Heuristic Search in Bounded-depth Trees: Best-Leaf-First Search

Wheeler Ruml
ruml@eecs.harvard.edu

Technical Report TR-01-02
Harvard University

January 30, 2002

Abstract

Many combinatorial optimization and constraint satisfaction problems can be formulated as a search for the best leaf in a tree of bounded depth. When exhaustive enumeration is infeasible, a rational strategy visits leaves in increasing order of predicted cost. Previous systematic algorithms for this setting follow a predetermined search order, making strong implicit assumptions about predicted cost and using problem-specific information inefficiently. We introduce a framework, *best-leaf-first search* (BLFS), that employs an explicit model of leaf cost. BLFS is complete and visits leaves in an order that efficiently approximates increasing predicted cost. Different algorithms can be derived by incorporating different sources of information into the cost model. We show how previous algorithms are special cases of BLFS. We also demonstrate how BLFS can derive a problem-specific model during the search itself. Empirical results on latin square completion, binary CSPs, and number partitioning problems suggest that, even with simple cost models, BLFS yields competitive or superior performance and is more robust than previous methods. BLFS can be seen as a model-based extension of iterative-deepening A*, and thus it unifies search for combinatorial optimization and constraint satisfaction with traditional AI heuristic search for shortest-path problems.

1 Introduction

In a combinatorial optimization problem, one must choose one of a discrete number of possible values for each problem variable in such a way that the problem-specific objective function is minimized. Often, this task is formulated as a tree search in which one selects a variable at each node and branches on its possible values. The goal then is to find the leaf with the lowest cost. Because these trees grow exponentially with problem size, complete enumeration of the leaves is often infeasible, and one attempts to visit the best leaves one can in the time available. One can think of constraint satisfaction problems in a similar way, with the goal being to find a leaf that violates zero constraints. Because an optimal leaf is recognizable in this case, visiting one early in the search can save an enormous amount of time.

Several different tree search procedures have been proposed for this setting. Depth-first search has the advantage of minimal overhead, generating each internal node in the tree no more than once. Often, a heuristic scoring function is used to rank the children of a node, and the search visits the preferred child first. If always choosing the preferred child does not yield an optimal solution, depth-first search revisits the decisions lowest in the tree first. This may be a poor strategy if the node ordering function can be inaccurate at the top of the tree. In limited discrepancy search (Harvey and Ginsberg, 1995; Korf, 1996), decisions at all levels of the tree are revisited quickly. A decision at which the top-ranked child node is not selected is called a discrepancy. Limited discrepancy search explores all paths with k discrepancies before any with $k+1$ discrepancies. Other algorithms, such as depth-bounded discrepancy search (Walsh, 1997) and iterative broadening (Ginsberg and Harvey, 1992), use still different search orders.

Each of these algorithms can be viewed as making strong assumptions regarding the expected costs of different leaves (Ruml, 2001a). Depth-first search, for instance, is a rational choice only if one assumes that the cost of every leaf whose path includes a discrepancy at the root is greater than the cost of the worst leaf that does not. Equivalently, the penalty for taking a discrepancy at a given depth is assumed to be greater than the cost of taking discrepancies at all deeper depths. Limited discrepancy search, on the other hand, assumes that discrepancies cost the same at all depths. Unfortunately, it is not always clear in advance which search order is best, and those faced with a new search problem are reduced to running many pilot experiments.

In this paper, we explore the advantages of representing the search algorithm's assumptions explicitly, in the form of a cost model. Because the search uses an explicit model, it can depend on parameters which are estimated on-line from the search tree itself, rather than assumed beforehand. It is also easy to arrange for it to take advantage of information such as heuristic child scores. We introduce a framework, *best-leaf-first search* (BLFS), for systematic search using such a model. The central idea is to visit leaves in an order that approximates increasing predicted cost. This is achieved by visiting all leaves whose predicted cost falls within a fixed bound, and then iteratively raising the bound.

After outlining BLFS, we will see two instantiations of the framework. The first, *indecision search*, uses a cost model that depends on the heuristic scores along the path to a leaf. We evaluate the algorithm's performance on both latin square completion and random binary CSPs. The second instantiation uses a cost model whose parameters are learned from the leaf costs observed during the search. We evaluate this algorithm on two different formulations of the combinatorial optimization problem of number partitioning. The results from both algorithms suggest that BLFS provides competitive or superior performance and is more robust than existing algorithms. We will conclude by showing how BLFS is analogous to the iterative-deepening A* (IDA*) algorithm for shortest-path problems (Korf, 1985). Their common framework of single-agent rationality provides a clean unification of search for combinatorial optimization and constraint satisfaction with the tradition of heuristic search in artificial intelligence for shortest-path problems.

2 Best-Leaf-First Search

The basic structure of BLFS is a simple loop in which we carry out successive depth-first searches. Each search visits all leaves whose cost is predicted to fall within a cost bound. Because the predicted costs are generated by a known model, we can choose bounds that are expected to cause twice as many nodes to be visited as on the previous iteration. Pseudo-code is shown in Figure 1. In order to implement this scheme, we must equip BLFS with a model of leaf costs that can support two different operations. The first is to predict, given a search node, the lowest cost of any leaf below it. This function is used to guide the search within each iteration of BLFS, allowing the algorithm to avoid descending into any subtree that does not contain a leaf we wish to visit on this iteration (step 9). The second operation the model must support is to predict, given a cost bound, the number of nodes that would be visited if that bound were used. This can be used to help find an appropriate cost bound for each iteration (step 5). In the

```
**BLFS-outer**
1. Visit a few leaves
2. *Nodes-desired* ← number of nodes visited so far
3. Loop until time runs out:
4.     Double *nodes-desired*
5.     Estimate cost bound that visits *nodes-desired* nodes
6.     Visit all leaves within cost bound (see BLFS-inner)

**BLFS-inner**(*node*, *bound*)
7. If leaf(*node*), visit(*node*)
8. else, for each *child* of *node*:
9.     If best-completion(*child*) $\leq$ *bound*
10.        BLFS-inner(*child*, *bound*)
```

Figure 1: Simplified pseudo-code for best-leaf-first search.

experiments reported below, a simple bisection search was performed over possible cost bounds, searching for one that yielded, according to the model, approximately the desired number of nodes. Any prediction within 5% of the desired value or greater by less than 50% was deemed sufficient. The search was limited to 10 bisections. To guard against inaccurate predictions, an iteration was terminated after visiting three times the desired number of nodes.

By approximately doubling the number of nodes visited on each iteration, BLFS limits its overhead in the worst-case situation in which the entire tree must be searched. In such a situation, depth-first search is optimal. If we assume that the full tree has n nodes, the worst case for BLFS is when its second-to-last iteration visits $n-1$ nodes. In such a case, all previous iterations visit a combined total of roughly n nodes, and thus BLFS visits roughly $3n$ nodes, only three times more than optimal. In practice, the main overhead in the algorithm has been cost bound estimation, which consumes up to 40% of the search time for short runs in our prototype implementation. This overhead can be nearly eliminated, however, by augmenting the bisection search with memory and interpolation.

The basic BLFS framework is remarkably simple. In the remainder of this paper, we will demonstrate its power and flexibility by instantiating the framework using two different cost models. The first model we will consider is a static one that is specified before the search begins.

3 BLFS with a Fixed Model: Indecision Search

As mentioned earlier, heuristic child ranking is often used to guide tree search. Most such ranking functions actually return a numerical score for each child, although the precise semantics of this score varies. The first cost model we will consider is based on these scores. We will assign each child a cost based on how much worse its score is than the score of the best child. If the child scores are $s_0, \ldots, s_b$, child i has a cost of $s_i - s_0$. Furthermore, we will assume that the cost of a leaf is simply the maximum of the costs of the nodes along its path from the root. This is a generalization of iterative broadening, which assumes that the cost of a leaf reflects the maximum rank of any child along its path. Another way to think about this cost model is that the cost of a node reflects how decisively the heuristic score separated it from the best child. Paths involving nodes for which the heuristic was indecisive in its ranking will be explored earlier in the search than those involving nodes whose scores were much worse. For this reason, we can call this instantiation of BLFS indecision search.

It is easy to support the operations needed for BLFS using this cost model. Because the preferred child is always free, the predicted cost of the best leaf below any node is just the maximum cost of any node encountered enroute to the node itself. To predict the number of nodes visited for a given cost bound, we assume independence and estimate the average expected branching factor at each level of the tree. This depends on the number of children at each level whose costs we expect to fall below the cost bound. Although we probably do not know all of the costs we will see, we will have seen many of them during the previous iteration of indecision search. Recall that each iteration explores a superset of the previous one, and that all of the child scores at each node of the previous search will have been computed in order to guide that search (step 9). We will use these scores as samples to estimate the probability distribution over possible costs for each child rank at each level of the tree. In the experiments reported here, these estimated distributions were represented as histograms (see the appendix for details). The expected effective branching factor at each level, call it b_k, can then be computed using the probability that each possible number of children is the most we can afford,[1] which we write as $p(\mathit{max\ is\ i})$:

$$\begin{aligned}
p(\mathit{max\ is\ i}) &= p(\mathit{can\ afford\ i}) \times (1 - p(\mathit{can\ afford\ i} + 1)) \\
b_k &= \sum_{i=1}^{\mathit{max\text{-}num\text{-}children}_k} i \times p(\mathit{max\ is\ i})
\end{aligned}$$

If a node at level k is a leaf with probability $\mathit{leaf\text{-}prob}_k$, then the number of nodes at that level, nodes_k, can be computed from the number of nodes at the previous level that aren't leaves, times their fertility:

$$\begin{aligned}
\mathit{nodes}_0 &= 1 \\
\mathit{nodes}_k &= \mathit{nodes}_{k-1} \times (1 - \mathit{leaf\text{-}prob}_{k-1}) \times b_{k-1}
\end{aligned}$$

The $\mathit{leaf\text{-}prob}_k$ parameters can be easily estimated during the previous iteration. By summing the nodes_k over the levels of the tree (and adding in the leaves generated at each level), we can estimate the number of nodes that will be visited for a given cost bound. Note that, although we estimate on-line the node costs we expect to observe in the tree, the underlying leaf cost model itself is fixed as exactly the maximum node cost in the path, and is not adjusted during search. The initial iteration of the algorithm (step 1) visits all leaves of predicted cost zero.

Other schemes in addition to iterative broadening can be viewed as approximations to indecision search. The randomized restarting technique of Gomes, Selman, and Kautz (1998) and the GRASP methodology of Feo and Resende (1995) both randomly permute all children whose scores are within a specified distance from the preferred child, between iterations of search. These techniques depend on an equivalence parameter that must be tuned using pilot experiments, and restarting also depends on a time limit parameter. Also, because they regard closely-scoring children as equivalent, these techniques throw away information that can be systematically exploited by indecision search.

Experiments were also performed using a cost model that predicted the cost of a leaf to be the sum of the child costs along its path, rather than just the maximum. This model seemed to perform similarly, and is more

[1] When recording the costs of a node's children, we assume they are independent and store them in separate distributions. But when calculating the number of children we can afford, we want to know the conditional probability that we can afford a child *given that we could afford the previous children.* To calculate this properly, we should instead store the difference in cost between each child and the previous one. (Thanks to Chung-chieh Shan for this insight.) Empirically, however, the two strategies seem to perform similarly, and the naive approach is faster, as it avoids a convolution.

cumbersome to manipulate, so we omit further discussion of it (see Ruml (forthcoming) for details).

3.1 Evaluation

We can use constraint satisfaction problems to evaluate indecision search, as they are commonly solved using a heuristic scoring function to rank children in increasing order of 'constrainingness.' We will examine two domains: latin square completion and random binary CSPs.

3.1.1 Latin Squares

A latin square is an n by n array in which each cell has one of n colors. Each row and column must contain each color exactly once. Gomes and Selman (1997) proposed the completion of partially-filled latin squares as a challenging benchmark problem, noting that it provides both regular structure, due to the row and column constraints, and random elements, due to the preassigned cells. We used forward-checking, choosing variables to assign according to the most-constrained variable heuristic of Brélaz (1979) and ranking values according to the logarithm of the promise heuristic of Geelen (1992). Following Meseguer and Walsh (1998), we used 1,000 latin squares, each with 30% of the cells assigned, filtering out any unsatisfiable problems. We tested depth-first search (DFS), two version of Korf's improved limited discrepancy search (ILDS), one taking discrepancies at the top first and the other taking them at the bottom first, depth-bounded discrepancy search (DDS), and indecision search.[2]

The performance of the algorithms is shown in Figure 2 in terms of the fraction of problems solved within a given number of node generations. Small horizontal error bars mark 95% confidence intervals around the means. Depth-first search was limited to 10,000 nodes per problem, hence its mean is a lower bound. From the figure, we see that 25% of the problems were solved by visiting a single leaf (the greedy solution). Depth-first search enumerates leaves very efficiently, but is notoriously brittle and becomes hopeless lost on many problems (Gomes et al., 2000). The discrepancy search algorithms immediately retreat to the root. Indecision search first explores all ties, which may occur at intermediate levels of tree. As the searches progress, the algorithms biased toward discrepancies at the top seem to be paying a

[2]Due to an historical accident, indecision search attempted to double the number of leaves visited on each iteration, rather than the number of nodes. It is not clear that this affected the results.

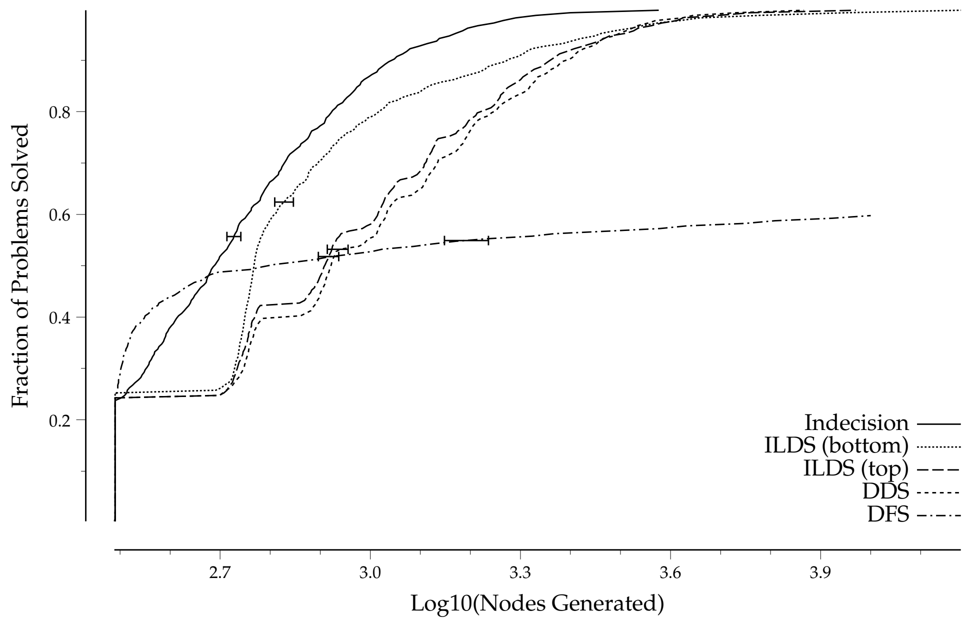

Figure 2: Distribution of search times when completing 21×21 latin squares with 30% of the cells preassigned.

price, as their progress comes in spurts. Indecision search makes efficient use of the heuristic score information, exhibiting a smooth performance profile, and it solves all the problems within 4,000 nodes (note the logarithmic scale). Similar behavior was observed on smaller instances, although the advantage of indecision search over the discrepancy methods seemed to increase as problems grew larger.

3.1.2 Binary CSPs

Binary CSPs have received much attention in the literature and were used by Meseguer and Walsh (1998) to evaluate depth-bounded discrepancy search and interleaved depth-first search. They tested on satisfiable problems of the $\langle n, m, p_1, p_2 \rangle$ type. These problems have n variables, each with m possible values. Exactly $p_1 n(n-1)/2$ of the possible pairs of variables are constrained and exactly $p_2 m^2$ of the possible value combinations are disallowed for each of those pairs. As p_2 increases toward 0.36, the constraints become tighter and the problems become more difficult to solve, exposing differences in performance between the algorithms. We will use the same heuristics as employed above with latin squares.

CSP Class	DFS	ILDS	DDS	Indec.
$\langle 30, 15, .4, .35 \rangle$	9,840	10,504	26,981	**8,839**
$\langle 50, 12, .2, .36 \rangle$	**80,851**	164,538	973,437	113,450
$\langle 100, 6, .06, .36 \rangle$	57,118	37,811	418,829	**24,065**

Table 1: The mean number of nodes generated to solve 100 instances of three classes of binary CSP.

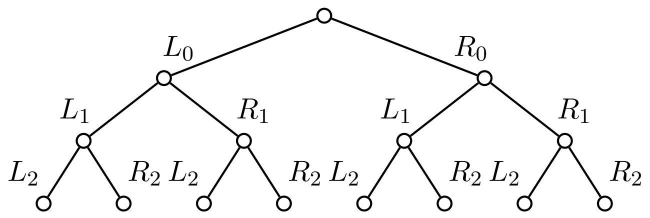

Figure 3: A simple model: leaf cost is the sum of edge costs which are parameterized by depth and child rank.

The performance of the algorithms on three problem classes is summarized in Table 1. Indecision search surpasses ILDS and DDS on all classes, and proves superior to DFS on the first and third. On the second class, indecision search takes on average only 1.4 times as many nodes as DFS, well within the worst-case factor of three.

On both benchmark domains, the indecision search instantiation of BLFS seems to provide a search order that is either superior to existing techniques or closely competitive. Next we will consider a cost model that is not fixed in advance, but is learned during the search itself.

4 BLFS with a Learned Model

In some domains, the child ranking function does not return a quantitative score, and the only information that is readily available to guide search is the costs of the leaves that have been visited. Following Ruml (2001a), we will use these observed costs to estimate the cost of taking a discrepancy at each level of the tree. More precisely, we will use a cost model that assumes that the cost of a leaf is the sum of the costs of the edges taken to reach it, and we will assume that the cost of an edge depends only on its depth and its rank in the sorted list of children. An example for a binary tree appears in Figure 3. A tree of depth d and branching factor b requires db parameters, one for each edge at each level. This generalizes DFS, ILDS, and DDS.

Because these edge costs will vary depending on the problem, we will estimate them during the search. In step 1 of BLFS, we will visit 10 random leaves. Each path forms a linear equation in the parameters of the model. If $leaf_i$ is the cost of the ith leaf visited, then three random paths might yield:

$$\begin{array}{cccccccl} L_1 & & + & L_2 & & + & & R_3 & = leaf_1 \\ L_1 & & + & & R_2 & + & L_3 & & = leaf_2 \\ & R_1 & + & L_2 & & + & L_3 & & = leaf_3 \end{array}$$

After visiting each leaf (in either step 1 or 7), we will update the parameters using a simple on-line linear regression algorithm. (In the experiments reported below, the method of Murata et al. (1997) was used. It gave very slightly better results when learning from random paths than the methods discussed by Sutton (1992).) To help ensure that the current cost bound yields the predicted number of nodes, a static copy of the model is made at the start of each iteration to guide the search. To further aid learning, the costs estimated in the experiments below were further constrained at the start of each iteration to be increasing with child rank at each depth. (In other words, it was assumed that the underlying ranking function was helpful rather than deceptive.)

This cost model also easily supports the operations required for BLFS. The cost of the best leaf in any subtree is just the sum of the edges traversed so far plus the sum of the costs of the cheapest options at each of the remaining levels. These optimal completions can be precomputed at the start of each iteration. To estimate the number of nodes that will be visited for a given bound, we just estimate the branching factor at each level, as for indecision search. We can consider the cost bound to be an allowance that is spent as we descend the tree. By estimating the distribution of allowance values expected at each level of the tree, we can estimate how many children whose best completion will be affordable at that level. (As in indecision search, these distributions are manipulated as histograms, as described in the appendix.) At the root, the allowance distribution is a spike at the given cost bound. The distribution of allowance at the next level is just the sum, over the possible children, of the portion of the current distribution that falls above the best completion cost for that child, translated toward zero by that cost. Each distribution in the sum is weighted by the proportion of the probability that survived the truncation.

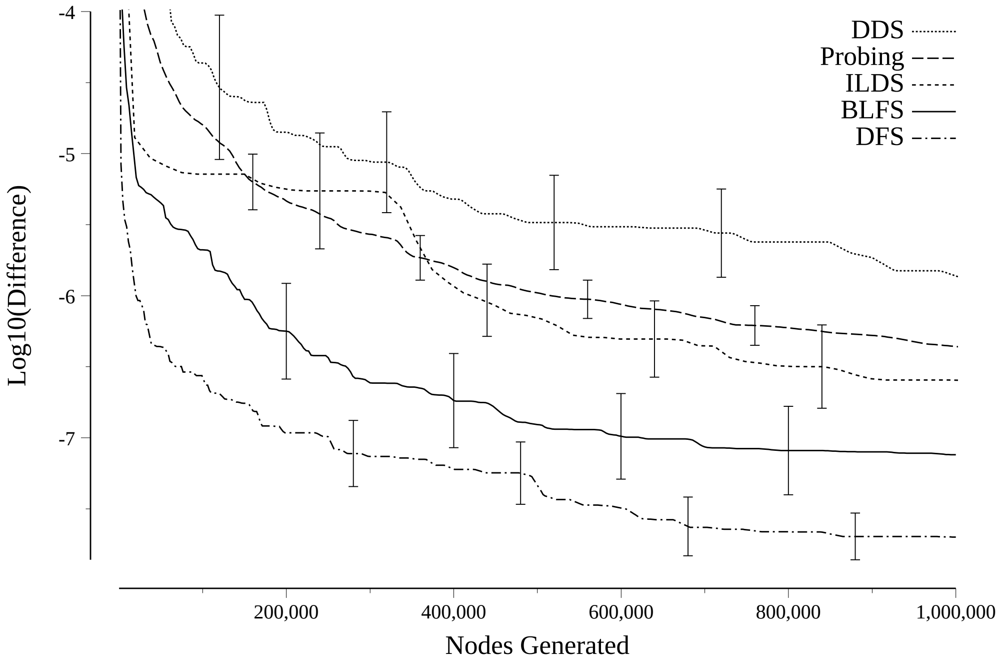

Figure 4: Greedy partitioning of 128 numbers

4.1 Evaluation

We evaluated the algorithm on two different formulations of the combinatorial optimization problem of number partitioning. The objective is to divide a given set of numbers into two disjoint groups such that the difference between the sums of the two groups is as small as possible. It has been used by many authors as a benchmark for search algorithms (Johnson et al., 1991; Korf, 1996; Walsh, 1997; Ruml, 2001a). Following Ruml, we used instances with 128 44-digit numbers or 256 82-digits numbers. Arbitrary precision integer arithmetic was used in the implementation, and results were normalized as if the original numbers had been between 0 and 1. The logarithm of the partition difference was used as the leaf cost.

4.1.1 Greedy Number Partitioning

The first formulation of partitioning as a search is a straightforward greedy encoding in which the numbers are sorted in descending order and then each decision places the largest remaining number in a partition, preferring the partition with the currently smaller sum. Figures 4 and 5 compare the

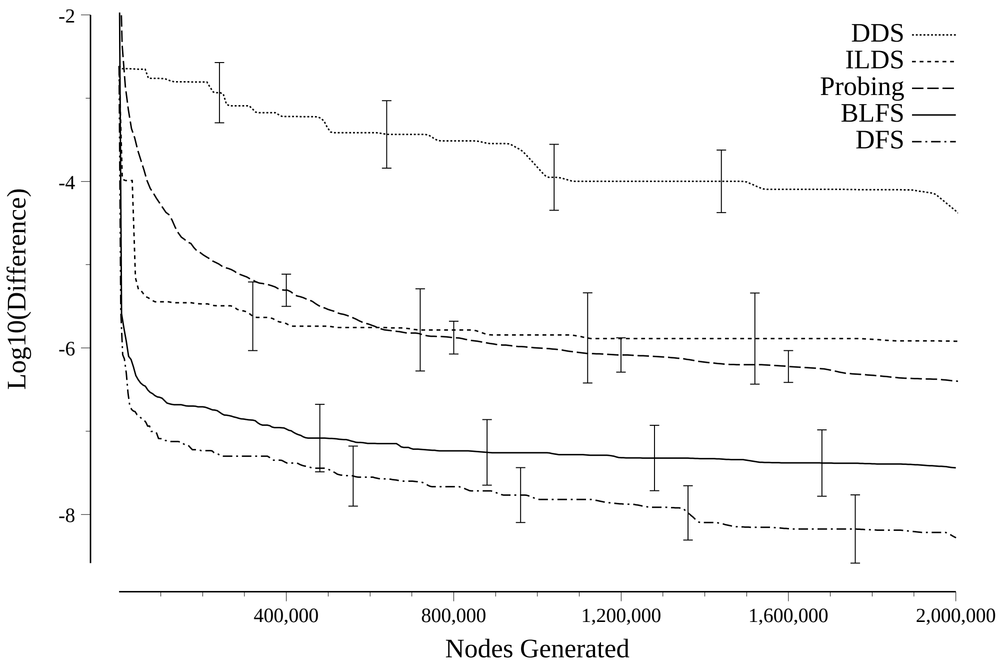

Figure 5: Greedy partitioning of 256 numbers

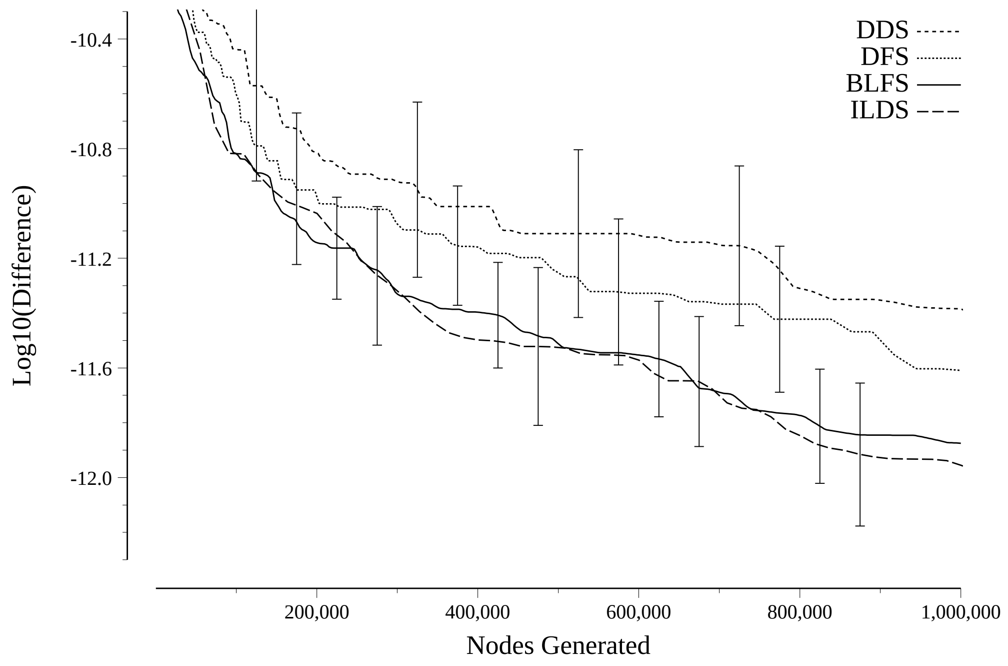

Figure 6: CKK representation for partitioning 128 numbers

performance of BLFS with DFS, ILDS, DDS, and the adaptive probing algorithm of Ruml (2001a), which guides search using a similar learned cost model but is stochastic and incomplete. Error bars in the figures indicate 95% confidence intervals around the mean. Although BLFS does not surpass DFS in this search space, it does seem to consistently track DFS as the problem size increases, unlike ILDS and DDS, whose solution quality actually decreases on the larger problems. BLFS also does not seem to suffer in comparison to adaptive probing, even though it has a further guarantee of completeness.

4.1.2 CKK Number Partitioning

A more sophisticated representation for number partitioning was suggested by Korf (1995), based on the heuristic of Karmarkar and Karp (1982). The essential idea is to postpone the assignment of numbers to particular partitions and merely constrain pairs of number to lie in either different bins or the same bin. Numbers are considered in decreasing order and constrained sets are reinserted in the list according to the remaining difference they represent. Figure 6 and 7 compare the performance of BLFS with DFS, ILDS,

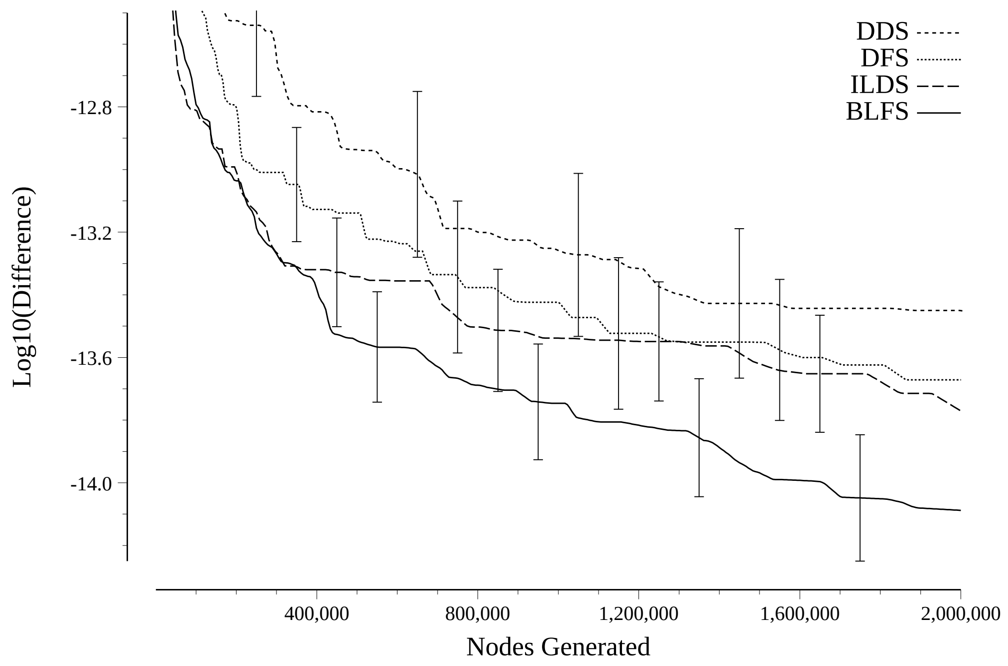

Figure 7: CKK representation for partitioning 256 numbers

	BLFS	IDA*
$f(n)$	best leaf below n	best path through n
model	from user	additive
$g(n)$	learned	from problem
$h(n)$	learned	from user
updating bound	estimation	add ϵ
	rational	optimal

Table 2: A comparison of BLFS and IDA*.

and DDS. (Adaptive probing takes too long to learn to follow the powerful heuristic in this space and would be off the top of both plots.) Whereas BLFS was tracking DFS in the greedy search space, it performs comparably to ILDS when using this CKK representation on smaller problems. And on larger problems, BLFS surpasses all the other algorithms.

In summary, we have seen that BLFS can successfully adapt to different search spaces, even given only observed leaf costs. It is more robust than other algorithms, always performing competitively with or better than the best previous strategy. Of course, comparisons against additional algorithms and tests in domains of more practical utility will yield a fuller perspective.

5 Relations to Shortest-Path Algorithms

Although BLFS was designed for combinatorial optimization and constraint satisfaction, its basic framework of iteratively expanding bounded search is also found in the iterative-deepening A* (IDA*) algorithm for shortest-path problems (see Table 2). Both algorithms visit nodes in an order that approximates increasing predicted cost. IDA* is guided by a node evaluation function, traditionally notated $f(n)$, which is the sum of two parts: the path cost from the start state to n, notated $g(n)$, and a heuristic estimate of the cost of the best path from n to the goal, notated $h(n)$. If $h(n)$ never overestimates, IDA* returns an optimal path. By the nature of its task, IDA* assumes the cost of a path is linear in the edge costs. The node evaluation in BLFS reflects the cost of the best leaf in the node's subtree, and its model was divided into additive components in this paper only for efficiency (as we saw when precomputing the optimal path completions for BLFS). Because the edge costs are not inherent in the problem, the model must be learned. And because the cost model is explicit and known, the cost bound can be updated efficiently even when few nodes have the same cost, unlike with IDA*.

This unification clarifies the common confusion that many newcomers to AI feel when they see the term 'heuristic search' applied to both shortest-path problems with an explicit $h(n)$ and also to procedures like DFS with a node ordering function. BLFS makes it clear that a node ordering function is just a rough indicator of the cost of the best leaf in the subtree, and by adhering to this semantics, it approximates a rational search order.

6 Possible Extensions

It would be very interesting to explore other models besides those investigated here. It should be straightforward to combine on-line learning of weights with the heuristic child scores used in indecision search. This would relax the assumption that heuristic scores are strictly comparable across levels. Multiple models could be trained simultaneously and the one with the lowest error on the previous iteration could be used to guide search. By constraining adjacent costs to be similar, fewer parameters would be needed in the model, and it might be feasible to consider learning models for both value choice and variable choice (Ruml, 2001b).

BLFS currently does not take into account the uncertainty in its cost model or the possible benefits of visiting a leaf predicted to be poor. A drastically misestimated cost can cause the search to avoid the corresponding edge and fail to correct the estimate. One way to remedy this would be to use as a node evaluation the probability that the node leads to the optimal leaf. This could be computed from a child cost model by estimating variance and assuming normality, following Ruml (2001a). The cost bound on each iteration would become a probability bound. This seems similar to the methods proposed by Bedrax-Weiss (1999), although her algorithm was trained and scheduled off-line.

Techniques for managing the trade-off between time and expected solution improvement are orthogonal to this work, and could be applied on top of it. Mayer (1994) and Hansson (1998) have done preliminary work in this direction.

7 Conclusions

We introduced best-leaf-first search (BLFS), a new framework for searching the bounded-depth trees that arise in combinatorial optimization and constraint satisfaction problems. BLFS generalizes previous work, and represents the first successful rational approach to search for this setting. Empirical results show that, even with simple cost models, BLFS performs well on a variety of synthetic benchmark problems, yielding results competitive with or superior to the best previous method for each problem. It retains completeness while adapting on-line to individual problem instances, and uses an explicit model of its assumptions. Perhaps most importantly, BLFS shows how search for combinatorial optimization and constraint satisfaction can be viewed from a perspective similar to that of traditional heuristic search for shortest-path problems, as the strategy of a rational agent trying to efficiently take advantage of heuristic information for problem-solving.

8 Acknowledgments

Stuart Shieber and the Harvard AI Research Group gave numerous helpful suggestions and comments. This work was supported in part by NSF grants CDA-94-01024 and IRI-9618848.

9 Appendix: Histogram Implementation

The probability distributions described in the text are estimated on-line using histograms. When adding samples to an empty histogram, individual data values are recorded until a fixed size limit is reached (100 in the experiments reported here). At this point, each value becomes the center of a bin which reaches halfway to its neighboring values. (Bins on the ends are symmetrical.) When additional samples are added, the weights of the appropriate bins are increased. When the weight of a single bin exceeds twice the lowest weight of any adjacent pair of bins, the heavy bin is split into equal halves and the light pair is collapsed. Operations such as the addition of distributions are fairly straightforward but tedious to program. Four cases must be considered, as the two distributions involved may each consist of point-like values or bins. Considering histogram elements in a random order avoids biasing bin accuracy. Time complexity is bounded by the square of the histogram size limit. Source code will be made freely available at the author's website.

References

Bedrax-Weiss, Tania. 1999. *Optimal Search Protocols.* Ph.D. thesis, University of Oregon, Eugene, August.

Brélaz, Daniel. 1979. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, April.

Feo, T. A. and M. G. C. Resende. 1995. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133.

Geelen, P. A. 1992. Dual viewpoint heuristics for binary constraint satisfaction problems. In B. Neumann, editor, *Proceedings of ECAI-92*, pages 31–35.

Ginsberg, Matthew L. and William D. Harvey. 1992. Iterative broadening. *Artificial Intelligence*, 55:367–383.

Gomes, Carla P. and Bart Selman. 1997. Problem structure in the presence of perturbations. In *Proceedings of AAAI-97*, pages 221–226.

Gomes, Carla P., Bart Selman, Nuno Crato, and Henry Kautz. 2000. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24:67–100.

Gomes, Carla P., Bart Selman, and Henry Kautz. 1998. Boosting combinatorial search through randomization. In *Proceedings of AAAI-98.*

Hansson, Othar. 1998. *Bayesian Problem-Solving Applied to Scheduling.* Ph.D. thesis, University of California, Berkeley.

Harvey, William D. and Matthew L. Ginsberg. 1995. Limited discrepancy search. In *Proceedings of IJCAI-95*, pages 607–613. Morgan Kaufmann.

Johnson, David S., Cecilia R. Aragon, Lyle A. McGeoch, and Catherine Schevon. 1991. Optimization by simulated annealing: An experimental evaluation; Part II, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, May-June.

Karmarkar, Narenda and Richard M. Karp. 1982. The differencing method of set partitioning. Technical Report UCB/CSD 82/113, Computer Science Division, University of California, Berkeley.

Korf, Richard E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109.

Korf, Richard E. 1995. From approximate to optimal solutions: A case study of number partitioning. In *Proceedings of IJCAI-95*.

Korf, Richard E. 1996. Improved limited discrepancy search. In *Proceedings of AAAI-96*, pages 286–291. MIT Press.

Mayer, Andrew Eric. 1994. *Rational Search*. Ph.D. thesis, University of California, Berkeley, December.

Meseguer, Pedro and Toby Walsh. 1998. Interleaved and discrepancy based search. In *Proceedings of ECAI-98*.

Murata, Noboru, Klaus-Robert Müller, Andreas Ziehe, and Shun-ichi Amari. 1997. Adaptive on-line learning in changing environments. In Michael Mozer, Michael Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems 9 (NIPS-96)*, pages 599–605. MIT Press.

Ruml, Wheeler. 2001a. Incomplete tree search using adaptive probing. In *Proceedings of IJCAI-01*, pages 235–241.

Ruml, Wheeler. 2001b. Stochastic tree search: Where to put the randomness? In Holger H. Hoos and Thomas G. Stützle, editors, *Proceedings of the IJCAI-01 Workshop on Stochastic Search*, pages 43–47.

Ruml, Wheeler. forthcoming. *Adaptive Tree Search*. Ph.D. thesis, Harvard University.

Sutton, Richard S. 1992. Gain adaptation beats least squares? In *Proceedings of the Seventh Yale Workshop on Adaptive and Learning Systems*, pages 161–166.

Walsh, Toby. 1997. Depth-bounded discrepancy search. In *Proceedings of IJCAI-97*.