An Effort Bias for Sampling-based Motion Planning

Scott Kiesel and Tianyi Gu and Wheeler Ruml

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. Most 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 kinodynamic motion planning. Our planner,** BEAST**, uses estimates of effort that are learned on-line to guide the expansion of a motion tree toward states through which a plan is estimated to be easy to find. We present results with four different simulated vehicles (car, hovercraft, blimp and quadrotor) in six different environments indicating that** BEAST **is able to find solutions much 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.**

I. 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 [1], [2]. 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 [3], KPIECE [4], and P-PRM [5].

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. For example, 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. Finding solutions quickly allows a robot to appear responsive in applications involving human interaction, and is also a crucial first ingredient for designing an anytime motion planner.

The authors are with the Department of Computer Science, University of New Hampshire, Durham, NH 03824, USA, skiesel, tg1034, ruml at cs.unh.edu. We gratefully acknowledge support from NSF (grant 1150068).

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 [6] 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 [7], [8]. 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) [9] and evaluate it with four different simulated vehicles (car, hovercraft, blimp and quadrotor) in six varied environments. The results suggest that BEAST successfully uses effort estimates to efficiently allocate planning effort: it finds solutions much faster than RRT, KPIECE, and P-PRM. Although the method ignores solution cost, we find that its trajectories still result in competitive goal achievement times. We see this work as a further demonstration of how ideas from heuristic graph search can be useful in sampling-based motion planning.

II. 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. We focus on one leading algorithm of each type.

A. KPIECE

Kinodynamic Planning by Interior-Exterior Cell Exploration, or KPIECE [4], 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) [10] and Path-Directed Subdivision Tree (PDST) [11] 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 RRT and KPIECE are often the reliable workhorses of motion planning, the success of heuristically-informed graph search algorithms such as A* in artificial intelligence would suggest that brute-force expansion into all unexplored regions of the state space (in a manner analogous to Dijkstra's algorithm) is not an optimal strategy. Exploring unvisited areas of the state space may not always be the fastest approach to finding the goal. For example, targeting exploration toward the goal could help improve performance.

B. P-PRM

P-PRM [5] is based on ideas from an earlier planner called Synergistic Combination of Layers of Planning (SyCLoP) [13]. They share the intuition that information from a discrete abstraction of the workspace can be used to bias exploration toward the goal. SyCLoP demonstrated significant speed-ups over unguided planners, and now the more recent P-PRM method has been shown to outperform SyCLoP.

P-PRM constructs its abstraction by using the geometric subspace of the state space to build a Probabilistic Roadmap (PRM) [14]. To distinguish between the concrete state space and the abstraction, we will reserve the terms state and motion for the concrete space and its motion tree, and vertex and edge for the abstract space and its PRM graph. P-PRM samples random vertices in the subspace, then connects each vertex to its nearest neighbors via an edge, forming a graph. The edges in the graph are collision checked (using geometric information only, no dynamics) and removed if a collision is found. The vertices implicitly induce a division of the state space into abstract regions, by associating any concrete state with the nearest abstract vertex. The edges summarize the connectivity of the regions. P-PRM runs a Dijkstra search [15] out from the vertex representing the abstract region containing the concrete goal to compute heuristic estimates of cost to the goal ($h(v)$ values). It then uses these heuristic values, and the associated shortest paths from the goal to each abstract node, to bias sampling.

P-PRM organizes its sampling by maintaining a queue of abstract vertices in the graph sorted by increasing scores (initially their h values, see [5] for details). At each search iteration the abstract vertex v_s with the lowest score is selected. An abstract vertex v_t along the cheapest path to the goal vertex rooted at v_s is chosen. This vertex is then used to create a random concrete state within some pre-specified sampling radius from v_t. This is now the 'target' state used for growing the motion tree, similarly to when plain RRT chooses a state uniformly at random. That means the nearest concrete state in the region represented by abstract vertex v_s is chosen as the root for the new propagation which is steered, if possible, toward the random state.[1] Any new abstract regions touched by the propagation attempt have their vertices 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 vertices when they are examined.

C. Speedy Search

Among single-query motion planners that don't require analytical steering functions or boundary-value-problem solvers, 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 [7], [16]. In this paper, we present an adaptation of this idea to motion planning, in which the state and action spaces are continuous and there is no predefined graph structure.

III. EXPLOITING EFFORT ESTIMATES

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

A. 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

[1] In our experiments, we choose the nearest state in the existing motion tree, regardless of the abstract region, as the root for the new propagation. This implementation performed better than the original in our preliminary experiments.

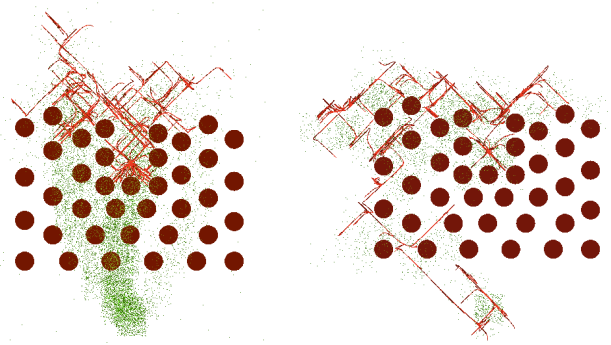


Fig. 1. Sampling (green dots) and tree growth (red arcs) for P-PRM (left) and BEAST (right) for a quadrotor vehicle in a 3D forest map.

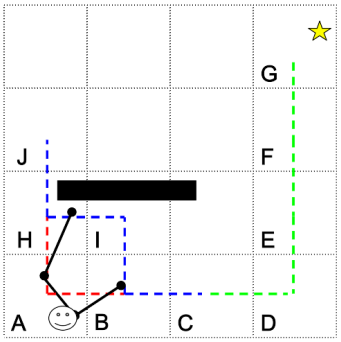


Fig. 2. An example of different types of edges reasoned about by BEAST.

effort to build. Building a motion tree will vary in difficulty across different regions of the workspace. Recognizing this, BEAST reasons about effort using a discrete abstraction of the state space, in the form of a directed graph of regions. As it grows the motion tree, it maintains for each edge in the graph an estimate of the effort that would be required to propagate concrete states from the region represented by the source vertex into the region represented by the destination vertex. At each iteration, it allocates its effort to grow the tree through the region of the state space that is estimated to require the least total effort (propagation attempts) before the motion tree reaches the abstract goal region. Figure 1 illustrates the behavior of BEAST in a forest map. In the left panel, P-PRM generates samples (green dots) along low-cost abstract paths to the goal, but it is challenging to grow the motion tree (red lines) toward them. In the right panel, after the same number of propagation attempts, BEAST has learned that it is difficult to propagate the motion tree downward and has focused on areas of the state space that it believes will be easy to traverse and hence it reaches the goal much faster.

In the experiments reported below, BEAST used a PRM workspace abstraction very similar to the one used by P-PRM. We begin by identifying the geometric component of the state space (such as position and orientation). We generate states uniformly at random in the abstract space (1000 in the experiments below). Neighboring abstract vertices (the five nearest in the experiments below) are connected by edges and collision checked, forming a directed graph. We check if the abstract start and goal vertices are in the same graph component, and generate additional samples until they are.

For each directed 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 itself. 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. Rather than discarding edges, we represent our uncertain belief about each edge e in a Bayesian style: we regard a propagation attempt as sampling a Bernoulli variable and we maintain a beta distribution with parameters $\alpha(e), \beta(e)$ over its success probability. The initial geometric collision check provides some evidence about this probability, and then each time we attempt to propagate from one region to the other during planning, we acquire additional evidence. In the experiments reported below, an edge e with a detected collision is initialized to $\alpha(e) = 1, \beta(e) = 10$, 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 $ee(e) = (\alpha(e) + \beta(e))/\alpha(e)$.

BEAST uses the abstract graph as a metareasoning tool to decide where it should spend its time growing the motion tree. We consider propagating from abstract regions already touched by the motion tree. Each edge from the corresponding vertex in the abstract graph represents a possible propagation attempt to a neighboring region. (This differs from P-PRM, for example, in which target samples can be far from the existing tree.) 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. (This also differs from P-PRM, in which regions' costs are updated individually without recomputing total path costs.) For 'exterior' edges, whose start region has not yet had a successful propagation into its destination region, computing $te(e)$ is straightforward: the effort to cross that edge plus the total effort-to-goal from the destination vertex. An illustration is given in Figure 2, in which a robot is planning a path from the smiley face in region A to the star in region G (a grid-based abstraction is used in this cartoon for simplicity). The solid black rectangular shape is an obstacle. Exterior edges are drawn in blue ($\vec{BC}, \vec{BI}, \vec{HI}, \vec{HJ}$) and the current motion tree is shown in black. For example, the $te(\vec{BC})$ is the estimated effort of $\vec{BC}$ plus the estimated efforts of the green edges leading to the goal ($\vec{CD}, \vec{DE}, \vec{EF}, \vec{FG}$). 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 $te(e) = ee(e) + te(e.dest)$.

'Interior' edges are more complex and are drawn in red in the figure ($\vec{AB}, \vec{AH}$). 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. If treated naively, the algorithm would never create additional states in previously-visited regions. 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 destination region, in the hopes that this will increase the probability of being able to propagate onward from there. An example of this in Figure 2 is $\vec{AH}$. Propagating along that edge would likely benefit $\vec{HJ}$, as the motions currently in the tree lead into an obstacle and may be hard to propagate further, yet there is additional unexplored space at the left side of region H from which further propagation could reach region J and hence the goal. We model this in a principled way by assuming that adding 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, where $d.out$ represents the outgoing edges from d,

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

This explicitly takes into account the possible benefits accrued by interior sampling: increasing the success of future propagations.

B. Details

Pseudocode for BEAST is presented in Figure 3. The algorithm is passed an abstraction of the workspace, a concrete start state and a concrete goal state (or region). 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.

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 inspired by D* Lite [17] to avoid replanning from scratch. D* Lite updates cost values from a vertex to vertices that appear relevant to an optimal path; in our case we are using effort-to-go (te) values and considering all vertices in the graph. (D*lite also handles a moving agent, which is not relevant in our situation.) While propagating effort at each vertex, we also store an effort estimate at each edge (the ee values).

BEAST uses the abstract graph to prioritize regions of state space for motion propagation. Specifically, it considers which edge in the abstract graph represent the best way to attempt to grow the motion tree. The edges represent work that can be done, and BEAST maintains a binary min-heap

```
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.Top()
 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.      Open.Push(Edge.End.GetOutgoingEdges())
14.      If Edge.End == AbstractGoal
15.        Open.Push(GoalEdge)
           // Goal Region To Goal State
16.    Else
17.      Edge.UpdateWithFailedPropagation()
18.    Abstraction.PropagateEffortEstimates()
```

Fig. 3. High-level pseudocode for the BEAST algorithm.

called $Open$, representing the edges whose source regions have been touched by the motion tree. In this way, BEAST differs from traditional discrete graph search algorithms in that it does not consider vertices to explore, but rather edges. $Open$ is initialized with outgoing edges from the abstract region containing the concrete start state (line 4), and is kept sorted in increasing order of total estimated effort (te). In each iteration, BEAST checks $Open$ for the edge with the lowest estimated effort (line 6). We then choose an existing concrete state in the edge's source region to propagate from (line 7). 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 new concrete target end state is generated from the edge's abstract end region uniformly at random within some radius centered around the region's centroid (line 8). Finally, 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.[2]

If the newly propagated motion at any point entered 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 was not reached, the β value is incremented (line 17). Each time we attempt to propagate between adjacent regions, we update our belief about the effort required to reach the goal by using

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

the corresponding abstract edge. This effectively changes the edge cost in the abstract graph and we then incrementally update the effort estimates throughout the graph based on this local update (line 18). If the edge was successfully propagated along, we also add its child edges (outgoing edges from its end region) to the $Open$ list, or update them if they are already present (line 13).

The goal region is handled specially (line 14). When it is reached, a special $GoalEdge$ is added to $Open$ (line 15) that, when expanded, will return a $StartState$ from the goal region and an $EndState$ focused around the actual concrete goal state. This promotes reaching the goal, even when the abstract goal region itself is large. One can view this edge as representing the choice to take a goal-biased sample in a traditional RRT.

C. Behavior of BEAST

BEAST is essentially a biasing technique for sampling-based motion planning. By reserving a fixed percentage of samples to be taken uniformly at random, BEAST can inherit the probabilistic completeness guarantee of the motion planning framework it is embedded in. BEAST's utility hinges on its abstraction capturing useful generalizations about the state space. For example, if the abstraction correctly approximates the geometric subspace of obstacle-free configuration space and dynamics are not important, one can expect BEAST to grow the motion tree directly to the goal. However, if the abstraction misses important aspects of the problem, for example if certain dynamics are crucial to a solution, then BEAST may not provide much speed-up. If the necessary dynamics are anticorrelated with geometric distance to the goal, and the abstraction is only based on geometry, then abstraction-guided methods like P-PRM and BEAST may even yield a slowdown. However, the on-line learning performed by BEAST does give it a certain robustness: if a given abstract edge is difficult to propagate along, it will eventually try a different route. On very simple problems, the overhead of forming and maintaining the abstraction may not be worth the possible decrease in state propagations and collision checking. And, of course, BEAST ignores solution cost, focusing entirely on trying to construct a trajectory as quickly as possible. In open spaces, minimizing the number of propagations will have the side-effect of minimizing trajectory length, but in cluttered spaces these metrics may be antagonistic.

IV. EXPERIMENTS

Given that BEAST is fundamentally heuristic, it is crucial to test its performance experimentally on a variety of vehicles and environments. Note that we are focusing on the kinodynamic motion planning problem, hence we focus on comparing to OMPL's control-based planners. We used implementations of KPIECE and RRT from OMPL, and we implemented P-PRM closely following the description and pseudo-code in the paper. We used OMPL's implementation of car, blimp and quadrotor vehicles, as detailed below, and we implemented a hovercraft in OMPL following [18]. We also used the 3 ladder, single wall, 2D forest, 3D forest, and fifthelement environments from OMPL (Figure 4). No path smoothing or other post-processing was done.

A. Setup

A simple mesh (Figure 4 panel a) was used for both the car and the hovercraft. The equations defining the car's motion and control inputs in OMPL are:

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

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)

The equations defining the hovercraft's motion and control inputs from [18] are:

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

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).

The mesh used for the blimp vehicle is shown in Figure 4 panel (f). The equations defining the blimp's motion and control inputs in OMPL are:

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

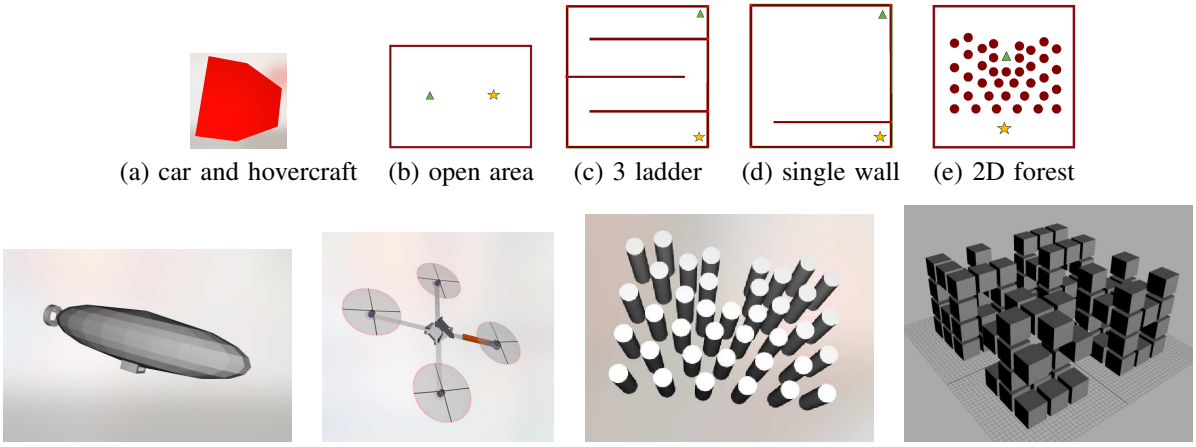
where (x, y, z) is the position, θ the heading, and the controls (u_f, u_z, u_θ) control their rate of change.

The mesh used for the Quadrotor vehicle is shown in Figure 4 panel (g). The equations defining the quadrotor's motion and control inputs in OMPL are:

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

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 $u = (u_0, u_1, u_2, u_3)$.

For the car, the goal radius was set to 0.1; the others 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 $-30 \leq x \leq 30$,



(a) car and hovercraft (b) open area (c) 3 ladder (d) single wall (e) 2D forest

(f) blimp (g) quadrotor (h) 3D forest (i) fifthelement

Fig. 4. The benchmark vehicles and environments. Top row: ground vehicles and 2D worlds. Bottom row: flying vehicles and 3D worlds.

$-30 \leq y \leq 30$ 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 in initial experiments over those state radii tried (2,4,6).

The environment meshes used for the ground vehicle experiments are presented in Figure 4 panels b–e and those used for flying vehicles in panels h and i. The open area (panel b) yields simple problems and was selected to test whether the ability to grow the motion tree directly toward the goal outweighs the overhead of maintaining an abstraction. The 2D 3 ladder (panel c) and 3D fifthelement environments (panel i) were selected to test the planner's ability to find complex paths, while the single wall (panel d) was selected to test the planner's ability to cope with narrow passages. The 2D and 3D forests (panels e and h) contain short hard-to-find solutions and longer easier-to-find solutions, aiming to test the ability of the planners to deal with clutter. The wide diversity of planning time/solution cost trade-offs directly tests the ability of BEAST to estimate planning effort and adjust its behavior accordingly.

All experiments were performed using a 3.16 GHz Intel E8500 CPU with 8GB RAM. For each environment, five start/goal pairs were used, and for each pair 50 different random seed values were used, yielding 250 runs for each vehicle/world combination. Start and goal states were chosen to yield non-trivial problems. In Figure 4, a triangle marks the approximate location of the start states and a star does the same for the goal. For example, in the forest domain, the start states were biased toward the upper center of the workspace while the goals were biased toward the lower center, generating 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. As our focus is on solving problems quickly, we stop each experiment after the first solution is found.

map	vehicle	P-PRM	KPIECE	RRT
open area	car	1.1–1.9	18–35	3.2–5.8
	hover.	0.7–1.1	3.2–6.9	5.9–10
single wall	car	4.9–6.3	6.2–9.1	6.2–8.1
	hover.	9.2–11	2.1–3.7	11–13
3 ladder	car	2.9–3.4	2.4–4.6	5.0–6.3
	hover.	8.7–10	1.3–1.7	9.1–10
2D forest	car	1.1–1.6	100–131	21–38
	hover.	1.0–1.8	9.4–16	17–25
3D forest	quad.	1.0–1.3	0.5–0.8	7.1–10
	blimp	2.5–3.3	60–75	28–43
fifthelement	quad.	1.0–1.2	3.8–4.9	4.6–6.1
	blimp	1.2–1.7	32–44	5.1– 21

Fig. 6. 95% confidence intervals for the median planning time, as a factor of BEAST's. Gray background indicates a lower bound.

B. Results

The first metric we examine is the planners' ability to solve our benchmark problems in a timely way. Figure 5 shows the fraction of solved problems for each algorithm in each vehicle/map combination. To give a sense of planner behavior, the table shows results with both a 60 second and 300 second CPU time limit. BEAST solves most problems within 60 seconds and still surpasses the other algorithms at 300 seconds, with the exception of the car in the 3 ladder environment, where P-PRM solves 2% more instances. In contrast, P-PRM, KPIECE, and RRT each have certain vehicle/world combinations where they struggle to solve even half of the problems.

To characterize the speed of BEAST, Figure 6 gives 95% confidence intervals on the median slowdown of the other planners relative to BEAST. In other words, a value of 1.9 means that the algorithm took 1.9 times as long as BEAST to find a complete and feasible solution. For cases in which a planner did not find a solution within 300 seconds, we count the instance as solved at the time limit (300 seconds), yielding a lower bound on the true slowdown. Table cells where this approximation was used are shaded gray. The

map	vehicle	60 secs				300 secs			
		BEAST	P-PRM	KPIECE	RRT	BEAST	P-PRM	KPIECE	RRT
open area	car	1	1	0.708	0.980	1	1	1	1
	hover.	1	1	0.880	0.872	1	1	1	1
single wall	car	0.996	0.544	0.448	0.416	1	1	0.908	0.908
	hover.	0.844	0.032	0.420	0	0.984	0.404	0.964	0.136
3 ladder	car	0.980	0.780	0.568	0.432	0.980	1	0.928	0.976
	hover.	0.920	0	0.636	0	1	0.164	0.960	0.012
2D forest	car	1	1	0.192	0.524	1	1	0.436	0.844
	hover.	0.904	0.819	0.225	0.080	0.992	0.972	0.550	0.313
3D forest	quad.	1	1	0.996	0.924	1	1	1	0.996
	blimp	0.992	0.864	0.032	0.160	1	0.992	0.088	0.676
fifthelement	quad.	1	1	1	0.976	1	1	1	1
	blimp	0.996	0.920	0.156	0.480	1	1	0.276	0.592

Fig. 5. Fraction of instances solved in each environment (5 start/goal pairs and 50 seeds = 250 instances each).

map	vehicle	P-PRM	KPIECE	RRT
open area	car	1.0–1.1	1.8–2.3	1.0–1.2
	hover.	1.0–1.1	1.6–1.9	1.4–1.8
single wall	car	1.0–1.1	1.2–1.4	1.0–1.1
	hover.	∞–∞	1.1–1.3	∞–∞
3 ladder	car	1.0–1.1	1.2–1.3	1.1–1.2
	hover.	∞–∞	1.0–1.1	∞–∞
2D forest	car	0.9–1.1	∞–∞	1.4–1.8
	hover.	0.8–0.9	2.8–∞	∞–∞
3D forest	quad.	0.9–1.0	1.0–1.2	1.1–1.4
	blimp	1.0–1.1	∞–∞	1.9–2.4
fifthelement	quad.	0.8–1.0	0.9–1.0	1.3–1.6
	blimp	0.9–0.9	∞–∞	1.0–1.3

Fig. 7. 95% confidence intervals for the median goal achievement time, as a factor of BEAST's.

mean slowdown factors ranging from 0.5–0.8 (KPIECE for quadrotor in 3D forest) to at least 100–131 (KPIECE for car in 2D forest). While the slowdowns vary by vehicle and environment, it is clear that overall BEAST is significantly faster than the other algorithms, often by large factors. KPIECE for the quadrotor in the 3D forest has a low median planning time, but its mean slowdown is 1.8–5.2, indicating that for harder problems it is much slower than BEAST. BEAST's search guidance outweighs the overhead of maintaining its abstraction, yielding high performance in both simple problems (open area) and complex ones (3 ladder, fifthelement). Interestingly, its advantage over its nearest competitor (P-PRM) persists even in non-cluttered environments such as open area and single wall.

While these results indicate that BEAST successfully meets its design goal of fast solution generation, it is natural to wonder about the quality of the generated trajectories, since BEAST ignores trajectory cost. To assess this trade-off, we calculated for each generated trajectory the 'goal achievement time,' that is, the planning time plus the time required to execute the trajectory. Figure 7 shows 95% confidence intervals on the median goal achievement time, relative to BEAST. For unsolved instances, we take the goal achievement time as infinite. The table indicates that the goal achievement time of BEAST is similar to P-PRM's, and better than KPIECE's and RRT's. This is encouraging, as BEAST finds trajectories faster and thus allows increased system responsiveness but apparently at no cost to the ultimate goal achievement time.

Additional experimental results are available in [19].

V. 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 in order to reach 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 focuses on low-cost paths leading to the goal, which can be difficult to find (as in Figure 1). 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 such as 3 ladder.

There are other motion planners besides P-PRM that leverage heuristic cost-to-go. Informed RRT* [20] uses ellipsoidal pruning regions to ignore areas of the state space that are guaranteed not to include a better solution than one already found. This is irrelevant for our purposes here, in which we aim to find a single solution quickly. BIT* [21] 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 [22], making it inapplicable to many problems.

Learning heuristic guidance during search has been previously proposed in both discrete graph search [23] and in motion planning. RSPP [24] uses previous planning episodes to learn heuristics. Adaptive RRT [25] chooses different tree growth strategies based on gained information. SEHS

[26] builds a heuristic emphasizing clearance. STRIDE [27] estimates the sampling density of the configuration space so as to avoid growing the motion tree into unpromising areas. In [28], productive controls to apply to grow the motion tree are learned on-line. Resampl [29] learns characteristics of the configuration space (for geometric planning) on-line to speed up subsequent queries. In contrast to this previous work, BEAST focuses on effort-to-go estimates, rather than coverage or cost-to-go, and learns while solving a single instance.

The idea of estimating probabilities over a graph to minimize solving time also appears in POMP [30], although they estimate collision probabilities along the edges of a PRM, rather than propagation success between regions.

While BEAST was designed as a motion planner, it might be useful for heuristically testing feasibility of motion planning problem instances [31]. In that scenario, solution cost is not just secondary, but irrelevant.

Finding solutions quickly is an important feature in many applications requiring responsiveness, but eventual convergence to an optimal cost solution is also desirable. We believe that BEAST can be an important starting step in an anytime kinodynamic motion planner that quickly finds a feasible solution and then spends its remaining planning time finding improved solutions.

VI. CONCLUSION

We presented a new motion planning approach called Bayesian Effort-Aided Search Trees. BEAST maintains and exploits on-line estimates of propagation effort through the state space to find solutions quickly. Results with a variety of vehicles and environments showed that BEAST found solutions much more quickly than other leading single-query kinodynamic motion planners. 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.

REFERENCES

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

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

[3] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

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

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

[6] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[7] J. T. Thayer and W. Ruml, "Using distance estimates in heuristic search." in *Proceedings of ICAPS*, 2009, pp. 382–385.

[8] ——, "Bounded suboptimal search: A direct approach using inadmissible estimates," in *Proceedings of IJCAI*, 2011, pp. 674–679.

[9] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[10] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," *International Journal of Computational Geometry & Applications*, vol. 9, no. 04n05, pp. 495–512, 1999.

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

[12] E. Plaku, "Region-guided and sampling-based tree search for motion planning with dynamics," *IEEE Transactions on Robotics*, vol. 31, pp. 723–735, 2015.

[13] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Motion planning with dynamics by a synergistic combination of layers of planning," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 469–482, 2010.

[14] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[15] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[16] C. M. Wilt and W. Ruml, "Speedy versus greedy search," in *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 2014.

[17] S. Koenig and M. Likhachev, "D* lite." in *Proceedings of AAAI*, 2002, pp. 476–483.

[18] K. M. Lynch, "Controllability of a planar body with unilateral thrusters," *IEEE Transactions on Automatic Control*, vol. 44, no. 6, pp. 1206–1211, 1999.

[19] S. Kiesel, "Robotics needs non-classical planning," Ph.D. dissertation, University of New Hampshire, 2016.

[20] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "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)*, 2014.

[21] ——, "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 3067–3074.

[22] C. Xie, J. van den Berg, S. Patil, and P. Abbeel, "Toward asymptotically optimal motion planning for kinodynamic systems using a two-point boundary value problem solver," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 4187–4194.

[23] J. T. Thayer, A. Dionne, and W. Ruml, "Learning inadmissible heuristics during search," in *Proceedings of the Twenty-first International Conference on Automated Planning and Scheduling (ICAPS-11)*, 2011.

[24] R. Diankov and J. Kuffner, "Randomized statistical path planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-07)*, 2007.

[25] J. Denny, M. Morales, S. Rodriguez, and N. M. Amato, "Adapting rrt growth for heterogeneous environments," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013, pp. 1772–1778.

[26] C. Chen, M. Rickert, and A. Knoll, "Combining space exploration and heuristic search in online motion planning for nonholonomic vehicles," in *IEEE Intelligent Vehicles Symposium (IV)*, 2013, pp. 1307–1312.

[27] B. Gipson, M. Moll, and L. E. Kavraki, "Resolution independent density estimation for motion planning in high-dimensional spaces," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 2437–2443.

[28] P. Cheng and S. M. LaValle, "Reducing metric sensitivity in randomized trajectory design," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 1, 2001, pp. 43–48.

[29] S. Rodriguez, S. Thomas, R. Pearce, and N. M. Amato, "Resampl: A region-sensitive adaptive motion planner," in *Algorithmic Foundation of Robotics VII*. Springer, 2008, pp. 285–300.

[30] S. Choudhury, C. M. Dellin, and S. S. Srinivasa, "Pareto-optimal search over configuration space beliefs for anytime motion planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-16)*, 2016.

[31] P. Cheng, G. Pappas, and V. Kumar, "Decidability of motion planning with differential constraints," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2007, pp. 1826–1831.