To appear in: *Proceedings 1995 International Joint Conference on Artificial Ingelligence (IJCAI-95)*, Montreal, August, 1995.

k-best: A New Method for Real-Time Decision Making

Joseph C. Pemberton
pemberto@cs.uoregon.edu
Computational Intelligence Research Laboratory
1269 University of Oregon
Eugene, OR 97403-1269
U. S. A.

Abstract

Many real-world problems, such as air-traffic control and factory scheduling, require that a sequence of decisions be made in real time. The real-time constraint means that we typically do not have sufficient time to find a complete solution to the problem using traditional methods before we must commit to a decision. We propose an incremental search approach to making real-time, sequential decisions, and then present a new decision method called k-best, which is both an extension of an existing real-time decision method (MINIMIN) and an approximation to a decision-theoretic approach to the real-time decision problem. We next provide an analytical bound on the worst-case expected error when k-best is used instead of the optimal decision method. The average-case performance of k-best is then compared to MINIMIN on a set of randomly generated problems. Our results show that k-best is an improvement over MINIMIN, although MINIMIN performs quite well. Given that MINIMIN is very efficient and easy to implement, we conclude that it should be the algorithm of choice for many real-time decision problems.

1 Introduction

One example of real-time decision making is factory scheduling when the objective is to keep a bottleneck resource busy. In this case, the amount of time available to decide which job should be processed next is limited by the time required for the bottleneck resource to process the current job. Once a new job is scheduled, the time until the new job finishes processing can be used to decide on the next job. In general, this class of problems requires the problem solver to incrementally generate a sequence of time-limited decisions. This consists of three sub-parts: gathering information (*e.g.,* what jobs need to be processed), deciding when to stop gathering information (*i.e.,* what the decision deadline is), and making a decision based on the available information (*e.g.,* what job to process next). For this paper, we have focussed on the last question, namely how to make decisions based on a partially explored problem space. We first summarize the incremental search approach to real-time decision making based on searching a random decision tree with limited computation time. We next define the last incremental decision problem and develop the necessary steps for making an optimal incremental decision. We then argue that this approach is impractical due to the size and complexity of the equations needed to express the expected value. Next, we present and analyze k-best, which is both an approximation to the optimal decision method and also an extension of MINIMIN. We present experimental results that compare the performance of MINIMIN and k-best on randomly generated problems. Finally, we discuss related work and conclusions.

2 A Real-Time Decision Problem

Real-time decision making can be modeled as searching a problem-space tree with a finite amount of computation. In this model, each node of the tree corresponds to a decision point, and the edges emanating from a node correspond to the choices (operators or actions) available for the decision at that node. Each edge has a cost or penalty that the problem solver will incur if it chooses to execute the action associated with that edge. The $node\_cost(x)$ is the sum of the edge costs along the path from the root to node x. An example random tree is shown in Figure 1. The objective is to reach a lowest-cost leaf node given the computational constraints. The more general problem of finding the minimum-cost leaf node in a random-tree has been studied in depth (see [Zhang and Korf, 1995] for a summary). In general, the task of finding a minimum-cost leaf node typically requires computation that is exponential in the tree depth. For real-time search, there is not sufficient time to find an optimal solution, so we must consider search methods that satisfy the real-time constraint.

We model the real-time constraint as a constant number of node generations allowed per decision. This is equivalent to having a deadline for each decision. We assume that the problem solver gathers information by expanding nodes (exploration) until the time available runs out, after which it chooses one of the children of the current root node as the new root node (decision making). This process is repeated, incrementally generating a complete solution.

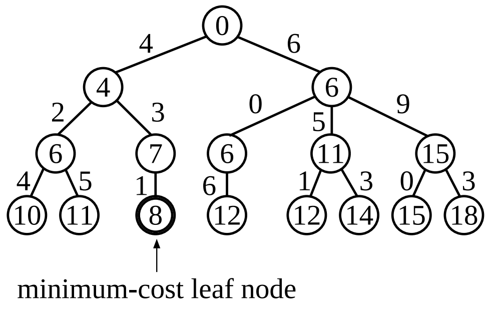


Figure 1: Example of a random tree problem.

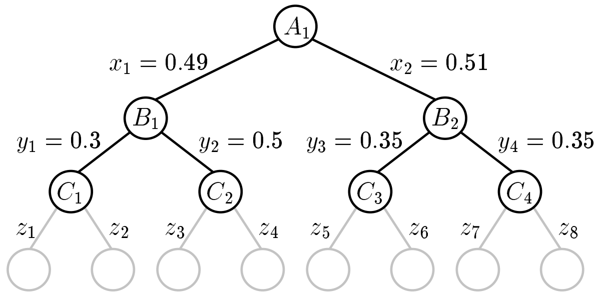


Figure 2: Example of the last incremental decision problem. Which node is a better decision, B_1 or B_2?

3 The Incremental Decision Problem

We begin by describing the last incremental decision problem, which is a simplified version of the general incremental decision problem. An example of a last incremental decision problem is shown in Figure 2. In this figure, the black edge costs ($x_1, x_2, y_1, y_2, y_3, y_4$) are known, and the grey edge costs ($z_1, ..., z_8$) are independently chosen at random from a known distribution. The decision task is to choose a child of the current root node so that the expected cost of a complete root-to-leaf path is minimized. After we choose a child of the current root node, then we can expand the frontier nodes below that child, and learn the remaining edge costs in the chosen subtree, before making the remaining decisions. The last incremental decision problem is basically a choice between a set of partially explored subtrees in which we will subsequently execute a sequence of optimal decisions.

The general incremental decision problem is just an extension of the last incremental decision problem to an arbitrary number of decisions. The main advantage of studying the last incremental decision problem over the general incremental decision problem is that we know that the remainder of the tree will be explored after the decision is made. This means that the distribution of minimum-cost paths that will be traversed below a given child of the root node is the same as the distribution of minimum-cost paths in the tree below that child, independent of the decision and exploration methods. This is why we have focussed our efforts on the last incremental decision problem.

4 Making a Last Incremental Decision

Consider the last incremental decision problem shown in Figure 2. The top two levels have been explored, and the bottom level of the tree (the gray edges and nodes) has not been explored. At this point, we must choose between nodes B_1 and B_2. We know that the remaining edge costs are chosen independently from a uniform distribution over $[0, 1]$, and that once we make a choice, we will be able to explore the remaining edge costs in the chosen subtree before making any additional decisions. The reader is encouraged to stop and answer the following question: Should we move to node B_1 or node B_2? The obvious answer to this question is to move toward B_1 because it is the first step toward the lowest-cost frontier node ($node\_cost(C_1) = .49 + .3 = .79$). This decision strategy, which is called minimin decision making in [Korf, 1990] (or simply MINIMIN), has been employed by others for single-agent search (*e.g.*, [Russell and Wefald, 1991]), and is also a special case of the two-player minimax decision rule [Shannon, 1950].

In fact, the optimal decision is to move to node B_2 because it has a lower expected minimum root-to-leaf path cost. The expected minimum root-to-leaf path cost (or simply expected minimum path cost) through either B_1 or B_2 is the equal to the edge cost between the root and the child node plus the expected cost of the path that will be traversed after the remaining edge costs are learned. To make an optimal last incremental decision (and to show that B_2 is the better decision in our example), we must calculate the expected minimum path cost through both children of the root node given the edge costs in the explored tree and the edge-cost distribution for the unexplored edges.

In order to calculate the expected minimum path cost for a child of the root node, we need to first calculate the distribution of minimum path costs for that child node. In general, the problem of calculating the distribution for a minimum-cost path from a child of the root node to a leaf node can be broken down into two steps. The first step is to determine the distribution for a minimum-cost path from a frontier node to a leaf in the tree. The second step is to combine the frontier node completion-cost distributions with the known edge costs in the explored search tree.

For node C_1 in Figure 2, there is one unexplored level between the frontier node (C_1) and the leaf nodes below it. The cumulative distribution $F_{C_1}(z)$ can be calculated as follows ($F(z) = z$ is the edge-cost distribution). We assume that the edge-cost distributions are independent.

$$\begin{aligned}
F_{C_1}(z) &= F(min[z_1, z_2]) \\
&= p(min[z_1, z_2] < z) \\
&= 1 - p(min[z_1, z_2] > z) \\
&= 1 - p(z_1 > z) \cdot p(z_2 > z) \\
&= 1 - (1 - p(z_1 < z)) \cdot (1 - p(z_2 < z)) \\
&= 1 - (1 - F(z)) \cdot (1 - F(z)) \\
&= 2F(z) - F(z)^2
\end{aligned}$$

Given the distribution for a minimum-cost path from a frontier node to a leaf node, the second step is to calculate the expected minimum path cost through a child

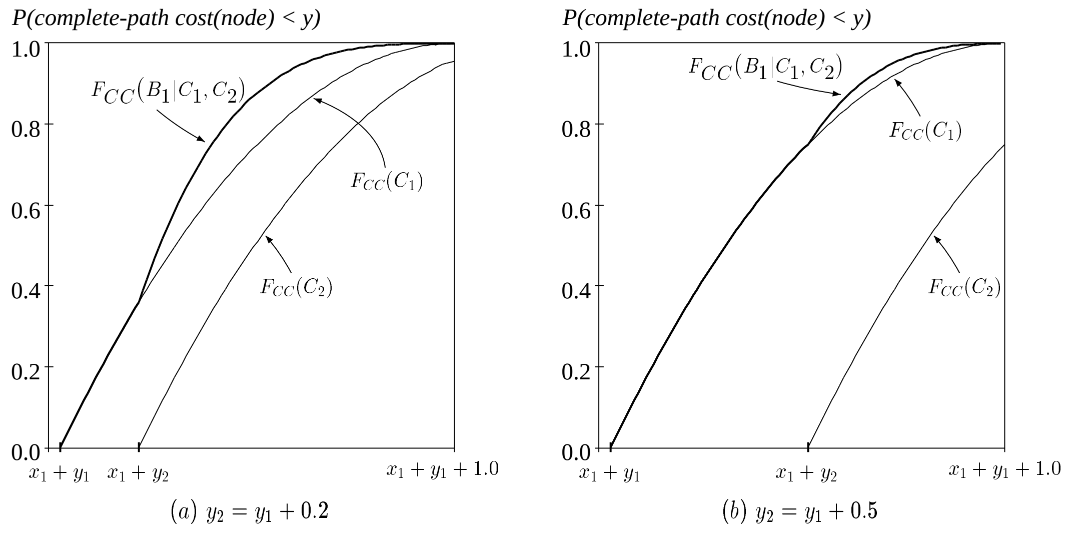


Figure 3: Completion-cost cumulative distribution functions for root-to-leaf paths through nodes B_1, C_1 and C_2 when (a) $y_2 = y_1 + 0.2$ and (b) $y_2 = y_1 + 0.5$.

of the root node. This involves constructing a piecewise combination of the frontier-node minimum-path-cost cumulative-distribution functions (CDF's). Figure 3 shows the CDF's for frontier nodes C_1 and C_2 as well as the combined CDF for node B_1 for two different relative values of y_1 and y_2 ($F_{CC}(C_1) = F_{C_1}(z - y_1)$, $F_{CC}(C_2) = F_{C_2}(z - y_2)$).

Finally, the cumulative-distribution functions for the children of the root node (*i.e.,* B_1 and B_2) are differentiated to yield the probability density functions (PDF's) for minimum path cost through these nodes. These PDF's are then used to calculate the expected minimum path costs through each child of the root by integrating over the range of possible root-to-leaf path costs. The optimal last incremental decision is then to move to the child with the lowest expected minimum path cost.

We can now return to the example in Figure 2. If we assume without loss of generality that $y_1 \leq y_2$, then the equation for the expected cost of a complete path through node B_1 is:

$$E(min\_path\_cost(B_1)) =$$

$$x_1 + \int_{y_1}^{y_2} 2(1 + y_1 - y) \cdot y\, dy \qquad (1)$$

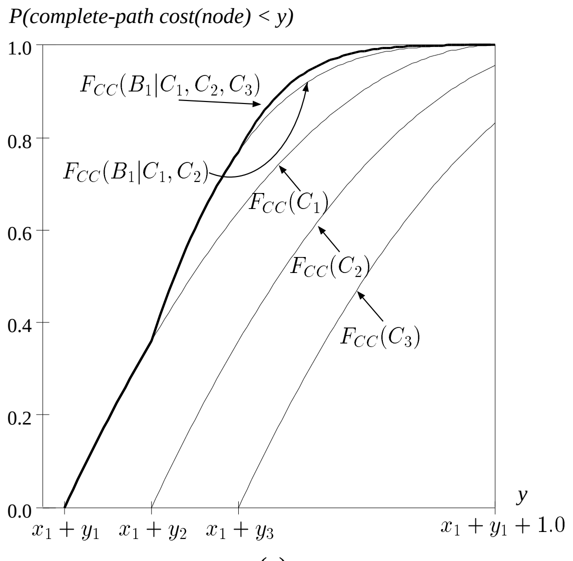
$$+ \int_{y_2}^{1+y_1} 2((1 + y_1 - y) \cdot (1 - 2y + 2y_2 + (y - y_2)^2)$$

$$+(1 + y_2 - y) \cdot (1 - 2y + 2y_1 + (y - y_1)^2 \cdot y\, dy$$

An analogous equation applies for node B_2. Substituting the edge costs from Figure 2 into Equation 1, we get $E(min\_path\_cost(B_1)) = 1.066$, whereas $E(min\_path\_cost(B_2)) = 1.06$. Thus node B_2 is the optimal decision for this example. Intuitively, a move to node B_1 relies on either z_1 or z_2 having a low cost, whereas a move to B_2 has four chances (z_5, z_6, z_7 and z_8) for a low final edge cost. Thus, an optimal last incremental decision is a move toward the child node with the lowest expected root-to-leaf path cost that will be traversed after the incremental decision is made and the remaining edge costs are learned.

In general, finding an optimal last incremental decision is impractical for all but a few small search trees. The difficulty comes from the fact that the distribution of minimum path costs through a child of the root can have a distinct distribution function for each frontier node in the child node's subtree. For example, consider the graph and tree shown in Figure 4. This tree has one more frontier node under B_1 than our original example (Figure 2). The graph shows the cumulative distribution of minimum root-to-leaf path costs through nodes C_1, C_2, C_3, and B_1 versus the cost of a path to a leaf node. We observe that each of the three frontier nodes under B_1 is responsible for a change to the minimum-path-cost distribution for B_1 ($F_{CC}(B_1)$). For example, the change in the distribution at $y = x_1 + y_2$ is due to the fact that at first every minimum-cost path through B_1 must also pass through C_1. Once the minimum path cost is greater than or equal to $x_1 + y_2$, then a minimum-cost path below B_1 can be through either C_1 or C_2. This additional choice causes a sharp rise in the cumulative distribution function for B_1 at $x_1 + y_2$, as shown in the graph. Since the number of frontier nodes in each subtree is exponential in the search depth, in the worst case the number of different functions needed to describe the minimum-path-cost distribution is also exponential in the search depth. This is why calculating the expected minimum root-to-leaf path cost and consequently making an optimal last incremental decision is only possible for a set of small search trees.

Thus far, we have presented two disparate incremental decision methods: MINIMIN, which is easy to calculate but makes suboptimal decisions, and the optimal decision strategy, which only works on small search trees. In the next section, we present a new decision method that bridges the gap between MINIMIN and the optimal decision method.



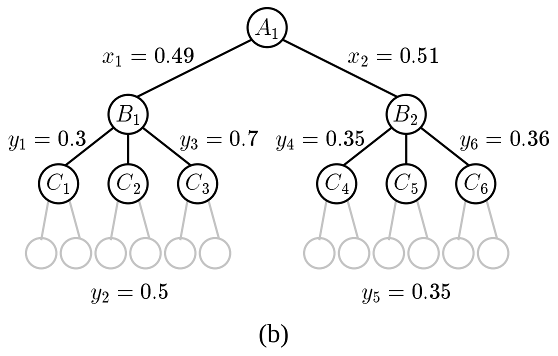


Figure 4: Completion-cost cumulative-distribution functions for the minimum root-to-leaf path cost through nodes B_1, C_1, C_2 and C_3 when $y_2 = y_1 + 0.2$ and $y_3 = y_2 + 0.2$ and the associated search tree (b).

5 k-Best: A New Incremental Decision Algorithm

k-best is a new algorithm for making incremental decisions. It operates by maintaining a list of the k-best frontier nodes under each child of the root node as the subtrees are explored. These k-best frontier node costs are then used to calculate an approximation of the expected minimum root-to-leaf path cost for each child of the root node. The approximation is calculated by assuming that the child's subtree only contains the k-best frontier nodes. The k-best decision is simply to move to the child with the lowest k-best estimate of the expected minimum root-to-leaf path cost.

Consider again the tree shown in Figure 4b. First, we observe that the the lowest-cost leaf node (C_1) under a given child of the root node has the greatest effect on the distribution of minimum-cost paths through that child because it determines the starting point of the minimum-path-cost distribution (*i.e.*, $F_{CC}(B_1)$ initially equals $F_{CC}(C_1)$ because C_1 is the lowest cost frontier node). The next largest effect is due to the location of the second-best frontier node (C_2) in a child's subtree, and the size of the effect diminishes as we approach the highest-cost frontier node. For example, when the minimum root-to-leaf path cost through B_1 becomes greater than or equal to $x_1 + y_2$, then the distribution of minimum path costs for B_1 shifts to a new function that combines the distributions of minimum path costs through C_1 and C_2. When the minimum path cost through B_1 is greater than or equal to $x_1 + y_3$, then the minimum-path-cost distribution for B_1 shifts again to reflect the third frontier node, although the size of the second shift is much smaller than the first shift. In fact, it is often the case that the higher-cost frontier nodes under a given child node do not contribute at all to the distribution of minimum-cost path through that child because their node cost exceeds the minimum frontier-node cost by more than the maximum possible edge cost.

k-best takes advantage of this observation by simply ignoring all but the k-best frontier node costs below each child of the root node. To make k-best tractable, we simply choose a value for k that is small enough so that we can calculate the expected minimum-path-cost equation. When $k = 1$, k-best and MINIMIN will make the same decisions (up to tie-breaking). Alternatively, when the number of frontier nodes in each subtree is less than or equal to k, then k-best makes optimal last incremental decisions. Thus k-best defines a continuum of decision algorithms between MINIMIN and the optimal decision method.

The obvious question at this point is what is the cost in terms of solution quality of this approximation? The worst-case expected error for a k-best decision occurs when k-best chooses a decision that only has k good frontier nodes, and all the frontier nodes below the decision that wasn't chosen by k-best are good, in the sense that their node costs are very close to (or perhaps equal to) the minimum frontier-node cost. This is the situation where the optimal incremental decision has greatest potential advantage over k-best. An example of this situation is shown in Figure 5, where ϵ is a small constant.

For this example, the k-leftmost frontier nodes all have cost x, which is the minimum frontier-node cost, and the other $m - k$ frontier nodes below node B_1 have node costs equal to $x + 1$, which is equal to the minimum frontier-node cost plus the maximum edge cost. This means that only these k minimum-cost frontier nodes can contribute to the expected minimum path cost below B_1. We assume that there are m frontier nodes below B_2, all of which have a node cost equal to $x + \epsilon$. We also assume that there is one remaining unexplored level of the tree and that the branching factor in this level is b.

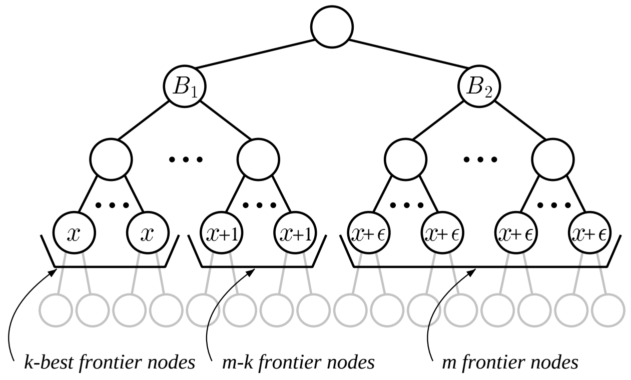


Figure 5: Example of the cost of using the k-best approximation to make incremental decisions.

The expected error of a k-best decision is simply the difference between the expected minimum path cost below the k-best decision (B_1), and the expected minimum path cost below the optimal decision (B_2). For simplicity, we assume that the unexplored edge costs are independently chosen from a uniform distribution over the range $[0,1]$ (*i.e.,* $F(z) = z$ $(0 \leq z \leq 1)$). Since the k minimum-cost frontier nodes are the only frontier nodes that can appear on a minimum-cost path below B_1, the expected minimum path cost below B_1 is equal to x plus the expected cost of a minimum choice between kb random edge costs. For this edge-cost distribution, the expected cost of a minimum choice between kb random edge costs is $1/(kb+1)$.

In order to calculate the expected minimum path cost below B_2, we simply observe that the optimal decision after moving to B_2 will be a choice between mb edge costs added to one of the m frontier nodes that have cost $x+\epsilon$. For this edge-cost distribution, the expected value of mb random choices is $1/(mb+1)$. Thus the expected minimum path cost below B_2 is $x+\epsilon+1/(mb+1)$.

The worst-case expected error for a k-best decision, on this example problem, is simply the difference between these two expected minimum path costs. In the limit as m goes to infinity and ϵ goes to zero, this difference approaches $1/(kb+1)$, which is the expected completion cost for the path below a set of k minimum-cost frontier node. This make sense, because as the number of frontier nodes below B_2 increases, it becomes more likely that a zero-cost edge will be generated.

Thus the worst-case expected error for k-best on a last incremental decision problem is bounded by the expected cost of a choice between kb unexplored edges. This expected-cost bound is a decreasing function of k. In addition, the amount of information that can be gained as k is increased is also a decreasing function of k. Intuitively, it seems reasonable that the most important piece of information about a root-child decision subtree is the minimum frontier-node cost, and the second most important piece of information is the second smallest frontier-node cost, etc.

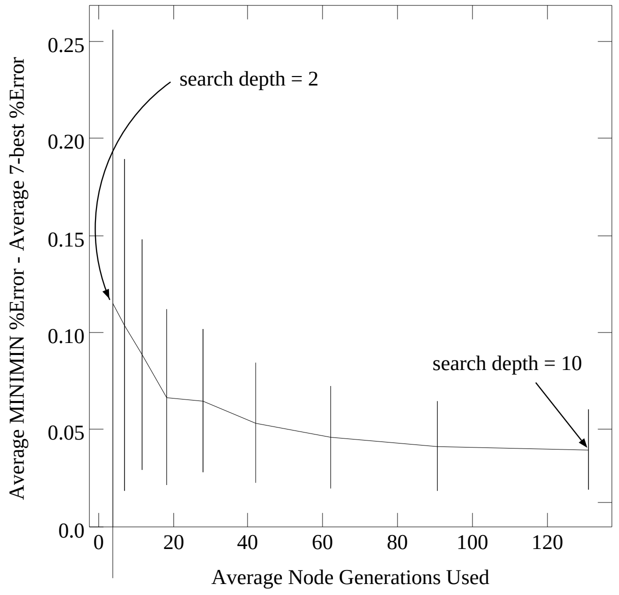


Figure 6: Average MINIMIN %error minus average 7-best %error versus average node generations ($b=2$).

6 Experimental Comparison of k-best and MINIMIN

In order to compare the average case performance of k-best and MINIMIN, we performed a set of experiments on random trees with fixed branching factor, and with edge costs that are independently chosen from the set $\{0, 1/r, 2/r, ..., (r-1)/r; r = 1024\}$. A value of 7 was chosen for k to make the k-best equations manageable. For each trial, random values were assigned to each edge of the problem-space tree, and then both MINIMIN and 7-best were allowed to explore the problem space using a depth-first branch-and-bound exploration up to one level above the bottom of the problem-space tree. This is a last incremental decision problem with one unexplored level. After the MINIMIN and 7-best decisions were calculated, the remainder of the tree was explored in order to optimally complete both paths. In addition to the 7-best and MINIMIN path costs, we also calculated the optimal path cost in the completely explored problem-space tree. We then recorded the percent solution-cost error, calculated as follows.

$$\%error = 100 * \frac{(algorithm\ solution\ cost - optimal\ cost)}{optimal\ cost}$$

We also compared the solution cost of the two algorithms, and recorded the number of times that each algorithm produced a lower-cost solution.

The graph in Figure 6 shows the difference in average solution cost error for MINIMIN and 7-best. The vertical axis is the difference between the average MINIMIN percent solution-cost error and the average 7-best solution-cost error, whereas the horizontal axis is the average generations used per decision for search depths ranging from 2 to 10. Each data point is an average of 10,000 trials, and the vertical line at each point indicates the 95% confidence interval. The results show

Search depth	7-best % wins	*MINIMIN* % wins	ratio
1	0.0 ± 0.00	0.0 ± 0.00	—
2	1.7 ± 0.08	1.5 ± 0.07	1.133
3	1.9 ± 0.08	1.7 ± 0.08	1.117
4	2.0 ± 0.09	1.6 ± 0.08	1.250
5	1.9 ± 0.08	1.5 ± 0.08	1.267
6	1.9 ± 0.08	1.5 ± 0.08	1.267
7	1.8 ± 0.08	1.5 ± 0.08	1.200
8	1.8 ± 0.08	1.5 ± 0.07	1.200
9	1.8 ± 0.08	1.4 ± 0.07	1.285
10	1.8 ± 0.08	1.4 ± 0.07	1.285

Table 1: Percent wins for 7-best versus MINIMIN.

that 7-best produces solutions with lower average percent error than MINIMIN. Although the improvement of 7-best over MINIMIN is very small, it is consistent over the range of random-tree problems considered. The results in Table 1 shows the percentage of the trials that either 7-best or MINIMIN produced a lower cost solution (*i.e.,* wins a head-to-head competition), for the same set of experiments (with 95% confidence intervals). We note that 7-best wins more of the head-to-head competitions than MINIMIN for a given search depth. Although the percentage wins by 7-best is only slightly higher than the percentage wins by MINIMIN, this difference is also consistent over the range of random-tree problems considered.

7 Making a Series of Decisions

Up to this point we have only considered the last incremental decision problem. We have also applied our k-best algorithm to the problem of making a series of incremental decisions. For a sequence of decisions, we can't guarantee that we will be able to see the bottom of the problem-space tree before the next decision, and thus the completion-cost distribution that we developed for the last incremental decision problem no longer accurately reflects the situation that will exist for the next decision. Instead, the distribution of minimum-cost paths that might be traversed below a frontier node will depend on the decision-making algorithm, the exploration algorithm, as well as the factors that affect the last incremental decision. For these reasons, and the fact that an optimal solution to the general incremental decision problem will be at least as complicated as the optimal last incremental decision problem, we have assumed that it is reasonable to model the general incremental decision problem as if it were a last incremental decision problem with one additional level of unexplored problem-space tree below the bottom of the search tree.

To compare MINIMIN and k-best on a series of decisions, we performed a new set of experiments where we fixed the search depth of the branch-and-bound exploration and increased the number of incremental decisions (*i.e.,* the depth of the tree). Figure 7 shows the percent wins (MINIMIN against 7-best) versus the *log* of the tree depth for a fixed search depth of 5 on a binary tree (with 95% confidence intervals). The results are aver-

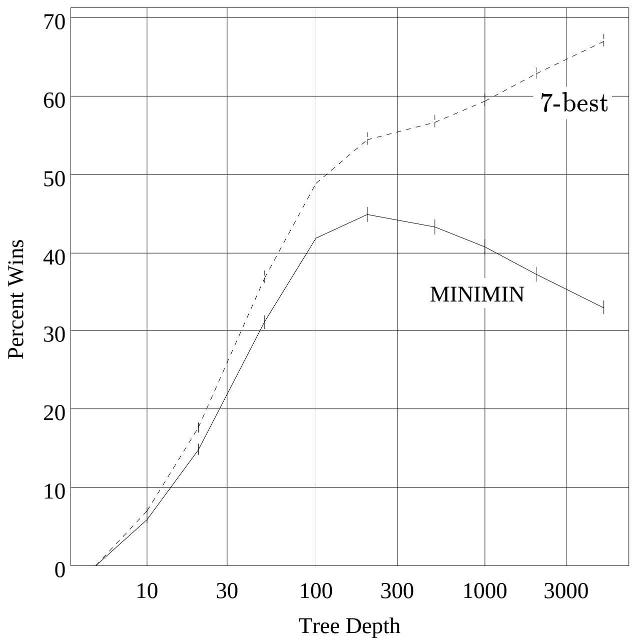


Figure 7: Percent wins by 7-best and MINIMIN versus tree depth.

aged over 10,000 trials, and show that as the number of decisions increases (*i.e.,* the tree depth), the percentage of the trials that each algorithm wins initially grows. This makes sense because the number of opportunities for the algorithms to make a different choice increases with the number of decisions. What we didn't expect is that for more than about 300 decisions, the percentage of the trials in which k-best produces a lower-cost solution than MINIMIN continues to grow, at the expense of the percentage of the trials in which MINIMIN produces a lower-cost solution. This result clearly shows that k-best makes better quality decision on average than MINIMIN.

8 Related Work

The results presented here is an extension of our previous work reported in [Pemberton and Korf, 1994]. Our initial work was motivated by Mutchler's analysis of how to spend scarce search resources to find a complete solution path [Mutchler, 1986]. Our work is also related to Russell and Wefald's work on DTA* [Russell and Wefald, 1991], although they have not directly addressed the last incremental decision problem. Eric Horvitz [Horvitz, 1990] has also investigated the problem of reasoning under resource constraints, which he called ***flexible computation.*** The MINIMIN decision method was initially employed by RTA* [Korf, 1990] and is a special case of the minimax decision rule that is widely used in game tree evaluation [Shannon, 1950].

The main difference between anytime algorithms [Dean and Boddy, 1988] and our real-time incremental search algorithms is that anytime algorithms address what we refer to as the complete solution problem, whereas we have focussed on the incremental decision-making problem. For example, in the random tree model, an anytime algorithm would generate a complete root-to-leaf path, whereas k-best focuses its attention on improving the quality of the next decision. Thus, instead of generating a complete root-to-leaf path all at once, k-best generates the root-to-leaf path one step at a time, while interleaving computation and execution.

In some sense, we can view the computation for each incremental decision as an anytime decision problem. Thus the difference between incremental search algorithms and anytime algorithms is in the way that the real-time search problem is formulated. Anytime algorithms try to find the *best complete solution* under a time constraint, whereas real-time incremental search algorithms try to find the *best next decision* under a time constraint.

9 Conclusions

We have presented k-best, which is a new method for making incremental search decisions. We have shown that the expected error of k-best on the last incremental decision problem with one unexplored level remaining is at most equal to the expected cost of the minimum of kb random edge costs, where k is the number of frontier nodes considered, and b is the branching factor of the nodes in the unexplored level of the tree. k-best is both an approximation of the optimal decision method, and an extension of MINIMIN. Our experimental results show that k-best decisions are slightly better quality than MINIMIN decisions on average, and that this improvement does add up over long sequences of decisions. Since MINIMIN is easy to implement, and is typically very efficient to execute, we recommend it as the first choice for real-time incremental decision problems. When the resource constraint is on the depth of the search rather than the time spent searching (*e.g.,* limited sensor range, unknown future job-scheduling requirements), then k-best provides a useful way to incorporate additional information about future decisions in order to improve the overall quality of the decision sequence.

Acknowledgements

This research was supported by NSF grant #IRI-9119825, the Air Force Office of Scientific Research under contract 92-0693, ARPA/Rome Labs under contracts F30602-91-C-0036 and F30602-93-C-00031, a grant from Rockwell International, and an equipment grant from Hewlett-Packard. We would like to acknowledge helpful discussions with Rich Korf, Weixiong Zhang, and the members of CIRL. Thanks to William Cheng for *tgif*, David Harrison for *xgraph*, and the Free Software foundation for *gcc*, *gdb*, and *gnuemacs*.

References

[Dean and Boddy, 1988] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings, Seventh National Conference on Artificial Intelligence (AAAI-88), St. Paul, MN*, pages 49–54, Palo Alto, CA, 1988.

[Horvitz, 1990] Eric J. Horvitz. *Computation and Action Under Bounded Resources.* PhD thesis, Stanford University, December 1990.

[Korf, 1990] Richard E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2–3):189–211, March 1990.

[Mutchler, 1986] David Mutchler. Optimal allocation of very limited search resources. In *Proceedings, Fifth National Conference on Artificial Intelligence (AAAI-86), Philadelphia, PA*, pages 467–471, Palo Alto, CA, 1986.

[Pemberton and Korf, 1994] Joseph C. Pemberton and Richard E. Korf. Incremental search algorithms for real-time decision making. In *Proceedings, Second International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, pages 140–145, 1994.

[Russell and Wefald, 1991] Stuart Russell and Eric Wefald. *Do the Right Thing.* MIT Press, Cambridge, MA, 1991.

[Shannon, 1950] C. E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41(7):256–275, 1950.

[Zhang and Korf, 1995] Weixiong Zhang and Richard E. Korf. Performance of linear-space search algorithms. *to appear in Artificial Intelligence*, 1995.