# Action Timing Discretization with Iterative-Refinement

Todd W. Neller*

Department of Computer Science
Gettysburg College
Gettysburg, PA 17325, USA
tneller@gettysburg.edu

**Abstract.** Artificial Intelligence search algorithms search discrete systems. To apply such algorithms to continuous systems, such systems must first be discretized, i.e. approximated as discrete systems. Action-based discretization requires that both action parameters and action timing be discretized. We focus on the problem of action timing discretization.
After describing an $\epsilon$-admissible variant of Korf's recursive best-first search ($\epsilon$-RBFS), we introduce iterative-refinement $\epsilon$-admissible recursive best-first search (IR $\epsilon$-RBFS) which offers significantly better performance for initial time delays between search states over several orders of magnitude. Lack of knowledge of a good time discretization is compensated for by knowledge of a suitable solution cost upper bound.

## 1 Introduction

Artificial Intelligence search algorithms search discrete systems, yet we live and reason in a continuous world. Continuous systems must first be discretized, i.e. approximated as discrete systems, to apply such algorithms. There are two common ways that continuous search problems are discretized: state-based discretization and action-based discretization. State-based discretization becomes infeasible when the state space is highly dimensional. Action-based discretization becomes infeasible when there are too many degrees of freedom. Interestingly, biological high-degree-of-freedom systems are often governed by a much smaller collection of motor primitives [3]. We focus here on action-based discretization.

Action-based discretization consists of two parts: (1) action parameter discretization and (2) action timing discretization, i.e. *how* and *when* to act. For example, consider robot soccer. Search can only sample action parameter continua such as kick force and angle. Similarly, search can only sample infinite possible action timings such as when to kick. The most popular form of discretization is uniform discretization. It is common to sample possible actions and action timings at fixed intervals.

In this paper, we focus on action timing discretization. Experimental evidence of this paper and previous studies [4] suggests that a fixed uniform discretization of time

* The author is grateful to Richard Korf for suggesting the sphere navigation problem, and to the anonymous AAAI and SARA reviewers for good insight and suggestions. This research was done both at the Stanford Knowledge Systems Laboratory with support by NASA Grant NAG2-1337, and at Gettysburg College.

is not advisable for search if one has a desired solution cost upper bound. Rather, a new class of algorithms that dynamically adjust action timing discretization can yield significant performance improvements over static action timing discretization.

Iterative-refinement algorithms use a simple means of dynamically adjusting the time interval between search states. This paper presents the results of an empirical study of the performance of different search algorithms as one varies the initial time interval between search states. We formalize our generalization of search, present our chosen class of problems, describe the algorithms compared, and present the experimental results.

The key contributions of this work are experimental insight into the importance of searching with dynamic time discretization, and two new iterative-refinement algorithms, one of which exceeds the performance of $\epsilon$-RBFS across more than four orders of magnitude of the initial time delay between states.

## 2 Search Problem Generalization

Henceforth, we will assume that the action discretization, i.e. which action parameters are sampled, is already given. From the perspective of the search algorithm, the action discretization is static, i.e. cannot be varied by the algorithm. However, action timing discretization is dynamic, i.e. the search algorithm can vary the action timing discretization. For this reason, we will call such searches "SADAT searches" as they have Static Action and Dynamic Action Timing discretization.

We formalize the SADAT search problem as the quadruple:

$$\{S, s_0, A, G\}$$

where

- $S$ is the state space,
- $s_0 \in S$ is the initial state,
- $A = \{a_1, \ldots, a_n\}$ is a finite set of action functions $a_i : S \times \Re^+ \rightarrow S \times \Re$, mapping a state and a positive time duration to a successor state and a transition cost, and
- $G \subset S$ is the set of goal states.

The important difference between this and classical search formulations is the generalization of actions (i.e. operators). Rather than mapping a state to a new state and the associated cost of the action, we additionally take a time duration parameter specifying how much time passes between the state and its successor.

A goal path can be specified as a sequence of action-duration pairs that evolve the initial state to a goal state. The cost of a path is the sum of all transition costs. Given this generalization, the state space is generally infinite, and the optimal path is generally only approximable through a sampling of possible paths through the state space.

## 3 Sphere Navigation Problem

Since SADAT search algorithms will generally only be able to approximate optimal solutions, it is helpful to test them on problems with known optimal solutions. Richard

Korf proposed the problem of navigation between two points on the surface of a sphere as a simple benchmark with a known optimal solution.[1] Our version of the problem is given here.

The shortest path between two points on a sphere is along the great-circle path. Consider the circle formed by the intersection of a sphere and a plane through two points on the surface of the sphere and the center of the sphere. The *great-circle path* between the two points is the shorter part of this circle between the two points. The *great-circle distance* is the length of this path.

The state space $S$ is the set of all positions and headings on the surface of a unit sphere along with all nonnegative time durations for travel. Essentially, we encode path cost (i.e. time) in the state to facilitate definition of $G$. The initial state $s_0$ is arbitrarily chosen to have position (1,0,0) and velocity (0,1,0) in spherical coordinates, with no time elapsed initially.

The action $a_i \in A, 0 \leq i \leq 7$ takes a state and time duration, and returns a new state and the same time duration (i.e. cost = time). The new state is the result of changing the heading $i * \pi/4$ radians and traveling with unit velocity at that heading for the given time duration on the surface of the unit sphere. If the position reaches a goal state, the system stops evolving (and incurring cost).

The set of goal states $G$ includes all states that are both (1) within $\epsilon_d$ great-circle distance from a given position $p_g$, and (2) within $\epsilon_t$ time units of the optimal great-circle duration to reach such positions. Put differently, the first requirement defines the size and location of the destination, and the second requirement defines how directly the destination must be reached. Position $p_g$ is chosen at random from all possible positions on the unit sphere with all positions being equiprobable.

If $d$ is the great-circle distance between (1,0,0) and $p_g$, then the optimal time to reach a goal position at unit velocity is $d - \epsilon_d$. Then the solution cost upper bound is $d - \epsilon_d + \epsilon_t$. For any position, the great-circle distance between that position and $p_g$ minus $\epsilon_d$ is the optimal time to goal at unit velocity. This is used as the admissible heuristic function $h$ for all heuristic search.

## 4 Algorithms

In this section we describe the four algorithms used in our experiments. The first pair use fixed time intervals between states. The second pair dynamically refine time intervals between states. The first algorithm, $\epsilon$-admissible iterative-deepening A$^*$, features an improvement over the standard description. Following that we describe an $\epsilon$-admissible variant of recursive best-first search and two novel iterative-refinement algorithms.

### 4.1 $\epsilon$-Admissible Iterative-Deepening A$^*$

$\epsilon$-admissible iterative-deepening A$^*$ search, here called $\epsilon$-IDA$^*$, is a version of IDA$^*$ [1] where the $f$-cost limit is increased "by a fixed amount $\epsilon$ on each iteration, so that the total number of iterations is proportional to $1/\epsilon$. This can reduce the search cost, at the expense of returning solutions that can be worse than optimal by at most $\epsilon$." [5].

[1] Personal communication, 23 May 2001.

Actually, our implementation is an improvement on $\epsilon$-IDA$^*$ as described above. If $\Delta f$ is the difference between (1) the minimum $f$-value of all nodes beyond the current search contour, and (2) the current $f$-cost limit, then the $f$-cost limit is increased by $\Delta f + \epsilon$. ($\Delta f$ is the increase that would occur in IDA$^*$.) This improvement is significant in cases where $f$-cost limit changes between iterations can significantly exceed $\epsilon$.

It is important to note that when we commit to an action timing discretization, the $\epsilon$-admissibility of search is relative to the optimal solution of this discretization rather than the optimal solution of the original continuous-time SADAT search problem.

### 4.2 $\epsilon$-Admissible Recursive Best-First Search

$\epsilon$-admissible recursive best-first search, here called $\epsilon$-RBFS, is an $\epsilon$-admissible variant of recursive best-first search that follows the description of [2, §7.3] without further search after a solution is found. As with our implementation of $\epsilon$-IDA$^*$, local search bounds increase by at least $\epsilon$ (when not limited by `B`) to reduce redundant search.

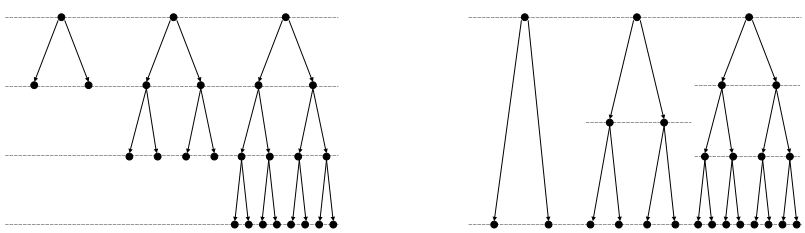
In Korf's style of pseudocode, $\epsilon$-RBFS is as follows:

```
eRBFS (node: N, value: F(N), bound: B)
IF f(N)>B, RETURN f(N)
IF N is a goal, EXIT algorithm
IF N has no children, RETURN infinity
FOR each child Ni of N,
  IF f(N)<F(N), F[i] := MAX(F(N),f(Ni))
  ELSE F[i] := f(Ni)
sort Ni and F[i] in increasing order of F[i]
IF only one child, F[2] := infinity
WHILE (F[1] <= B and F[1] < infinity)
  F[1] := eRBFS(N1, F[1], MIN(B, F[2] + epsilon))
  insert Ni and F[1] in sorted order
RETURN F[1]
```

The difference between RBFS and $\epsilon$-RBFS is in the computation of the bound for the recursive call. In RBFS, this is computed as `MIN(B, F[2])` whereas in $\epsilon$-RBFS, this is computed as `MIN(B, F[2] + epsilon)`. `F[1]` and `F[2]` are the lowest and second-lowest stored costs of the children, respectively. A correctness proof of $\epsilon$-RBFS follows the structure of Korf's RBFS correctness proof [2] with minor modifications.

The algorithm's initial call parameters are the root node $r$, $f(r)$, and $\infty$. Actually, both RBFS and $\epsilon$-RBFS can be given a finite bound $b$ if one wishes to restrict search for solutions with a cost of no greater than $b$ and uses an admissible heuristic function. If no solution is found, the algorithm will return the $f$-value of the minimum open search node beyond the search contour of $b$.

In the context of SADAT search problems, both $\epsilon$-IDA$^*$ and $\epsilon$-RBFS assume a fixed time interval between a node and its child. The following two algorithms do not.



(a) Iterative-deepening DFS. (b) Iterative-refinement DFS.

**Fig. 1.** Iterative search methods.

### 4.3 Iterative-Refinement $\epsilon$-RBFS

Iterative-refinement [4] is perhaps best described in comparison to iterative-deepening. Iterative-deepening depth-first search (Figure 1(a)) provides both the linear memory complexity benefit of depth-first search and the minimum-length solution-path benefit of breadth-first search at the cost of node re-expansion. Such re-expansion costs are generally dominated by the cost of the final iteration because of the exponential nature of search time complexity.

Iterative-refinement depth-first search (Figure 1(b)) can be likened to an iterative-deepening search to a fixed time-horizon. In classical search problems, time is not an issue. Actions lead from states to other states. When we generalize such problems to include time, we then have the choice of how much time passes between search states. Assuming that the vertical time interval in Figure 1(b) is $\Delta t$, we perform successive searches with delays $\Delta t$, $\Delta t/2$, $\Delta t/3$, ... until a goal path is found.

Iterative-deepening addresses our lack of knowledge concerning the proper depth of search. Similarly, iterative-refinement addresses our lack of knowledge concerning the proper time discretization of search. Iterative-deepening performs successive searches that grow exponentially in time complexity. The complexity of previous unsuccessful iterations is generally dominated by that of the final successful iteration. The same is true for iterative-refinement.

However, the concept of iterative-refinement is not limited to the use of depth-first search. Other algorithms such as $\epsilon$-RBFS may be used as well. In general, for each iteration of an iterative-refinement search, a level of (perhaps adaptive) time-discretization granularity is chosen for search and an upper bound on the solution cost is given. If the iteration finds a solution within this cost bound, the algorithm terminates with success. Otherwise, a finer level of time-discretization granularity is chosen, and search is repeated. Search is successively refined with respect to time granularity until a solution is found.

Iterative-Refinement $\epsilon$-RBFS is one instance of such search. The algorithm can be simply described as follows:

```
IReRBFS (node: N, bound: B, initDelay: DT)
FOR I = 1 to infinity
  Fix the time delay between states at DT/I
  eRBFS(N, f(N), B)
  IF eRBFS exited with success, EXIT algorithm
```

Iterative-Refinement $\epsilon$-RBFS does not search to a fixed time-horizon. Rather, each iteration searches within a search contour bounded by $B$. Successive iterations search to the same bound, but with finer temporal detail. `DT/I` is assigned to a global variable governing the time interval between successive states in search.

### 4.4 Iterative-Refinement DFS

The algorithm for Iterative-Refinement DFS is given as follows:

```
IRDFS (node: N, bound: B, initDelay: DT)
FOR I = 1 to infinity
  Fix the time delay between states at DT/I
  DFS-NOUB(N, f(N), B)
  IF DFS-NOUB exited with success, EXIT algorithm
```

Our depth-first search implementation DFS-NOUB uses a node ordering (NO) heuristic and has a path cost upper-bound (UB). The node-ordering heuristic is as usual: Nodes are expanded in increasing order of $f$-value. Nodes are not expanded that exceed a given cost upper bound. Assuming admissibility of the heuristic function $h$, no solutions within the cost upper-bound will be pruned from search.

## 5 Experimental Results

In these experiments, we vary only the initial time delay $\Delta t$ between search states and observe the performance of the algorithms we have described. For $\epsilon$-IDA$^*$ and $\epsilon$-RBFS, the initial $\Delta t$ is the only $\Delta t$ for search. The iterative-refinement algorithms search using the harmonic refinement sequence $\Delta t$, $\Delta t/2$, $\Delta t/3$, . . ., and are limited to 1000 refinement iterations. $\epsilon$-admissible searches were performed with $\epsilon = .1$.

Experimental results for success rates of search are summarized in Figure 2. Each point represents 500 trials over a fixed, random set of sphere navigation problems with $\epsilon_d = .0001$ and $\epsilon_t$ computed as 10% of the optimal time. Thus, the target size for each problem is the same, but the varying requirement for solution quality means that different delays will be appropriate for different search problems. Search was terminated after 10 seconds, so the success rate is the fraction of time a solution was found within the allotted time and refinement iterations.

In this empirical study, means and 90% confidence intervals for the means were computed with 10000 bootstrap resamples.

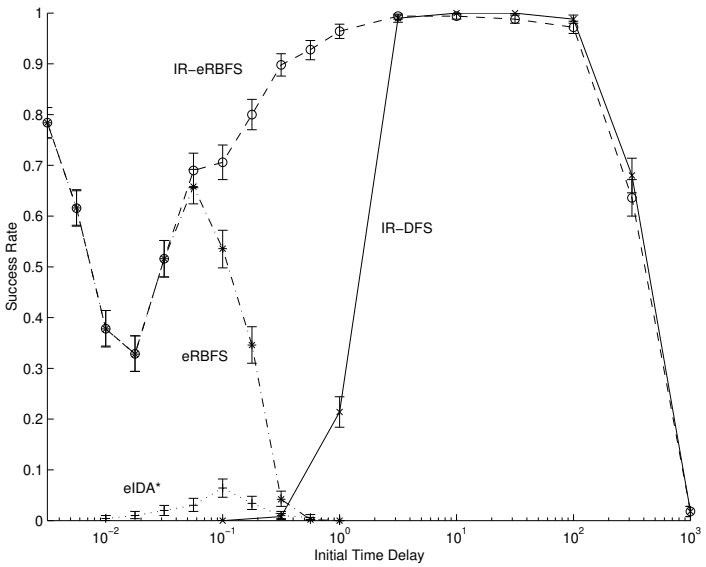



**Fig. 2.** Effect of varying initial $\Delta t$.

Let us first compare the performance of iterative-refinement (IR) $\epsilon$-RBFS and $\epsilon$-RBFS. To the left of the graph, where the initial $\Delta t_0$ is small, the two algorithms have identical behavior. This region of the graph indicates conditions under which a solution is found within 10 seconds on the first iteration or not at all. There is no iterative-refinement in this region; the time complexity of the first iteration leaves no time for another.

At about $\Delta t_0 = .1$, we observe that IR $\epsilon$-RBFS begins to have a significantly greater success rate than $\epsilon$-RBFS. At this point, the time complexity of search allows for multiple iterations, and thus we begin to see the benefits of iterative-refinement.

Continuing to the right with greater initial $\Delta t_0$, IR $\epsilon$-RBFS nears a 100% success rate. At this point, the distribution of $\Delta t$'s over different iterations allows IR $\epsilon$-RBFS to reliably find a solution within the time constraints. We can see the distribution of $\Delta t$'s that most likely yield solutions from the behavior of $\epsilon$-RBFS.

Where the success rate of IR $\epsilon$-RBFS begins to fall, the distribution of first 1000 $\Delta t$'s begins to fall outside of the region where solutions can be found. With our refinement limit of 1000, the last iteration uses a minimal $\Delta t = \Delta t_0/1000$. The highest $\Delta t_0$ trials fail not because time runs out. Rather, the iteration limit is reached. However, even with a greater refinement limit, we would eventually reach a $\Delta t_0$ where the iterative search cost incurred on the way to the good $\Delta t$ range would exceed 10 seconds.

Comparing IR $\epsilon$-RBFS with IR DFS, we first note that there is little difference between the two for large $\Delta t_0$. For mid-to-low-range $\Delta t_0$ values, however, we begin to see the efficiency of $\epsilon$-RBFS over DFS with node ordering as the first iteration with a

solution path presents a more computationally costly search. Without a perfect heuristic where complex search is necessary, $\epsilon$-RBFS shows its strength relative to DFS. Rarely will problems be so unconstrained and offer such an easy heuristic as this benchmark problem, so IR $\epsilon$-RBFS will be generally be better suited for all but the simplest search problems.

In summary, iterative-refinement algorithms are statistically the same as or superior to the other searches over the range of $\Delta t_0$ values tested. IR $\epsilon$-RBFS offers the greatest average success rate across all $\Delta t_0$. With respect to $\epsilon$-RBFS, IR $\epsilon$-RBFS offers significantly better performance for $\Delta t_0$ spanning more than four orders of magnitude. These findings are in agreement with previous empirical studies concerning a submarine detection avoidance problem [4].

## 6 Conclusions

This empirical study concerning sphere navigation provides insight into the importance of searching with dynamic time discretization. Iterative-refinement algorithms are given an initial time delay $\Delta t_0$ between search states and a solution cost upper bound. Such algorithms iteratively search to this bound with successively smaller $\Delta t$ until a solution is found.

Iterative-refinement $\epsilon$-admissible recursive best-first search (IR $\epsilon$-RBFS) was shown to be similar to or superior to all other searches studied for $\Delta t_0$ spanning over five orders of magnitude. With respect to $\epsilon$-RBFS (without iterative-refinement), a new $\epsilon$-admissible variant of Korf's recursive best-first search, IR $\epsilon$-RBFS offers significantly better performance for $\Delta t_0$ spanning over four orders of magnitude.

Iterative-refinement algorithms are important for search problems where reasonable values for $\Delta t$ are (1) unknown or (2) known and one wishes to find a solution more quickly and reliably. The key tradeoff is that of knowledge. Lack of knowledge of a good time discretization is compensated for by knowledge of a suitable solution cost upper bound. If one knows a suitable solution cost upper bound for a problem where continuous time is relevant, an iterative-refinement algorithm such as IR $\epsilon$-RBFS is recommended.

## References


1. Richard E. Korf. Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.
2. Richard E. Korf. Linear-space best-first search. *Artificial Intelligence*, 62:41–78, 1993.
3. Maja J. Mataric. Sensory-motor primitives as a basis for imitation: Linking perception to action and biology to robotics. In C. Nehaniv and K. Dautenhahn, editors, *Imitation in Animals and Artifacts*, Cambridge, MA, USA, 2000. MIT Press. See also USC technical report IRIS-99-377.
4. Todd W. Neller. *Simulation-Based Search for Hybrid System Control and Analysis*. PhD thesis, Stanford University, Palo Alto, California, USA, June 2000. Available as Stanford Knowledge Systems Laboratory technical report KSL-00-15 at www.ksl.stanford.edu.
5. Stuart Russell and Peter Norvig. *Artificial Intelligence: a modern approach*. Prentice Hall, Upper Saddle River, NJ, USA, 1995.