A Bayesian Effort Bias for Sampling-based Motion Planning

Scott Kiesel and **Wheeler Ruml**

Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA

Abstract

Recent advances in sampling-based motion planning have exploited concepts similar to those used in the heuristic graph search community, such as computing heuristic cost-to-go estimates and using state-space abstractions to derive them. Following this trend, we explore how the concept of search effort can be exploited to find plans quickly. Previous work in motion planning attempts to find plans quickly by preferring states with low cost-to-go. Recent work in graph search suggests that following search-effort-to-go estimates can yield faster planning. In this paper, we demonstrate how this idea can be adapted to the context of motion planning. Our planner, BEAST, uses on-line Bayesian estimates of effort to guide the expansion of a motion tree toward states through which a plan is estimated to be easy to find. We present results in five simulated domains (Kinematic and Dynamic Car, Hovercraft, Blimp and Quadrotor) indicating that BEAST is able to find solutions more quickly and has a higher success rate than previous methods. We see this work as further strengthening the algorithmic connections between motion planning and heuristic graph search.

Introduction

We address the problem of single-query kinodynamic motion planning: given a start state, description of the obstacles in the workspace, and a goal region, find a dynamically feasible continuous trajectory (a sequence of piece-wise constant controls) that takes the robot from the start state to the goal region without intersecting obstacles (Choset et al. 2005; LaValle 2006). We work within the framework of motion trees, popularized by sampling-based motion planning, in which the planner grows a tree of feasible motions from the start state, attempting to reach the goal state. This approach is appealing because it applies to any vehicle that can be forward simulated, allowing the planner to respect realistic constraints such as acceleration limits. Examples of algorithms taking this approach include RRT (LaValle and Kuffner 2001), KPIECE (Şucan and Kavraki 2009), and P-PRM (Le and Plaku 2014).

Although the figure of merit on which these algorithms are usually compared is the time taken to find a (complete and feasible) solution. Close examination of these algorithms reveals that their search strategies are not explicitly designed to optimize that measure. RRT uses sampling with a voronoi bias to encourage rapidly covering the entire state space. KPIECE uses more sophisticated coverage estimates to achieve the same end. Focusing on regions of the state space with low motion tree coverage helps to grow the tree outward, but is not focused on reaching the goal. Coverage promotes probabilistic completeness but not necessarily finding a solution quickly.

In artificial intelligence, a central principle for exploring large state spaces is to exploit heuristic information to focus problem-solving in promising regions. The A* heuristic graph search algorithm serves as the central paradigm. In motion planning, the P-PRM algorithm exploits heuristic cost-to-go information to guide growth of its motion tree, with the aim of finding solutions faster than unguided methods. While focusing on low cost regions directs sampling toward the goal, it ignores the effort that can be required for a motion planner to thread a trajectory through a cluttered area. In this way, cost-to-go estimates can encourage the search to focus on challenging portions of the state space, slowing the search. Fundamentally, optimizing solution cost is not the same as optimizing planning effort.

Recent work in heuristic graph search has recognized the separate roles of cost and effort estimates in guiding search, particularly when solutions are desired quickly (Thayer and Ruml 2009; Thayer and Ruml 2011). In this paper, we show how to exploit that idea in the context of motion planning. We propose an algorithm, Bayesian Effort-Aided Search Trees (BEAST), that biases tree growth through regions in the state space believed to be easy to traverse. If motion propagation does not go as anticipated, effort estimates are updated online based on the planner's experience, and used to redirect planning effort to more fruitful parts of the state space. We implement this method in the Open Motion Planning Library (OMPL) (Sucan, Moll, and Kavraki 2012) and evaluate it in five different simulated domains (Kinematic and Dynamic Car, Hovercraft, Blimp and Quadrotor). The results suggest that BEAST successfully uses effort estimates to efficiently allocate planning effort: it finds solutions faster than RRT, KPIECE, and P-PRM and is the only method able to solve all benchmark instances. We see this work as a further demonstration of how ideas from heuristic graph search can be useful in motion planning.

Previous Work

There has been much previous work on biases for sampling-based motion planners. The two most prominent types in the recent literature have been to bias toward less explored portions of the state space or to bias exploration toward regions of the state space believed to contain low cost solutions. Both of these biases have shown strong results in finding solutions quickly. The two state of the art algorithms considered in this paper are KPIECE (Şucan and Kavraki 2009) and P-PRM (Le and Plaku 2014).

KPIECE

Kinodynamic Planning by Interior-Exterior Cell Exploration, or KPIECE (Şucan and Kavraki 2009), is an algorithm that uses a multi-level projection of the state space to estimate coverage in the state space. It then uses these coverage estimates to reason about portions of the state space to explore next. Expansive Space Trees (EST) (Hsu, Latombe, and Motwani 1999) and Path-Directed Subdivision Tree (PDST) (Ladd and Kavraki 2005) also focus on less explored portions of the state space but have been shown to be outperformed by KPIECE. The general all-around good performance of KPIECE has led to its selection as the default motion planner in OMPL.

KPIECE is focused on quickly covering as much of the state space as possible. It always gives priority to less covered areas of the state space. When an area of low coverage is discovered it attempts to extend the motion tree into that area. It uses a coarse resolution initially to find out roughly which area is less explored. Within this area, finer resolutions can then be employed to more accurately detect less explored areas.

While KPIECE targets exploring unvisited areas of the state space, this may not always be the fastest approach to finding the goal. Certainly targeting exploration toward the goal could help improve performance.

P-PRM

P-PRM (Le and Plaku 2014) is based on ideas from an earlier planner called Synergistic Combination of Layers of Planning (SyCLOP) (Plaku, Kavraki, and Vardi 2010). It shares the intuition that information from a discrete abstraction of the workspace can be used to identify low level paths that may lead to the goal. While SyCLOP was shown to be very successful, in recent work P-PRM has been shown to outperform SyCLOP in a variety of planning problems.

P-PRM uses the geometric component of the state space to construct a Probablistic Roadmap (PRM) (Kavraki et al. 1996). It generates random states in the geometric space, then connects each state to its nearest neighbors via an edge, forming a graph. The edges in the graph are collision checked and removed from the graph if a collision along them is found. The graph vertices represent regions of geometric space and the edges summarize the connectivity of the regions.

P-PRM runs a Dijkstra search out from the abstract region containing the concrete goal to compute h-values, or heuristic estimates of cost to the goal. It then uses these heuristic values, and the associated shortest paths from the goal to each abstract node, to bias sampling.

It searches by maintaining a queue of abstract states in the graph sorted by increasing scores (initially their h-value, see the paper for details). At each search iteration the abstract state with the lowest score is selected. An abstract state along the cheapest precomputed path rooted at the currently selected state is chosen. This state is then used to create a random concrete state within some pre-specified state radius. This is now the "target" state used similarly to when plain RRT chooses a state uniformly at random. That means that the nearest state in the existing motion tree is chosen as the root for the new propagation which is steered (if possible) toward the random state. Any new abstract states touched by the propagation attempt are added to the queue if not previously enqueued.

P-PRM tries to pursue the completion of low cost paths by following its heuristic estimates in the abstract space. It tries to avoid getting stuck during planning by penalizing the score of abstract states when they are examined.

Speedy Search

While RRT and KPIECE are often the reliable workhorses of motion planning, the success of heuristically-informed graph search algorithms such as A* (Hart, Nilsson, and Raphael 1968) in artificial intelligence would suggest that brute-force expansion into all unexplored regions of the state space (in a manner similar to Dijkstra's algorithm) is not an optimal strategy. P-PRM has been shown to provide state of the art performance by exploiting heuristic cost-to-go guidance. Yet recent results in the heuristic graph search community show that exploring the state space based on cost often does not give the best speedup.

In the context of discrete graphs, Greedy search, which focuses on nodes with low heuristic cost-to-goal, is often surpassed by 'Speedy search', which focuses on nodes with a low estimated number of hops (or graphs edges) to the goal (Thayer and Ruml 2009; Wilt and Ruml 2014). In this paper, we present one attempt at adapting this idea to motion planning, in which the state and action spaces are continuous and there is no predefined graph structure.

Exploiting Effort Estimates

While there is not a direct translation of the "number of edges to the goal" concept, there is still a notion of search effort. In heuristic search, the fewer expansions needed to find the goal, typically the quicker a solution is found. In sampling-based motion planning, the unit of measure would be the number of samples, or propagation attempts in the motion tree. Each forward propagation of the system state requires collision checking, which is computationally expensive. The fewer propagation attempts made before finding the goal, typically the faster a solution is found (assuming reasonable iteration overhead).

Overview

Bayesian Effort-Aided Search Trees (BEAST) is a novel method that tries to find solutions as quickly as possible by

constructing solutions which it estimates require the least effort to build. It maintains online Bayesian estimates of the effort of connecting abstract regions of the state space and allocates its search effort to the region of the state space that is estimated to require the lowest effort to connect to the abstract goal region.

BEAST exploits a discrete abstraction of the state space. In the experiments reported below, we use a PRM workspace abstraction very similar to the one used by P-PRM. We begin by identifying the geometric component of the state space. The abstraction will exist only in this subspace. We generate uniformly random states in the abstract space (1000 in the experiments below). As in P-PRM, these states induce a division of the state space into abstract regions (by associating any concrete state with the nearest abstract state). Neighboring abstract states (the 5 nearest in the experiments below) are connected by directed edges, forming a directed graph. (If the abstract start and goal regions remain unconnected, additional samples are taken until they are.)

As just discussed, for each edge e, BEAST maintains an effort estimate, $ee(e)$, of how many propagation attempts would be required on average to take a concrete state contained in the abstract region represented by the source vertex of the edge to a concrete state contained in the abstract region represented by the end vertex. These estimates are initialized by a geometric collision check along the abstract edge. However, BEAST explicitly acknowledges that this quick check in geometric space is only a rough approximation of a robot's ability to steer from one region to the other. We represent our uncertain belief about each edge in a Bayesian style: we regard a propagation attempt as sampling a Bernoulli variable and we maintain a beta distribution (with parameters α, β) over its success probability. The initial geometric collision check provides some evidence about this probability, and then each propagation attempt during planning provides additional evidence. In the experiments reported below, an edge with a detected collision is initialized to $\alpha = 1, \beta = 10$, and all other edges are initialized to $\alpha = 10, \beta = 1$. Successful attempts increase α by one and unsuccessful attempts increase β by one. Based on our belief, we estimate the number of propagation attempts that will be necessary in order to have a successful one as $(\alpha + \beta)/\alpha$.

BEAST uses the abstract graph as a metareasoning tool to decide where it should spend its time growing the motion tree. We only consider abstract regions touched by the motion tree and each edge from the corresponding vertex in the abstract graph represents a possible propagation attempt. We compute, for each directed edge e, the expected total effort $te(e)$ required to reach the abstract goal if we start propagating a state from its start region through its end region and onward to the goal. For 'exterior' edges, whose start region has not yet had a successful propagation into its end region, this is straightforward: the effort to cross that edge plus the total effort-to-goal from the end vertex. More formally: if, for every vertex in the abstract graph, we let $te(v)$ be the minimum over its outgoing edges e of $te(e)$, then $te(e) = ee(e) + te(e.end)$. 'Interior' edges are more complex. Unless the goal region has been reached, any interior edge will lead to an exterior edge that has a lower total effort estimate, so such edges may not appear to be useful for propagation. However, recall that our state space abstraction might be very rough, and not all concrete states falling in the same abstract region are necessarily equivalent. We may well want to propagate along an interior edge in order to add additional states to the end region, in the hopes that this will increase the probability of being able to propagate onward from there. We model this by assuming that an additional state in the destination region will raise its α by $1/n$, where n is the number of states already in the region. (We want this bonus to depend inversely on the number of existing states, to reflect the decreasing marginal utility of each additional state.) So for an interior edge e with a destination vertex d that contains n states in its abstract region,

$$te(e) = ee(e) + \min_{e_2 \in e.out} \frac{e_2.\alpha + 1/n + e_2.\beta}{e_2.\alpha + 1/n} + te(e_2.dest).$$

BEAST($Abstraction, Start, Goal$)

```
1.  AbstractStart = Abstraction.Map(Start)
2.  AbstractGoal = Abstraction.Map(Goal)
3.  Abstraction.PropagateEffortEstimates()
4.  Open.Push(AbstractStart.GetOutgoingEdges())
5.  While NotFoundGoal
6.    Edge = Open.Pop()
7.    StartState = Edge.Start.Sample()
8.    EndState = Edge.End.Sample()
9.    ResultState = Steer(StartState, EndState)
      // Or Propagate With Random Control
10.   Success = Edge.End.Contains(ResultState)
11.   If Success
12.     Edge.UpdateWithSuccessfulPropagation()
13.     If Edge.End == AbstractGoal
14.       Open.Push(GoalEdge)
          // Goal Region To Goal State
15.   Else
16.     Edge.UpdateWithFailedPropagation()
17.   Abstraction.PropagateEffortEstimates()
18.   Open.Push(Edge)
19.   If Success
20.     Open.Push(Edge.End.GetOutgoingEdges())
```

Figure 1: Pseudocode for the BEAST algorithm.

Details

Pseudocode for BEAST is presented in Figure 1. The algorithm is passed an abstraction of the workspace, concrete start state and a concrete goal state. BEAST first begins by propagating effort estimates through the abstract graph outward from the region containing the concrete goal state (line 3). For efficiency, the collision checking and beta distribution initialization can be done lazily.

We use the pseudocode in Figure 2 to estimate the number of propagation attempts needed if the planner were to start by propagating along a specific edge. For exterior edges, this effort value is straightforward (line 22).

On Line 25 for interior edges, we examine each of the

GetEffort($Edge$)
21. **If Not** Edge.interior
22. **Return** ee(Edge) + te(Edge.End)
23. **Else**
24. Child_Edges = Edge.End.GetOutgoingEdges()
25. **Return** ee(Edge) + $\min_{Child \in Child\_Edges}$ OptimisticBenefit(Child) + te(Child.End)

OptimisticBenefit(Edge)
26. PositiveEffect = 1. / Edge.Start.NumStates
27. Optα = Edge.α + PositiveEffect
28. **Return** (Optα + Edge.β) / Optα

Figure 2: Pseudocode for calculating an edge effort value.

children of the current edge to see which child edge would require the least effort to arrive at the goal if it were provided another state in its start region. We take the minimum effort over the children and add in the estimated effort of propagating along the current edge.

If effort estimates were static, a single pass of Dijkstra's algorithm would suffice to compute te values. In our case, edge effort estimates change over time so we use an incremental best-first search called D* Lite (Koenig and Likhachev 2002) to avoid replanning from scratch. D* Lite updates the heuristic estimates for cost to go to the goal at each vertex in the graph, in our case we are using effort (te) to go instead. While propagating effort at each vertex we also store an effort estimate at each edge which is calculated using $GetEffort$.

To reiterate, this value can be seen as an estimate of how many samples will be required to reach the goal if you were to choose to propagate along an edge and then choose the minimum effort edges thereafter. A queue called $Open$ is then initialized with outgoing edges from the abstract region containing the concrete start state(line 4). $Open$ is sorted in increasing order of edge effort. The search always considers the least effort edge first.

The algorithm proceeds by popping the edge off $Open$ with the lowest estimated effort (line 6). This edge is then sampled at its start abstract region and its end abstract region in lines 7-8. In our implementation, the concrete state in the edge's start region that has been selected the fewest number of times is chosen as the $StartState$. A concrete state is chosen from the edge's abstract end region uniformly at random within some radius centered around the region's centroid.

An attempt is made to grow the tree from $StartState$ to $EndState$ using a steering function (line 9). In our implementation if no steering function was available in OMPL, we instead generated 10 random controls, applied each to $StartState$ and the resulting motion that got closest to $EndState$ was chosen. [1]

If the newly propagated motion at any point reached the target abstract region (the selected edge's end region), the edge is updated with a successful trial (line 10-12). This simply adds one to the α value of the beta distribution associated with this edge.

If the target region is not reached, the β bucket is incremented (line 16). With each trial to propagate along an edge we update our belief about the effort required to reach the goal by using the edge. This effectively changes the edge "cost" in the abstract graph and we use our incremental search to update the effort estimates throughout the graph based on this local update (line 17).

If the edge was successfully propagated along, we also add its child edges (outgoing edges from the current edge's end region) to the $Open$ list (line 20). We re-add the current edge to the $Open$ in all outcomes (line 18).

There is also a special case (line 13) added which enables us to use sparse abstractions. With sparse abstractions we can compute our effort values more efficiently during each iteration. However, when the goal abstract region is reached, with a sparse abstraction, it may cover a large portion of the state space. Growing the tree into a possibly large goal region may not be focused enough to find a state close to the goal state. To combat this we add a special $GoalEdge$ to $Open$ (line 14). This is an edge that when expanded will return a $StartState$ from the goal abstract region and an $EndState$ focused around the actual concrete goal state.

Experiments

All experiments were run using control algorithms from the OMPL framework where available (KPIECE and RRT). P-PRM was implemented following closely along with the description and pseudo code included in the paper. Experiments also used OMPL's implementation of a Kinematic Car, Dynamic Car, Blimp and Quadrotor vehicle, as detailed below. We implemented a Hovercraft in OMPL following Lynch (1999).

Kinematic Car

The mesh used for the Kinematic Car vehicle is shown in Figure 3 panel (a). The equations defining the Kinematic Car's motion and control inputs in OMPL are as follows:

$$\dot{x} = u_0 \cdot cos(\theta),$$
$$\dot{y} = u_0 \cdot sin(\theta),$$
$$\dot{\theta} = \frac{u_0}{L} \cdot tan(u_1)$$

where the control inputs (u_0, u_1) are the translational velocity and the steering angle, respectively, and L is the distance between the front and rear axle of the car which is set to 1.

Dynamic Car

The mesh used for the Dynamic Car vehicle is shown in Figure 3 panel (a). The equations defining the Dynamic Car's

[1] This functionality was implemented at the control sampler level in OMPL for each domain so any algorithm using "sampleTo" provided by the domain's control sampler received equal benefit.

motion and control inputs in OMPL are as follows:

$$\begin{aligned}\dot{x} &= v \cdot cos(\theta),\\ \dot{y} &= v \cdot sin(\theta),\\ \dot{\theta} &= \frac{v \cdot m}{L} \cdot tan(\phi),\\ \dot{v} &= u_0,\\ \dot{\phi} &= u_1\end{aligned}$$

where v is the speed, ϕ the steering angle, the controls (u_0, u_1) control their rate of change, m is the mass of the car (set to 1), and L is the distance between the front and rear axle of the car (also set to 1)

Hovercraft

The mesh used for the Hovercraft vehicle is shown in Figure 3 panel (a). The equations defining the Hovercrafts's motion and control inputs from Lynch (1999) are as follows:

$$\begin{aligned}\dot{x} &= \frac{F}{M} cos(\theta) - \frac{B_t}{M} x,\\ \dot{y} &= \frac{F}{M} sin(\theta) - \frac{B_t}{M} y,\\ \dot{\theta} &= \frac{\tau}{0.5 \cdot M \cdot R^2} - \frac{B_r}{M} \cdot \theta\end{aligned}$$

where F is the force exerted by the thrusters and τ is the torque exerted by the thrusters. B_t and B_r are the translational and rotational friction coefficients (both set to 0). M is the mass of the robot and R is the radius of the robot (both set to 1).

Blimp

The mesh used for the Blimp vehicle is shown in Figure 3 panel (b). The equations defining the Blimp's motion and control inputs in OMPL are as follows:

$$\begin{aligned}\ddot{x} &= u_f \cdot cos(\theta),\\ \ddot{y} &= u_f \; sin(\theta),\\ \ddot{z} &= u_z,\\ \ddot{\theta} &= u_\theta\end{aligned}$$

where (x, y, z) is the position, θ the heading, and the controls (u_f, u_z, u_θ) control their rate of change.

Quadrotor

The mesh used for the Quadrotor vehicle is shown in Figure 3 panel (c). The equations defining the Quadrotor's motion and control inputs in OMPL are as follows:

$$\begin{aligned}m\ddot{p} &= -u_0 \cdot n - \beta \cdot \dot{p} - m \cdot g,\\ \alpha &= (u_1, u_2, u_3)^T,\end{aligned}$$

where p is the position, n is the Z-axis of the body frame in world coordinates, α is the angular acceleration, m is the mass, and β is a damping coefficient. The system is controlled through $u = (u_0, u_1, u_2, u_3)$.

In the Kinematic and Dynamic Car domains the goal radius was set to 0.1, the remaining domains each used a goal radius of 1. The goal distance of a state was based only on the distance in the XY or XYZ dimensions. Other parameters that were used included a propagation step value of 0.05, min and max control durations of 1 and 100 respectively, and intermediate states were included during planning. The workspace was bounded by $-30 \leq x \leq 30$, $-30 \leq y \leq 30$ and $-5 \leq z \leq 5$.

KPIECE and RRT were run using their default parameters. P-PRM was also run using its suggested parameters described in the paper. The state radius size for sampling was shared between P-PRM and BEAST. This value was set to 6, which gave good visible coverage over the abstract regions and the best performance over those state radii tried: {2,4,6}.

The obstacle mesh used for the experiments is presented in Figure 3 panel (d). For each vehicle, 5 start and goal pairs were used, and for each start and goal pair 50 different random number generator seed values were used. This provided 250 runs for each of the domains. The start states were biased toward the center of the workspace while the goal was biased toward the lower center of the workspace. This setup tends to generate problems in which the optimal solution threads its way carefully between the obstacles, but it is much easier to take a more costly route around the obstacles. This wide diversity of planning time/solution cost trade-offs directly tests the ability of BEAST to estimate planning effort and adjust its behavior accordingly. A motion planning that explicitly tries to find plans quickly ought to exhibit superior performance. A planning timeout of 60 seconds was used.

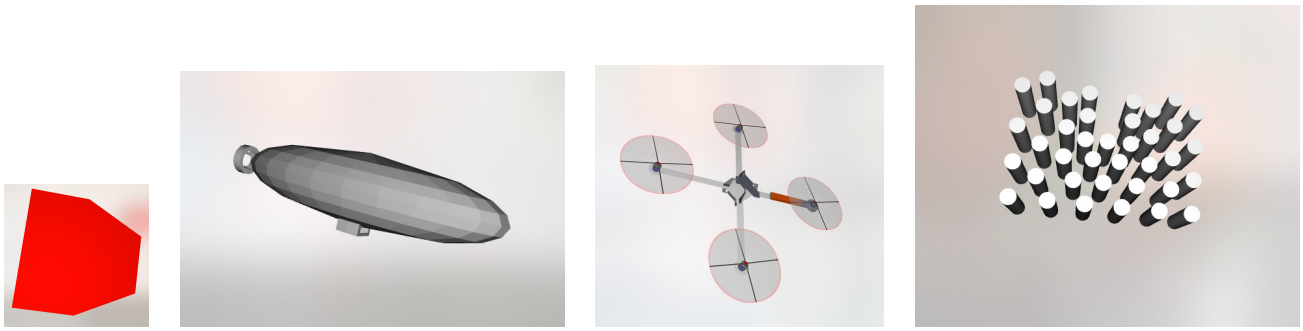
Results

The results of the experiments are presented in Figure 4. Each box represents the middle 50% of the data, with a horizontal line at the median. Whiskers extend to the furthest point within 1.5 times the interquartile range. The remaining outliers are plotted with circles. The 95% confidence interval around the mean is depicted with a gray rectangle. The plots in each panel are sorted according to their means. In order to have enough data points to create plots, algorithm runs that timed out without providing a valid solution are still included in the plot. These runs are represented by the time collected by OMPL after the timeout was issued. Several of the plots have been clipped at the top so that the top two performers remain legible.

In the Blimp domain, BEAST has the lowest mean planning time as well as the lowest variance in its performance. In the Quadrotor domain, BEAST again has the lowest mean, but P-PRM appears to have slightly lower variance.

A video of the sampling and tree growth of each of the algorithms considered in this paper can be found at `https://www.youtube.com/watch?v=Or8sQBOrVh4`. It is a top down visualization of a Quadrotor planning instance. It illustrates RRT's slow coverage of the entire state space, KPIECE's rapid coverage of the state space, P-PRM's focus on estimated low cost paths and BEAST's focus on finding low effort solutions.

The number of runs where each algorithm was unable to solve an instance is provided in Figure 5. In the Blimp do-



(a) (b) (c) (d)

Figure 3: The car, blimp and quadrotor vehicles used in the experiments, and the forest environment.

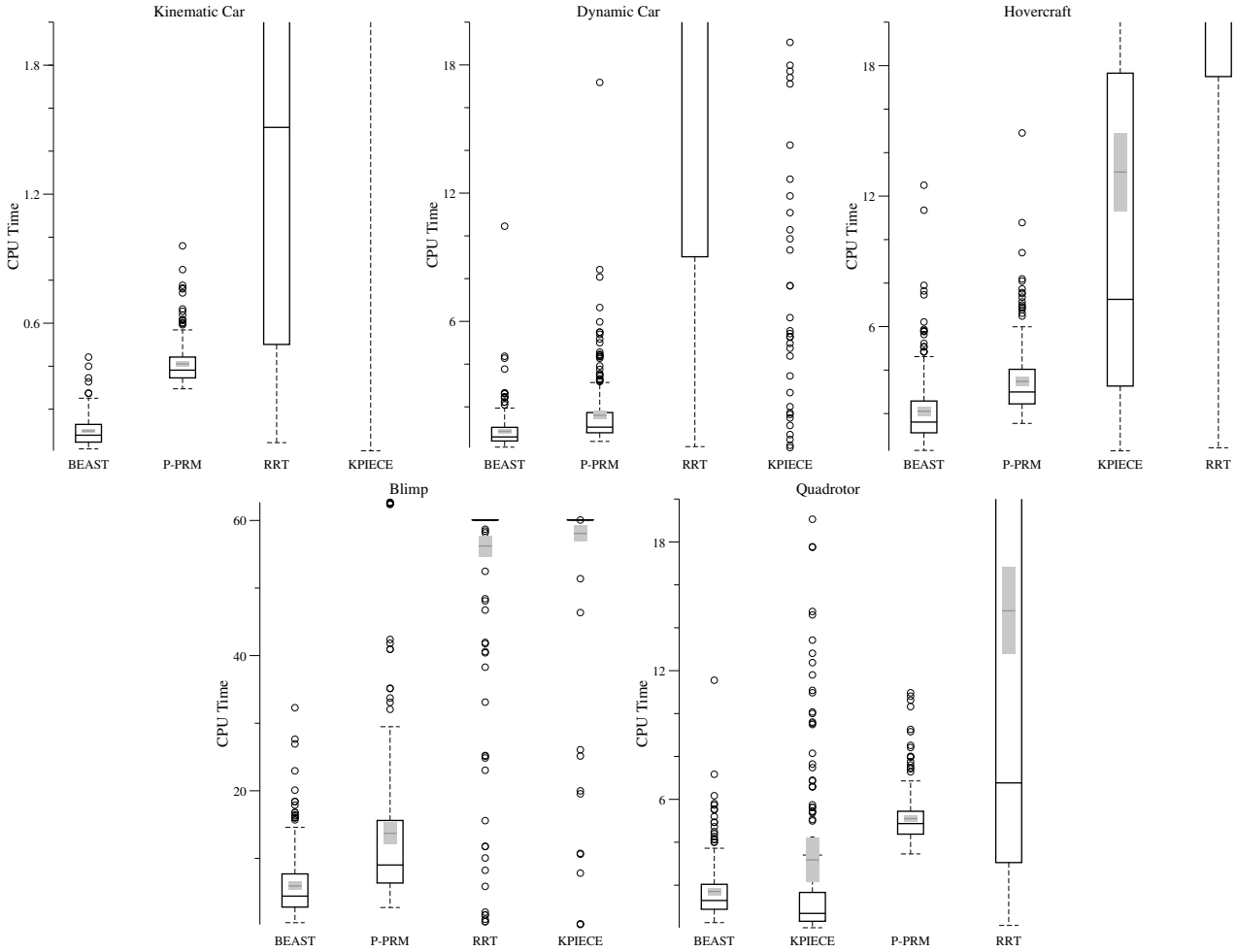


Figure 4: Computation time for 5 start goal pairs and 50 random seeds (250 instances).

	RRT	KPIECE	P-PRM	BEAST
Kinematic Car	0	99	0	0
Dynamic Car	108	189	0	0
Hovercraft	116	8	0	0
Blimp	221	238	11	0
Quadrotor	12	2	0	0

Figure 5: Number of unsolved instances for 5 start goal pairs and 50 seeds (250 instances).

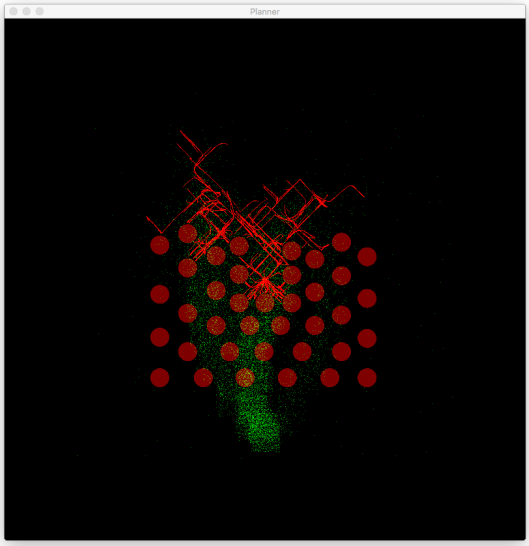


Figure 6: P-PRM sampling and tree growth example in the Quadrotor domain (top down view).

main, BEAST is the only algorithm that is able to find a solution to all the instances within the timeout. In the Quadrotor domain, BEAST and P-PRM are both able to find solutions to all instances while KPIECE and RRT are not able to within the timeout.

Discussion

One of the major benefits of BEAST is that it explicitly focuses on areas of the state space that it believes will be easy to traverse while heading toward the goal. KPIECE will eventually explore the same regions of the state space but does so without focusing on paths toward the goal. P-PRM does focus on paths leading to the goal, but focuses on paths associated with low cost. These paths can be arbitrarily difficult to find given obstacle configurations.

This is shown in Figure 6 where many P-PRM generates samples (green dots) along abstract paths to the goal, but it is challenging to grow the motion tree (red lines) toward them. Eventually from the uniform random sampling and increasing cost estimates for the states it has selected many times, search begins to spill around and through the obstacles (red circles).

Another feature of BEAST that helps it construct its tree more efficiently is that it focuses its tree growth either internal to the existing tree or directly along the fringe of the existing tree. This focus on the boundary of the motion tree is very similar to that of KPIECE, yet the two methods allocate their exploration effort very differently. P-PRM does not focus its sampling near the existing tree and can generate samples arbitrarily far away, which are less helpful when growing the tree through tight spaces.

There are other motion planners that leverage heuristic cost-to-go, but in ways very different from BEAST. Informed RRT* (Gammell, Srinivasa, and Barfoot 2014) uses ellipsoidal pruning regions to ignore areas of the state space that are guaranteed not to include a better solution. BIT* (Gammell, Srinivasa, and Barfoot 2015) uses heuristic cost estimates directly in its search strategy, but for kinodynamic planning it requires a boundary value problem solver to rewire trajectories between sampled states, making it inapplicable to many problems.

Finding solutions quickly is an important feature in many applications, but convergence to an optimal solution is also highly desirable. In future work, we plan to combine our effort based planner BEAST with heuristic cost estimates, yielding an anytime planner which quickly finds a solution and then spends its remaining planning time improving its incumbent solution cost.

Conclusion

We have presented a new algorithm called Bayesian Effort-Aided Search Trees. BEAST exploits and updates Bayesian estimates of propagation effort through the state space to find solutions quickly. Results on a variety of domains showed that BEAST on average found solutions the fastest and was the only algorithm to find solutions to every instance in the benchmark set. We see this work as reinforcing the current trend toward exploiting ideas from AI graph search in the context of robot motion planning, and providing further evidence that searching under time pressure is a distinct activity from searching for low-cost solutions.

Acknowledgments

We gratefully acknowledge support from NSF (grant 1150068).

References

[Choset et al. 2005] Choset, H.; Lynch, K.; Hutchinson, S.; Kantor, G.; Burgard, W.; Kavraki, L.; and Thrun, S. 2005. *Principles of robot motion: theory, algorithms, and implementation*. MIT Press.

[Gammell, Srinivasa, and Barfoot 2014] Gammell, J. D.; Srinivasa, S. S.; and Barfoot, T. D. 2014. Informed RRT*: Optimal incremental path planning focused through an admissible ellipsoidal heuristic. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[Gammell, Srinivasa, and Barfoot 2015] Gammell, J. D.; Srinivasa, S. S.; and Barfoot, T. D. 2015. Batch informed trees (BIT*): Sampling-based optimal planning via the

heuristically guided search of implicit random geometric graphs. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, 3067–3074. IEEE.

[Hart, Nilsson, and Raphael 1968] Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on* 4(2):100–107.

[Hsu, Latombe, and Motwani 1999] Hsu, D.; Latombe, J.-C.; and Motwani, R. 1999. Path planning in expansive configuration spaces. *International Journal of Computational Geometry & Applications* 9(04n05):495–512.

[Kavraki et al. 1996] Kavraki, L. E.; Švestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on* 12(4):566–580.

[Koenig and Likhachev 2002] Koenig, S., and Likhachev, M. 2002. D* lite. In *AAAI/IAAI*, 476–483.

[Ladd and Kavraki 2005] Ladd, A. M., and Kavraki, L. E. 2005. Motion planning in the presence of drift, underactuation and discrete system changes. In *Robotics: Science and Systems*, 233–240.

[LaValle and Kuffner 2001] LaValle, S. M., and Kuffner, J. J. 2001. Randomized kinodynamic planning. *The International Journal of Robotics Research* 20(5):378–400.

[LaValle 2006] LaValle, S. M. 2006. *Planning Algorithms*. Cambridge University Press.

[Le and Plaku 2014] Le, D., and Plaku, E. 2014. Guiding sampling-based tree search for motion planning with dynamics via probabilistic roadmap abstractions. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, 212–217. IEEE.

[Lynch 1999] Lynch, K. M. 1999. Controllability of a planar body with unilateral thrusters. *Automatic Control, IEEE Transactions on* 44(6):1206–1211.

[Plaku, Kavraki, and Vardi 2010] Plaku, E.; Kavraki, L. E.; and Vardi, M. Y. 2010. Motion planning with dynamics by a synergistic combination of layers of planning. *Robotics, IEEE Transactions on* 26(3):469–482.

[Şucan and Kavraki 2009] Şucan, I. A., and Kavraki, L. E. 2009. Kinodynamic motion planning by interior-exterior cell exploration. In *Algorithmic Foundation of Robotics VIII*. Springer. 449–464.

[Sucan, Moll, and Kavraki 2012] Sucan, I. A.; Moll, M.; and Kavraki, L. E. 2012. The open motion planning library. *Robotics & Automation Magazine, IEEE* 19(4):72–82.

[Thayer and Ruml 2009] Thayer, J. T., and Ruml, W. 2009. Using distance estimates in heuristic search. In *ICAPS*, 382–385.

[Thayer and Ruml 2011] Thayer, J. T., and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *IJCAI*, 674–679.

[Wilt and Ruml 2014] Wilt, C. M., and Ruml, W. 2014. Speedy versus greedy search. In *Seventh Annual Symposium on Combinatorial Search*.