# Information-Based Alpha-Beta Search and the Homicidal Chauffeur

Todd W. Neller*

Gettysburg College
tneller@gettysburg.edu

**Abstract.** The standard means of applying a discrete search to a continuous or hybrid system is the uniform discretization of control actions and action timing. Such discretization is fixed a priori and does not allow search to benefit from information gained at run-time. This paper introduces Information-Based Alpha-Beta Search, a new algorithm that preserves and benefits from the continuous or hybrid nature of the search. In a novel merging of alpha-beta game-tree search and information-based optimization, Information-Based Alpha-Beta Search makes trajectory-sampling decisions dynamically based on the maximum-likelihood of search pruning. The result is a search algorithm which, while incurring higher computational overhead for the optimization, manages to so increase the quality of the sampling, that the net effect is a significant increase in performance. We present a new piecewise-parabolic variant of the algorithm and provide empirical evidence of its performance relative to random and uniform discretizations in the context of a variant of the homicidal chauffeur game.

## 1 Motivation

Locally optimal ("greedy") control decisions are not necessarily globally optimal control decisions for many systems. Some sort of "lookahead" facility is necessary to derive optimal or approximately optimal control policies. Tree-based search techniques have proven powerful for lookahead in some complex discrete systems, but are not easily applied to continuous or hybrid systems. One reason is that an a priori discretization of control actions is necessary. A good discretization is often not easy to discern.

We introduce a new technique which performs discretization dynamically during search according to maximum-likelihood reasoning. This technique lifts the burden of discretization from the control engineer and, as we shall see, can outperform standard discretization approaches. Thus the primary contribution of this paper is a technique that enables both easier and better use of search techniques for lookahead in continuous and hybrid control systems. The remainder of this section relates these thoughts in greater detail.

* This work was done both at the Stanford Knowledge Systems Laboratory with support by NASA Grant NAG2-1337, and at Gettysburg College. Author's address: Department of Computer Science, Gettysburg College, Gettysburg, PA, 17325, USA.

Many optimal control techniques seek to optimize some performance measure by steepest ascent or descent of the measure's gradient. While locally optimal, such "greedy" control actions are not necessarily optimal with respect to finite or infinite time-horizons. In discrete systems, this is easily illustrated with the game of chess. If the performance measure is based on material advantage, the player who employs a greedy policy falls victim to a clever opponent's sacrifices where short-term material advantage is followed by a stronger counter-attack. The player with foresight looks beyond the immediate gain to the ramifications of that gain.

For continuous systems, we may imagine a pursuit-evasion game in which the evader wants to avoid prolonged close proximity with a slightly faster pursuer. The greedy evader immediately in front of the pursuer will seek to maintain as much distance from the pursuer as possible by running away in the same direction of pursuit. The evader with foresight looks beyond the immediate situation to see that charging towards the pursuer and then breaking to the side will put it behind the pursuer. A temporary increase in proximity makes possible a significant decrease in proximity as the pursuer struggles to turn and renew pursuit. For hybrid systems, we may imagine a pursuit-evasion game with environmental constraints (e.g. dangerous airspace, terrain features, etc.).

From these examples, we can see the performance benefit of global "lookahead" versus local "greedy" policies. In situations where we have a small, finite state space (or a continuous or hybrid state space approximable as such), we can apply dynamic programming to compute a globally optimal control policy. However, such situations are rare in practice. With an increase in the dimensionality of a state space, either the necessary memory or the granularity of the discretization grows *exponentially*, making such approaches infeasable.

However, in the case where (1) an approximate continuous or hybrid simulation model can be derived for the system, (2) limited lookahead with this model enables better performance, and (3) a sampling of the *action space* (i.e. space of possible control input vectors) is adequate to inform a good decision, we can benefit from tree-search techniques. Such techniques sample possible *sequences of control actions* and use that sample to approximate optimal behavior.

From any given state, a sample of possible control actions and an assumed delay before the next action yields a sample of possible future states. These future states may in turn be used to generate a sample of further future states until we reach a search stopping condition. The result is a discrete search tree which can be used to approximate the utility of a sample of control actions. The best of these actions in turn approximates the optimal control action.

One drawback to standard search approaches is that they assume that the actions (i.e. control inputs) are discretized *a priori* into a finite set of actions. Finding a good discretization of control inputs is often not a trivial task. By committing to a fixed or *static* action discretization, standard approaches cannot *dynamically* adapt the sampling at run-time in order to improve the quality of the discretization.

To illustrate this, we turn again to the pursuit-evasion example. Suppose the evader samples a few possible directions for evasion and looks ahead at branching trees of possible trajectories in these directions. Lookahead indicates that some directions are especially good. In greatest likelihood, the evader will find better control actions closer to those sampled actions which appear best. This intuitive sort of dynamic sampling behavior is *not possible in pre-existing search formalisms.*

This paper introduces a new formalism for hybrid games, and a new and powerful technique which works with a subset of such games: Information-Based Alpha-Beta Search. Not only does the algorithm preserve the continuous or hybrid nature of control actions in search, but it also reaps performance benefits from this preservation.

## 2 Alpha-Beta Search

In this section, we describe the game-tree search technique commonly known as "alpha-beta search" or "minimax search with alpha-beta pruning". Before introducing basic concepts of game-tree search, it is important to motivate the use of such algorithms and place them in context.

At heart, artificial intelligence (AI) search algorithms are a form of *optimization.* The primary distinction between search algorithms and other forms of optimization is that search algorithms optimize utility (i.e. performance index) over *sequences of actions* (i.e. control vector sequences). Unlike optimal control formulations of multi-stage problems, AI search does not assume a finite number of stages. However, by computational necessity, the search is often biased towards shorter sequences of actions. Any process which can benefit from examining the ramifications of short sequences of control decisions can benefit from this form of optimization.

A classical AI game-tree search problem definition concerns discrete system dynamics and consists of three components: (1) an *initial state* (including system state and current player), (2) a *successor function* (a mapping from states to sets of possible future states[1], and (3) a *utility function* (a mapping from states to the value of being in that state). (1) and (2) define the state space. An alternate formalism we use breaks component (2) into two parts: (2a) an *operator function* (a mapping from states to sets of legal actions), and (2b) a mapping from states and actions to resulting states. We prefer this formalism as it is often easier to define possible states indirectly by defining possible actions.

A classical AI game-tree search algorithm typically works as follows: We begin with a set of unsearched states consisting of the initial state (1). Iteratively, we choose and remove a state from our unsearched state set, evaluate its utility, generate possible future states, and add these to the unsearched state set. At any time, the search algorithm has searched a portion of the search tree and may modify its valuation of actions by propogating utilities up the tree. Since states

[1] When a state maps to an empty set, the state is said to be *terminal.*

can be repeated in the tree, a specific state at a position in the tree is called a *node*. Different algorithms will vary in how they choose the next unsearched state. Since most search trees cannot be searched exhaustively, different algorithms also have different stopping conditions.

Although the *minimax* algorithm can be used to evaluate a game search tree (i.e. game tree), this process can be made significantly more efficient for zero-sum games through a technique called *alpha-beta pruning*. During search, if one can prove through zero-sum constraints that rational play will never lead to a node, search from that node can be "pruned" (i.e. discontinued).

Since we assume a zero-sum game, we can have a single utility associated with each node. One player (MAX) seeks to maximize this utility, and the other (MIN) seeks to minimize it. Assume without loss of generality that at the initial search node (or *root*), it is MAX's turn. At each search node, we keep track of two values $\alpha$ and $\beta$. Along the path (i.e. sequence of actions) from the root to a node, $\alpha$ is the best score for MAX, and $\beta$ is the best score for MIN. Put another way, play along that path guarantees a score of *at least* $\alpha$ and *at most* $\beta$. At the root node, these are $-\infty$ and $\infty$ respectively.
To compute MAX-Value($s$, $\alpha$, $\beta$):

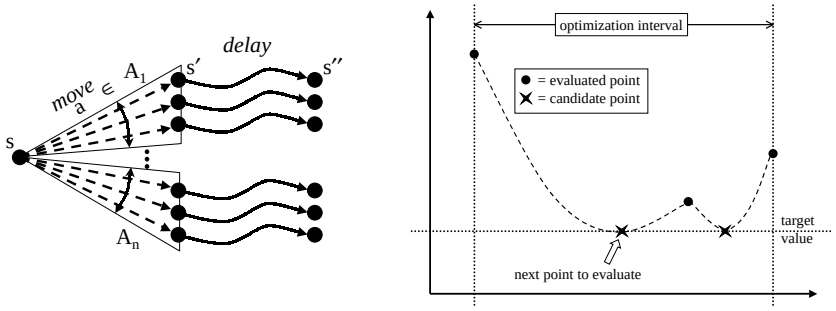
1. If state $s$ is terminal or triggers a terminating condition of search, then return the utility of $s$.
2. For each legal action $a$ of MAX in state $s$:
   (a) Let state $s'$ be the result of action $a$ in state $s$.
   (b) Set $\alpha$ equal to the greater of $\alpha$ and MIN-Value($s'$, $\alpha$, $\beta$).
   (c) If $\alpha$ is greater than or equal to $\beta$, then "prune" search and return $\beta$.
3. Return $\alpha$.

To compute MIN-Value($s$, $\alpha$, $\beta$):

1. If state $s$ is terminal or triggers a terminating condition of search, then return the utility of $s$.
2. For each legal action $a$ of MIN in state $s$:
   (a) Let state $s'$ be the result of action $a$ in state $s$.
   (b) Set $\beta$ equal to the lesser of $\beta$ and MAX-Value($s'$, $\alpha$, $\beta$).
   (c) If $\beta$ is less than or equal to $\alpha$, then "prune" search and return $\alpha$.
3. Return $\beta$.

A player MAX will take the action corresponding to the successor state with the greatest MIN-Value. Although other terminating conditions of search are often employed, we will use the simplest throughout: a limitation on the *depth* of search – the maximum length of an action sequence. The number of possible actions in a state is called the *breadth* of search.

This discrete search algorithm assumes a discrete dynamical system. For any state, there are a finite number of actions which result in a finite number of successor states. In order to apply such an algorithm to a continuous or hybrid dynamical system, we must first approximate the system as a discrete system through a process called *discretization*.




(a) DASAT Hybrid System Game

(b) Parabolic Information-Based Optimization

**Fig. 1.**

There are two types of discretization which must occur: (1) discretization of actions, and (2) discretization of action timing. For (1), it is common practice to discretize a continuum of possible control inputs with a uniform sampling. For (2), it is common to pick a fixed time interval $\Delta t$ for which the control inputs hold as simulation advances to the next successor state.

It is often not obvious what constitutes a good decision for either type of discretization. Worse yet, by committing to such a discretization before search, it is not possible to reap the benefit of information obtained during search to form a better discretization. In the next section, we describe a more general game formalism which assumes a fixed (static) discretization for action timing, but allows an adaptive (dynamic) discretization for control actions.

## 3 DASAT Hybrid System Game

In this section, we present a formalism for hybrid system games called *DASAT hybrid system games*, and a subclass of such games that can be searched with our new techniques. DASAT stands for Dynamic Action discretization with Static Action Timing discretization.

Formally, a *DASAT hybrid system game* is defined as a 7-tuple

$$\{S, s_0, \mathbf{A}, p, l, m, d\}$$

where

- $S$ is the hybrid state space[1] with a finite number of finite discrete variable domains, and a finite-dimensional continuous space,
- $s_0 \in S$ is the initial state,

- $\mathbf{A}$ is a finite set $\{A_1, \ldots, A_n\}$ of continuous action regions indexed $\{1, \ldots, n\}$,
- $p$ is the number of players,
- $l : S \times \{1, \ldots, p\} \rightarrow \mathbf{A}'$ where $\mathbf{A}' \subset \mathbf{A}$ is a *legal move* function mapping from a state and player number to a finite set of legal continuous action regions which contain points representing all legal actions that may be executed in that state by that player,
- $m : S \times \mathbf{a}^p \rightarrow S \times \Re^p$ is a *move* function mapping from a state and simultaneous player actions (region index, region point pairs) to a resulting state and the utility of the combined actions for each player,
- $d : S \rightarrow S \times \Re^p$ is a *delay* function mapping from a state to the resulting state and the utility of the trajectory segment for each player. This delay governs the hybrid evolution of the system through time between moves.

This formalism is visualized in Figure 1(a). For any state $s$ in our hybrid state space $S$, there are a finite number of action regions $\{A_1, \ldots, A_n\}$. Each region can be thought of as a closed, contiguous set of allowable control input vectors. Let $a$ be an action (e.g. a single control input vector) from one of these action regions. The *move* function maps $s$ and $a$ to $s'$, applying an instantaneous transition to system state, and computing the associated change in utility. The *delay* function maps $s'$ to $s''$, simulating the evolution of the hybrid system over some time interval $\Delta t \geq 0$. Overall, the system state evolves through both moments of action (moves) and periods of inaction (delays).

The total utility of any finite trajectory is computed as the sum of the trajectory move and delay utilities. In this time-invariant formalism, time can easily be encoded in a continuous clock variable, and time variant behavior can thus be easily achieved.

This formalism is very general in both its dynamics and game-theoretical form. Indeed, in the latter respect, it is too general for the application of alpha-beta search. Two restrictions are necessary to define a class of DASAT hybrid system games for which the techniques of this paper are applicable: (1) The zero-sum constraint of the player score function must be preserved in the *move* and *delay* functions, and (2) a single player must be allowed to move at a time. For (2), we can include the current player in the state, and have the *legal move* function only allow a "null" move for all other players. An example of these restrictions in practice is presented in the following section.

## 4 Homicidal Chauffeur Game Variant

In this section we present a variant of the *homicidal chauffeur game*[2, 3]. The differences in this variation are (1) fixed time delays between actions, (2) a different cost function, and (3) game theoretic information structure (i.e. who knows what when). Regarding (1), the homicidal chauffeur game is a *differential game*[2]. In our variant, we have a fixed discretization of action timing. Regarding (2), the homicidal chauffeur game has only two player utilities representing capture and non-capture. In our variant, utility is calculated by integrating inverse distance of a trajectory to the origin (pursuer) over time. Regarding (3),

the homicidal Chauffeur game as a differential game has both players acting at once. In our variant, players take actions in turn with complete knowledge of previous actions.

The homicidal chauffeur game is a pursuit-evasion game in which a chauffeur in a circular car with limited turning radius seeks to hit a slower pedestrian who can change direction instantly. Speed and maneuverability are asymmetric. In our variant, the object of the pursuer/evader is to minimize/maximize their inverse distance integrated over time.

The pursuer can choose a value $u^1 \in [-1, 1]$ where -1 and 1 are extreme left and right turns respectively. The evader chooses any angle $u^2$. The pursuer and evader travel at constant velocities 1 and $v_2$ respectively. Let us assume an $(x_1, x_2)$ coordinate frame relative to the pursuer. The pursuer faces the positive $x_2$-axis. Then the system evolves according to:

$$\dot{x}_1 = -u^1 x_2 + v_2 \sin(u^2), \qquad \dot{x}_2 = -1 + u^1 x_1 + v_2 \cos(u^2)$$

For our information structure, we assume that the pursuer chooses $u^1$ immediately after the evader chooses $u^2$ with knowledge of that choice. There is then a finite, positive delay $\Delta t$ before the evader again chooses $u^2$. The system evolution is piecewise continuous with discrete controller transitions.

This game is not as general as it could be. In particular, we could have multiple ranges of control inputs and hybrid rather than continuous dynamics. This problem was chosen because (1) it is well known, and (2) it is easily visualizable to validate results.

Within our formalism, the hybrid states consists of two continuous variables ($x_1$ and $x_2$) and one discrete variable ($p$, the current player). Continuous action regions are $A_1 = [-1, 1]$, $A_2 = [-\pi, \pi]$, and $A_3$ is a "null move" singleton set with any value. The legal move function maps all states and player $i$ to the singleton action region set $A_i$ when the current player is $i$, and maps to $A_3$ otherwise. Suppose $i$ is the current player: The move function changes $u^i$ according to $a_i$ (the $i$th action of the action vector). The move function also changes the current player and has no utility change. The delay function simulates the system forward $\Delta t_i$ time units. The delay function also has zero-sum utility change (positive for pursuer, negative for evader) according to the inverse distance between players integrated over $\Delta t_i$. After the evader acts and before the pursuer acts we have no delay ($\Delta t_1 = 0$). After the pursuer acts and before the evader acts we have a fixed positive $\Delta t_2$ which was set to 0.1.

We now turn our attention to a new class of algorithms suited to this subclass of DASAT hybrid system games.

## 5 DASAT Alpha-Beta Search

In our new approach to search, we redefine MAX-Value and MIN-Value as follows (changes underlined):
To compute MAX-Value($s$, $\alpha$, $\beta$):

1. If state $s$ is terminal or triggers a terminating condition of search, then return the utility of the trajectory to $s$.
2. While we have not yet reached our breadth limit:
   (a) *Choose a continuous action region $A_i$ and a legal action $a \in A_i$ of MAX in state $s$.*
   (b) Let state $s'$ be the result of the move function of state $s$ and action $a$.
   (c) Let state $s''$ be the result of the delay function of state $s'$.
   (d) Set $\alpha$ equal to the greater of $\alpha$ and MIN-Value($s''$, $\alpha$, $\beta$).
   (e) If $\alpha$ is greater than or equal to $\beta$, then "prune" search and return $\beta$.
3. Return $\alpha$.

To compute MIN-Value($s$, $\alpha$, $\beta$):

1. If state $s$ is terminal or triggers a terminating condition of search, then return the utility of the trajectory to $s$.
2. While we have not yet reached our breadth limit:
   (a) *Choose a continuous action region $A_i$ and a legal action $a \in A_i$ of MIN in state $s$.*
   (b) Let state $s'$ be the result of the move function of state $s$ and action $a$.
   (c) Let state $s''$ be the result of the delay function of state $s'$.
   (d) Set $\beta$ equal to the lesser of $\beta$ and MAX-Value($s''$, $\alpha$, $\beta$).
   (e) If $\beta$ is less than or equal to $\alpha$, then "prune" search and return $\alpha$.
3. Return $\beta$.

Note that there are essentially three major differences between this search and classical alpha-beta search. The first is a result of our formalism for computing utility changes through discrete actions and simulation delays. The second is a result of the fact that there are potentially infinite actions we could choose. The italicized step is the dynamic action discretization step of DASAT Alpha-Beta Search. The third is a result of our alternate formalism for generating state successors in search. We first instantaneously apply an action (e.g. change a control input). We then simulate the effect on the system for a duration fixed by our delay function.

Finally, we note that since this is a two-player zero-sum game, we can simply use the first element of the DASAT game utility vector. Thus player 1 is MAX, and player 2 is MIN.

In this framework, there are many possible techniques for dynamic discretization. The first technique we use for comparison is to dynamically choose a fixed uniform sampling. This serves as a control to tell us what we gain by performing sampling dynamically. The second technique we use is a random sampling. As we shall see, this simple stochastic technique has its advantages as well. The third technique relies on a class of maximum-likelihood-based optimization techniques called information-based optimization. We describe this type of optimization in the next section.

## 6 Information-Based Optimization

We defer the details of our information-based optimization algorithm to [4] and provide only an overview for brevity. Information-based optimization is a type of global optimization that seeks to achieve a *target value* by, at each step of optimization, evaluating the point *most likely* to have that value *given previous evaluations.*

We provide a brief sketch of this optimization in Figure 1(b). Given a set of evaluated points, we wish to choose the next point for evaluation which has greatest likelihood of having our desired target value. In our parabolic variant, we assume that (1) the unknown function is most likely piecewise parabolic, and (2) smaller magnitude constants for quadratic terms are more likely than greater magnitude constants.

Then for single-dimensional functions, we may compute candidate points between prior evaluated points by finding the intersections of the target line and tangential parabolas which pass through adjacent evaluated points. We then choose the candidate point associated with the parabola having a quadratic term of minimum magnitude. In Figure 1(b), we prefer the left candidate as it is associated with the least "steep" parabola. For single-dimensional functions, interval end points are initially chosen.

## 7 Information-Based Alpha-Beta Search

In Section 5, we noted that there are a variety of ways one can dynamically choose which actions to evaluate from a continuum of actions. Commonly, one performs a static uniform discretization. Information-Based Alpha-Beta Search chooses the next action according to information-based optimization, where (1) actions are evaluated by the MAX- or MIN-Values of resulting states, and (2) the target value for optimization is the pruning value $\beta$ or $\alpha$.

Information-based optimization chooses an action to evaluate. A subtree search takes place to evaluate the quality of this action, which is then used in the selection of the next action, and so forth. At every node, Information-Based Alpha-Beta Search uses prior subtree searches from that node to choose the next action most likely to lead to pruning. MAX (MIN) seeks actions which will most-likely have a score of $\beta$ ($\alpha$), the pruning limit. Initially, this pruning limit is infinite and Information-Based Alpha-Beta Search always chooses the middle of the largest gap between evaluated points.

Since an optimization is performed at each search node, there is considerable computational overhead in this approach. However, we will see in the next section that under certain circumstances, this computationally intensive process is compensated for by the quality of the sampling and/or the quality of the pruning.

## 8 Results

The following empirical study supports conclusions of the initial empirical study of Information-Based Alpha-Beta Search in the context of magnetic levitation control [5, ch. 4].

The most important parameters of search are search breadth ($b$) and depth ($d$) as the computational cost is bounded above by a constant times $b^d$. Each experiment varies either breadth or depth of search.

Since there are many possible problem parameters and initial conditions, we arbitrarily chose the following fixed parameters for our experimentation: $\Delta t = 0.1$, $\beta = 0.34$, and $v_2 = 0.8$[2]. In all trials, initial $x_1$ and $x_2$ values were chosen on a uniform $20 \times 20$ grid from $[-2, -2]$ to $[2, 2]$. For each initial state $(x_1, x_2)$ we play two algorithms against each other and compare their resulting scores, times for computation, and number of nodes expanded. More specifically, one algorithm as the evader performs a search, and takes the action recommended by search. The next algorithm then does the same as the pursuer, and this is repeated for fours turns of play. We then repeat the trial with the algorithms switched.

In this empirical study, means and 90% confidence intervals for the means were computed with 10000 bootstrap resamples. Graphs for experiments where breadth or depth are varied have circles representing resampled means and lines representing 90% confidence intervals for the means.

The first important parameter we vary is search breadth, i.e. the number of actions we sample at each node. We hold search depth fixed at 6.

Varying search breadth and comparing Information-Based Alpha-Beta Search with DASAT alpha-beta search with random discretization, we generally see a *time versus score tradeoff*. See Figure 2. Information-based discretization consistently outscores random discretization as we would expect. The magnitude of the score difference lessens with increase in search breadth. Optimization is more costly per node than random selection, but random selection must search nearly exponentially more nodes with the increase in breadth. While random discretization performs faster for small breadths, there is no statistically significant difference for larger breadths.

Varying search breadth and comparing Information-Based Alpha-Beta Search with DASAT alpha-beta search with uniform discretization, we generally see the *same scores but with nearly exponential savings in time* as the breadth increases. See Figure 3. For almost all breadths, we see no significant statistical difference in score. However, information-based discretization results in significantly greater pruning. This pruning more than compensates for the computational cost of the optimization, and yields nearly exponential savings in time with the increase in search breadth.

To understand why information-based and uniform discretizations yield such similar scores, we review the behavior of information-based optimization at the

[2] The choice of $\beta$ with respect to $v_2$ is only significant in the context of the original homicidal chauffeur game.

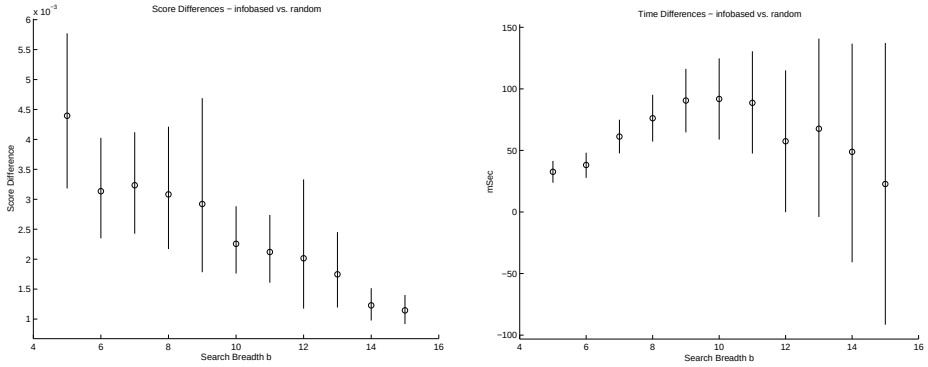



**Fig. 2.** Score and time difference confidence intervals for information-based versus random discretization with varying search breadth

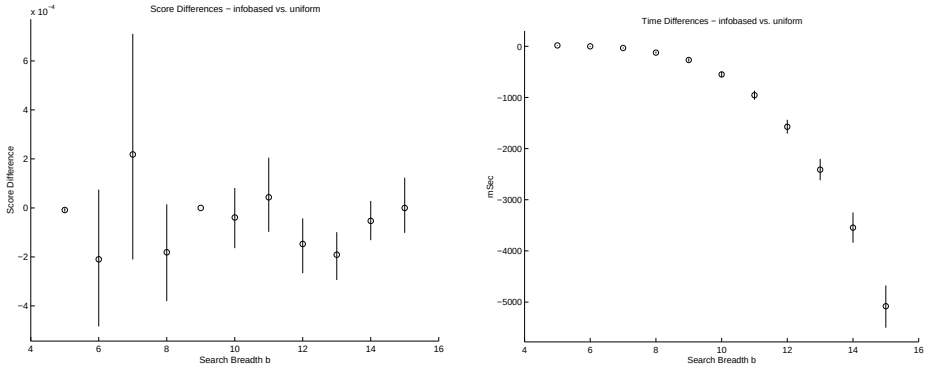



**Fig. 3.** Score and time difference confidence intervals for information-based versus uniform discretization with varying search breadth

start of search. Search starts with $\alpha = -\infty$ and $\beta = \infty$, resulting in a sampling where the next point is always chosen from the middle of the largest gap between previous points. The sampled points are then identical for both methods when $b = 2, 3, 5, 9, 17, \ldots, 2^i + 1$ where $i$ is a non-negative integer. At other times, the resulting sampling is not uniform and thus scores are somewhat worse (without statistical significance in most cases).

A simple modification we could make to our algorithm is to choose a uniform discretization *with an information-based ordering of evaluations* whenever $\alpha = -\infty$ and $\beta = \infty$. A more interesting and potentially significant question would be whether the maximum-likelihood decision process of information-based optimization could efficiently make use of our knowledge of breadth $b$. Should our choice of the next point for evaluation seek also to increase the likelihood of pruning in future iterations?

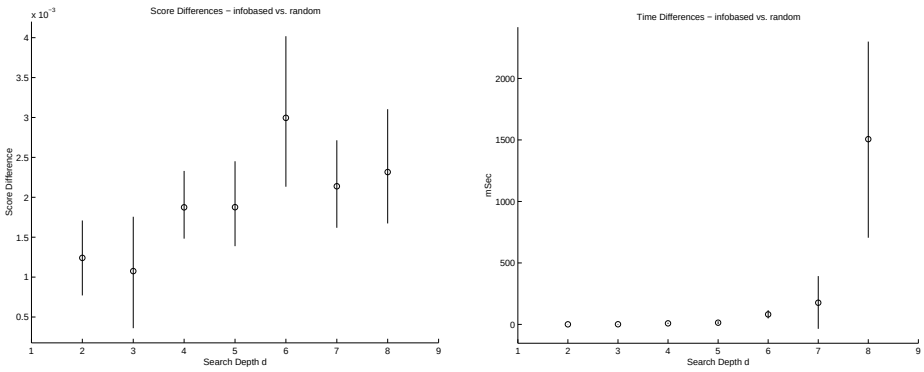



**Fig. 4.** Score and time difference confidence intervals for information-based versus random discretization with varying search depth

The next important parameter we vary is search depth, i.e. the length of the action sequences (and thus the time horizon) we consider in search. We hold search breadth fixed at 10.

Varying search depth and comparing Information-Based Alpha-Beta Search with DASAT alpha-beta search with random discretization, we again generally see a *time versus score tradeoff*. See Figure 4. Information-based discretization consistently outscores random discretization as we would expect. The difference in score increases with depth. We do observe an exponential decrease in the relative number of nodes searched as we increase depth using information-based discretization. However, the quality of the pruning does not compensate for the computational cost of optimization, and the overall run-time difference increases nearly exponentially for higher search depths.

In summary, we observe consistently better scores from information-based discretization, but random discretization appears to be significantly more efficient for higher search depths. One could imagine an interesting hybrid of the two, where search at greater depths is performed stochastically and more crucial decisions shallower in search are performed with information-based optimization.

Varying search depth and comparing Information-Based Alpha-Beta Search with DASAT alpha-beta search with uniform discretization, we again generally see the *same scores but with nearly exponential savings in time* as the depth increases. See Figure 5. For almost all depths, we see no significant statistical difference in score. However, as we increase search depth, we observe a clear exponential decrease in the relative number of nodes searched. This pruning more than compensates for the computational cost of the optimization, and again yields nearly exponential savings in time with the increase in search depth.

In summary, information-based discretization, when compared to random discretization yields substantially better decisions but at greater computational cost. Information-based discretization, when compared to uniform discretization,

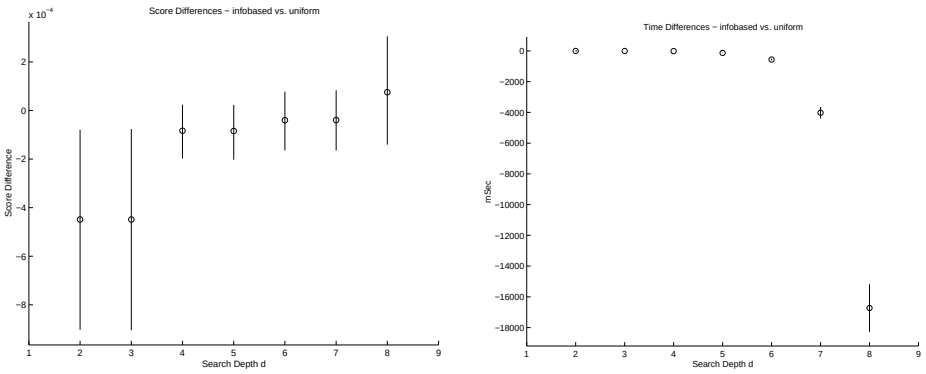



**Fig. 5.** Score and time difference confidence intervals for information-based versus uniform discretization with varying search depth

yields similar quality of decisions but with significantly lesser computational cost. This is consistent with the previous empirical study of [5].

We also experimented with varying $\Delta t$. Results showed that control with lookahead significantly outperforms greedy control. The policy leads to one of two symmetric equilibria in the upper quadrants where the pursuer's maximal turn affects no change in relative position.

## 9 Discussion

As this is but one empirical study, it is worth taking some time to help the reader understand what to expect for significantly different games. Even without empirical study, we can sketch the gross effects on computational complexity.

**Varying number of players:** As we increase the number of players while holding the number of turns per player fixed, we linearly increase search depth, which *exponentially* increases search time.

**Varying time horizon:** As we increase the time horizon while holding the frequency of actions constant, we also linearly increase search depth, which *exponentially* increases search time.

**Varying system dynamics:** Our homicidal chauffeur example has continuous dynamics, but we could just as well have hybrid dynamics. However, different dynamics results in different computational complexity for simulation. Simulation run-time has a *proportional* effect on search run-time.

**Varying system dimensionality:** The dimensionality of a hybrid state has no bearing on the complexity of search except through its effect on the computational complexity of simulation.

**Varying control input dimensionality:** If we increase the number of control inputs, then to keep the sampling density constant, the number of samples would need to increase exponentially. Thus the breadth $b$ and run-time of search would grow *exponentially*.

One common misunderstanding of advanced search techniques is the skill required to use them. There is an important distinction between search algorithms and search problems. Indeed, none of the search algorithm code of [5] needed modification. Rather, we simply redefined those elements corresponding to our DASAT hybrid system game definition. This has an important ramification: A change in the game definition results in change in the behavior recommended by search. Thus, to the extent that search is fast and efficient, search enables *model-based control* where control is based on an internal model of system behavior and adapts control policy as the model changes.

## 10 Conclusions

The classical formalism for game-tree search assumes that the game is a discrete system. In order to apply such techniques to continuous or hybrid systems, the user must first approximate system dynamics by performing both action and action timing discretizations a priori.

We have extended the game formalism to allow both (1) dynamic sampling from closed, contiguous sets of allowable actions (i.e. control vectors) during search, and (2) general hybrid system dynamics. We have also introduced a new algorithm called Information-Based Alpha-Beta Search that works with a subclass of these games. This empirical study and that of [5, ch. 4] show that for two such games, Information-Based Alpha-Beta Search recommends control actions similar to that of uniform discretization, but at considerable savings in computational time.

## Acknowledgements

We wish to thank Claire Tomlin for the problem suggestion, and Kim and Clif Presser for simplification of formulæ for our parabolic variant.

## References

1. Michael S. Branicky. *Studies in Hybrid Systems: modeling, analysis, and control.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1995.
2. Rufus Isaacs. *Differential Games, 2nd ed.* Kruger Publishing Company, Huntington, NY, USA, 1975. First edition: Wiley, NY, 1965.
3. A. W. Merz. *The Homicidal Chauffeur – a differential game.* PhD thesis, Stanford University, Palo Alto, California, 1971. Report No. 94305, Department of Aeronautics and Astronautics.
4. Todd W. Neller. Information-based optimization approaches to dynamical system safety verification. In Thomas A. Henzinger and Shankar Sastry, editors, *LNCS 1386: Hybrid Systems: computation and control, First International Workshop, HSCC'98, Proceedings*, pages 346–359. Springer, Berlin, 1998.
5. 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.