outperforms depth-first and limited discrepancy search.

References

[Bayardo and Schrag, 1996] R. Bayardo and R. Schrag. Using CSP look-back techniques to solve exceptionally hard SAT instances. In *Proceedings of CP-96*, 1996.

[Cheeseman *et al.*, 1991] P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the really hard problems are. In *Proceedings of the 12th IJCAI*, pages 331–337, 1991.

[Crawford and Baker, 1994] J.M. Crawford and A.B. Baker. Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems. In *Proceedings of the 12th AAAI*, 1092–1097, 1994.

[Gent and Walsh, 1994] I.P. Gent and T. Walsh. Easy problems are sometimes hard. *Artificial Intelligence*, 335–345, 1994.

[Ginsberg and Harvey, 1992] M. L. Ginsberg and W. D. Harvey. Iterative broadening. *Artificial Intelligence*, 55(2-3):367–383, 1992.

[Harvey and Ginsberg, 1995] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of 14th IJCAI*, 1995.

[Harvey, 1995] W. D. Harvey. *Nonsystematic Backtracking Search.* PhD thesis, Stanford University, 1995.

[Korf, 1985] R. Korf. Depth-first iterative deepening: an optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[Korf, 1993] R. Korf. Linear-space best-first search. *Artificial Intelligence*, 62(1):43–78, 1993.

[Korf, 1996] R. Korf. Improved limited discrepancy search. In *Proceedings of 13th AAAI.* 1996.

[Mitchell *et al.*, 1992] D. Mitchell, B. Selman, and H. Levesque. Hard and Easy Distributions of SAT Problems. In *Proceedings of 10th AAAI*, 459–465, 1992.

[Smith and Grant, 1995] B.M. Smith and S. Grant. Sparse Constraint Graphs and Exceptionally Hard Problems. In *Proceedings of 14th IJCAI*, 1995.

Appendix

LDS-BBS is specified by the following pseudo-code:

function LDS-BBS(l)
  for $k = 0$ **to** max depth
    $\langle goal, height \rangle$ = PROBE($root$,k,l)
    if $goal \neq$ NIL **return** $goal$
  return NIL

function PROBE($node$,k,l)
  if goal-p($node$) **then return** $\langle node, 0 \rangle$
  $s = children(node)$
  if $k > 0$ **then** $s = reverse(s)$
  $i = 0$, $count = 0$, $maxheight = 0$
  for $child$ in s
    if $k = 0$ **and** $count \geq 1$ **then break**
    if $k > 0$ **and** $i = 0$ **then** $k' = k - 1$ **else** $k' = k$
    $\langle goal, height \rangle$ = PROBE($child, k', l$)
    $maxheight = \max(maxheight, 1 + height)$
    if $goal \neq$NIL **then return** $\langle goal, 0 \rangle$
    $i = i + 1$
    if $height \geq l$ **then** $count = count + 1$
  return $\langle$ NIL, $maxheight \rangle$

There is a small error in the specifications that appear in [Harvey, 1995; Harvey and Ginsberg, 1995] as the 7th line of PROBE was "if $k \geq 0$ and $i = 0$ then $k' = k - 1$ else $k' = k$". This error allows k' can be set to -1. This prevents bounded backtracking from terminating in the previous line when the bound, l is reached. As a consequence, many more nodes are searched than intended. For example, the 0th iteration explores completely the subtree to the left of the root, irrespective of the backtracking bound, b.