Abstraction-guided Sampling for Motion Planning

University of New Hampshire
Department of Computer Science
Technical Report 12-01

Scott Kiesel, Ethan Burns and Wheeler Ruml
May 1, 2012

Abstraction-guided Sampling for Motion Planning

Scott Kiesel and Ethan Burns and Wheeler Ruml

Abstract Motion planning in continuous space is a fundamental robotics problem that has been approached from many perspectives. Rapidly-exploring Random Trees (RRTs) use sampling to efficiently traverse the continuous and high-dimensional state space. Heuristic graph search methods use lower bounds on solution cost to focus effort on portions of the space that are likely to be traversed by low-cost solutions. In this paper, we bring these two ideas together in a technique called f-biasing: we use estimates of solution cost, computed as in heuristic search, to guide sparse sampling, as in RRTs. Estimates of solution cost are quickly computed using an abstract version of the problem, then an RRT is constructed by biasing the sampling toward areas of the space traversed by low cost solutions under the abstraction. We show that f-biasing maintains all of the desirable theoretical properties of RRT and RRT*, such as completeness and asymptotic convergence to optimality. We also present experimental results showing that f-biasing finds cheaper paths faster than previous techniques. We see this new technique as strengthening the connections between motion planning in robotics and combinatorial search in artificial intelligence.

Key words: motion planning, heuristic search, rapidly-exploring random trees, abstraction

1 Introduction

We begin by recalling Dijkstra's algorithm [4], the well-known search technique for finding paths in a discrete state space graph. Dijkstra's algorithm explores a graph by visiting its nodes in ascending order according to the cost necessary to reach them and it is guaranteed to find a cheapest path from an initial node to any node

Scott Kiesel and Ethan Burns and Wheeler Ruml
University of New Hampshire, e-mail: {skiesel,eaburns,ruml} at cs.unh.edu

in the graph. Unfortunately, the search is unfocused and will explore portions of the graph that lead away from the goal as well as those that lead toward it. To alleviate this problem, the A* algorithm [6] uses a cost-to-go estimate called a heuristic. When a heuristic estimate is available, A* always visits fewer nodes than Dijkstra's algorithm, as it avoids portions of the graph that only participate in high cost solutions.

Rapidly-exploring random trees (RRTs) [11] are a popular technique for motion planning in continuous spaces. The RRT algorithm builds a tree of paths by sampling configurations. The point in the tree nearest to each new sample is steered toward the sample, creating a new path segment and a new node in the tree. RRTs are complete in the limit of infinite samples, however they do not optimize for low cost solutions. Karaman and Frazzoli's RRT* algorithm [7] re-wires the tree when lower cost paths can be found to existing nodes near each sample point. RRT* is both complete and asymptotically optimal. However, much like Dijkstra's algorithm for discrete graph search, RRT* will expend effort exploring portions of configuration space that lead exactly away from the goal as well as towards it.

The main contribution of this paper is a new technique called f-biasing, named after the value f used by A* to order its search effort. Just as A* improves over Dijkstra's algorithm, f-biasing focuses exploration of RRT-based algorithms toward areas that are more likely to lead to the goal configuration, and to do so via low cost trajectories. To use f-biasing, we first solve a discretized and abstracted version of the motion planning problem. Then, using the cost estimates found in the abstracted problem, we bias the location of samples in the RRT so that they are more likely to be drawn from portions of configuration space that contain low cost solutions to the abstracted problem.

After discussing the method in detail, we prove that f-biasing maintains the completeness and convergence properties of RRT and RRT*. We then compare f-biased RRT and RRT* to their unbiased and goal-biased versions using three vehicles of increasing complexity: a simple straight-line vehicle, the Dubins car, and a hovercraft. f-biasing finds its first solutions more quickly in all domains except Dubins car with RRT*, where our current f-biasing implementation has more re-wiring overhead and this is only as fast as the other methods. We also show anytime profiles that demonstrate that f-biasing both solves more problems and is able to improve its solution quality more quickly than other techniques. Finally, we show how f-biased RRT can provide a larger improvement over unbiased RRT than the RRT* algorithm. Broadly, we see this work as strengthening the connections between motion planning in robotics and combinatorial search in artificial intelligence that were pioneered by algorithms like RRT* and R* [16].

2 Previous Work

We begin with a discussion of related work in both heuristic search and robotics.

2.1 Heuristic Search

A* [6] is an optimal search algorithm for discrete graphs [3]. A* visits nodes in increasing order of estimated solution cost $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of the path from the initial node to node n and $h(n)$ is the heuristic value of n, estimating the cost from n to a goal node. In this paper, we are bringing the use of heuristics to the area of continuous motion planning. This raises the question: where do heuristics come from?

One technique for creating heuristics is by relaxing the constraints of the problem. Essentially, this technique adds extra edges between states that do not exist in the original problem. Likhachev and Ferguson [15] provide two examples of relaxation as applied to motion planning problem. The first example is their removal of obstacles from a motion planning problem to create a simpler relaxed problem that can be solved quickly. The second example is the ignoring of vehicle dynamics in order to relax motion constraints. Solutions to these relaxations are lower bounds on the cost-to-go in the original problem and are used to guide search.

Currently, some of the most powerful heuristics used by the search and AI planning communities are created using abstraction. An abstraction is a many-to-one mapping from the search space to a smaller abstract representation of the search space. For example, Remolina and Kuipers's [18] topological maps are a form of abstraction created by mapping regions of space to single nodes in a map. Sturtevant and Geisberger [20] also present an overview and a comparison of recent advances in the area of abstraction-based heuristics for grid pathfinding.

Pattern databases (PDBs) [2] are one of the most popular abstraction-based methods and are closest in spirit to f-biasing. A PDB contains the cost-to-goal for every state in an abstract representation of the search space, computed by performing Dijkstra's single-source shortest path algorithm in reverse from the abstract representation of the goal to every node in the abstract state space. During search, the abstract costs from the PDB are used as admissible heuristic estimates for search states: when a heuristic estimate is needed for a node, the solution cost for the abstract representation of the node is used as the estimate.

2.2 Rapidly-exploring Random Trees

Rapidly-exploring random trees (RRTs) [11] grow a tree from the initial configuration toward random samples in configuration space. Each iteration of the RRT algorithm samples a random configuration, finds the node in the tree that is nearest to the sample, and then adds a new node to the tree by steering the nearest node toward the sample. In the limit of infinite samples, an RRT will densely cover the configuration space.

The RRT* algorithm [7] is a simple modification to the standard RRT algorithm that allows it to find cheaper motion plans faster. Whenever a new node is added to the tree, nearby nodes are updated if they can be reached by a cheaper path via

the new node. The re-wiring performed by RRT* is closely analogous to a common technique used in heuristic search algorithms, such as A*, in which, whenever a cheaper path with a lower g value is found to a node, the cheaper path is used and the more expensive path is discarded. This can be seen as a form of dynamic programming. Unlike A*, however, RRT* makes no use of a heuristic estimator.

Other variants of the basic RRT algorithm have been proposed, such as bidirectional RRT [10]. In this paper, we only evaluate f-biasing on the basic RRT algorithm and RRT*, however any sampling technique, such as f-biasing, could easily be applied to bidirectional RRTs.

2.2.1 RRT Sampling Schemes

Previous authors have also recognized that uniform exploration is not the most efficient choice for a single query motion planning algorithm. There are a variety of previous proposals for biasing sample selection in an attempt to decrease the time required to find the first solution, improve the handling of navigation near obstacles, and increase the exploration of the configuration space. Most of the techniques summarized here are discussed in greater detail by LaValle [12].

Unbiased Random Sampling: Unbiased random sampling, the method that was originally proposed for generating an RRT, has the benefit of covering the configuration space without prejudice and is appropriate for domains where no prior knowledge or only inaccurate knowledge is available. The following biasing techniques attempt to exploit additional information to find better solutions faster.

Goal-biased Sampling: Goal-biased sampling [13] selects the goal configuration, or configurations near the goal, more often than uniform sampling in an attempt to grow the RRT more quickly toward the goal. There are two major flavors of goal biasing. First, the goal configuration itself can be selected as a sample with some fixed probability p, otherwise an unbiased sample is used. The second version of goal biasing selects configurations near the goal instead of only the goal itself. One common method for this is to use a Gaussian distribution [13, 19] around the goal configuration. These both can overcome minor local obstacles, however, if a configuration lies in a heavily obstructed part of the space far from the goal, it will be difficult for the tree to escape the obstructions.

Heuristic-biased Sampling: Urmson and Simmons [21] introduced heuristic-biased sampling, which biases samples to be nearer to those nodes that the RRT reached via lower cost paths. This method has been shown to find cheaper motion plans, however, its biasing is based on the cost of paths found by the RRT regardless of whether or not these paths lead toward the goal. Like Dijkstra's algorithm, heuristic-biased sampling will explore portions of the space that lead away from the goal if it has reached them via cheaper paths than those leading toward the goal. Instead, we would like to sample from areas that we expect to be traversed by cheap trajectories that actually reach the goal.

Path-biased Sampling: The previous method that is most similar to ours is path-biased sampling [22, 9]. While it was developed independently, path-biasing is sim-

ilar because it can be seen as using the solution to an abstract or simplified representation of the motion planning problem such as a discrete grid or visibility graph [17]. An RRT is then constructed by choosing samples along the solution path of the abstract problem with a probability p and uniformly otherwise. Using this technique, samples tend to occur along a possible low cost trajectory from the initial configuration to the goal.

Basic path-biasing fails if the path found in the simple problem doesn't take into account constraints of the complex motion planning problem. To hedge against this possibility, Krammer et al. [9] propose a modified variant that draws samples from a Gaussian distribution around the abstract solution path. As we discuss next, f-biasing uses a more principled approach by selecting samples from areas of the configuration space with a probability based on the solution cost in the abstract space. Effectively, f-biasing takes into account all low-cost paths in the abstract space simultaneously instead of focusing on a single path. Furthermore, we will demonstrate that this is effective even for vehicles with complex dynamics, such as a hovercraft.

3 f-biased Sampling

We have discussed heuristic search and the benefits that it gains by using a heuristic to focus its effort on areas of a search space that reside on low cost solutions. Next, we saw that many of the most powerful state-of-the-art heuristics are created by using abstraction, and lastly, we described RRTs, which use sparse, uniform random sampling to explore the continuous and high-dimensional nature of motion planning problems. f-biased sampling combines these three ideas: heuristic search, abstraction, and sample-based motion planning. The first step in using an f-biased RRT is to create an abstract representation of the motion planning domain. Next, Dijkstra's algorithm is used to pre-compute the cost of the shortest path through each abstract node from the initial configuration to the goal in the abstract space, as in PDBs. Like a heuristic, these abstract solution costs give the ability to focus the RRT's growth toward configurations that map to abstract states with low costs. We now explain each of these steps in greater detail.

3.1 Abstraction

The abstraction is represented by a weighted directed graph that is small enough to be searched exhaustively with Dijkstra's algorithm. There are many possible techniques for generating an abstract representation of a problem. In our implementation, we use a simple uniform discretization of configuration space to create an n-dimensional grid, where n is less than or equal to the dimensionality of the configuration space. Each vertex in the abstract graph is a discrete configuration that

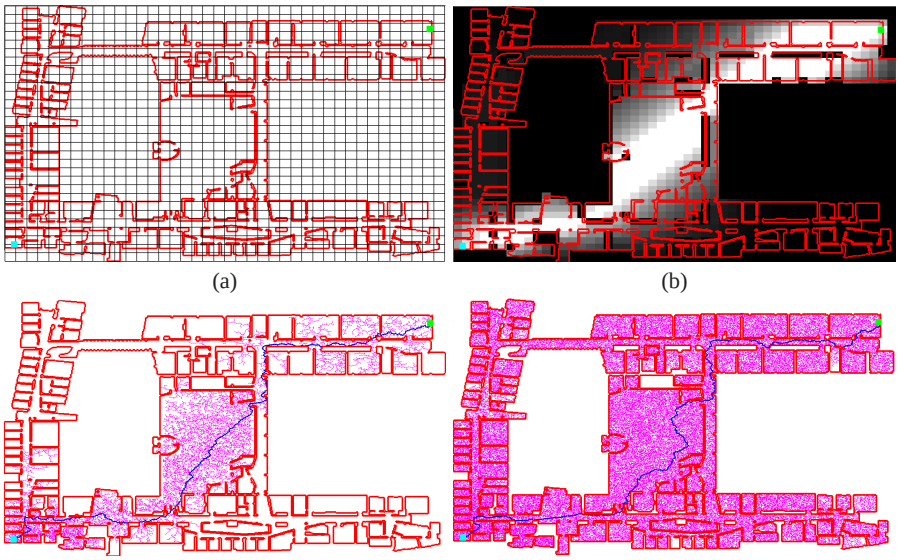

Fig. 1 An example map showing abstraction (a), f values (b), an f-biased RRT (c) and regular RRT (d).

represents all configurations in the continuous space that fall within its Voronoi hyper-rectangle. Adjacent vertices in the abstract graph are connected via an edge if neither vertex is obstructed by an obstacle. In our implementation, a vertex is obstructed if its discrete configuration is contained within an obstacle. The weight of each edge reflects an estimate of the cost of the navigating between the two discrete configurations that it connects.

Figure 1(a) shows a polygonal map of the second floor of our building along with a possible abstraction, represented as a coarse grid overlaid on the continuous domain. Each cell of the coarse grid is a vertex in the abstract graph and the graph has eight-connectivity. This is an extreme simplification, but as our experiments will show, it suffices to guide motion planning.

For each motion planning query, we map the initial and goal configurations to their abstract nodes in the abstract representation of the state space. We then compute the cost of the path from the initial node through each abstract node to the goal node. Because we use a discrete abstraction, this can be done in linear time in the size of the abstract space by using two calls to Dijkstra's single-source shortest path algorithm: one that computes the shortest path from the initial node to each node, $g(n)$ in A* terminology, and another that computes the shortest path from each node to the goal node, $h(n)$. The sum of these values gives the cheapest cost of a solution path passing through the given node, $f(n) = g(n) + h(n)$.

Figure 1(b) shows the f values for a motion planning problem using the abstraction from Fig. 1(a). The initial configuration is shown as a light blue square in the lower-left corner and the goal is shown as a green square in the upper-right corner.

Each abstract node is shaded, with black representing high f cost. As we can see, even this simple abstraction suffices to uncover that it would be more desirable to focus RRT growth into the lighter areas of the map while spending less time considering the dark portions.

3.2 Growing an f-biased Tree

To create an f-biased RRT, we proceed as in the standard RRT or RRT* algorithm, however, more samples are taken from configurations that correspond to low cost abstract nodes. To accomplish this, each node is assigned a score so that nodes with low f values have high scores and nodes with high f values have low scores. These scores are then normalized to sum to 1 and the normalized scores give the probability with which an abstract node is selected for sampling. Once an abstract node is selected, a sample from the concrete configuration space is drawn uniformly from its preimage—the set of concrete configurations that map to the selected node in the abstract space.

The score for each abstract node is given by $s = f_{\min}^{\omega} / f^{\omega}$, where $f_{\min}$ is the minimum f value of all abstract nodes and ω is a configurable parameter representing the strength of the f-bias. Increasing ω increases the influence of the abstract nodes that are closer to $f_{\min}$, narrowing the corridor from which most of the samples are drawn. Decreasing ω decreases the influence of the abstract nodes that are closer to $f_{\min}$ and increases the amount of exploration. In our experiments, we used $\omega = 4$ as it was found to give good performance in a small set of preliminary experiments.

In some cases, an abstract node has an f value of infinity, for example, if it resides within an obstacle. We would still like to generate samples from these areas, so when f is infinite, we define the score to be $s_{\min}/2$ where $s_{\min}$ is the minimum score of all nodes with finite f. The nth abstract node is selected for sampling with probability $p_n = s_n / \sum_{i=0}^{N-1} s_i$. Finally, a sample is generated uniformly from among all possible configurations in the preimage of the selected node. Figure 1(c) shows a completed f-biased RRT, along with its solution path. For comparison, Fig. 1(d) shows the first solution found by an unbiased RRT. Notice that, in the f-biased RRT, most of the exploration is focused in the lighter cells that reside along the diagonal between the initial configuration and the goal. The unbiased RRT required many more samples and explored the entire map.

4 Theoretical Analysis

Previous results on RRT and RRT* are robust enough to survive the bias introduced by our technique.

Lemma 1. *Under f-biasing, there exists a positive constant that bounds from below the probability of selecting each configuration.*

Proof: Under f-biasing, every abstract node has a positive probability of being selected to sample within. Every configuration in the preimage of an abstract node has positive probability of being sampled. □

Theorem 1. *Using f-biased sampling does not disrupt the probabilistic completeness of the RRT or RRT* algorithms.*

Proof: Lemma 1 is exactly the condition required by the completeness proof for RRT given by LaValle and Kuffner [14], thus completeness is maintained. The completeness proof of Karaman and Frazzoli for RRT* [7] is inherited directly from RRT, thus RRT*'s completeness is also preserved. □

Theorem 2. *Using f-biased sampling does not disrupt the asymptotic optimality of the RRT* algorithm.*

Proof: The proof of RRT*'s asymptotic optimality [7] relies on the rewiring step to monotonically decrease path costs, which requires positive probability of adding any configuration to the vertex set. This property is ensured by Lemma 1. Said another way, RRT* merely rewires the same vertex set as constructed by RRT. Using f-biasing preserves non-zero probability of generating every possible RRT vertex set, hence it preserves asymptotic optimality. □

5 Experimental Results

Next, we evaluate the performance of f-biased RRTs experimentally on three different path planning domains.

5.1 Implementation Details

We attempted to obtain a copy of the RRT* implementation by Karaman and Frazzoli [7] for comparison, however, the source code was not available at the time of our request. Instead, we wrote our own implementation of RRT and RRT* in C++ using the same K-D tree implementation that was used by Karaman and Frazzoli (available from `http://code.google.com/p/kdtree/`). Our RRT* implementation also used their technique for reducing the size of the ball from which nodes are considered for re-wiring as more samples are generated. All techniques in our comparison used the same implementation and data structures; the only difference between the techniques was the decision of where in the configuration space the samples were generated. All experiments were performed on a 3.16 GHz Core2 duo PC with 8GB RAM running Linux.

For f-biasing, the abstract node from which to sample was selected in constant time by inserting a reference to each abstract node multiple times into a large array

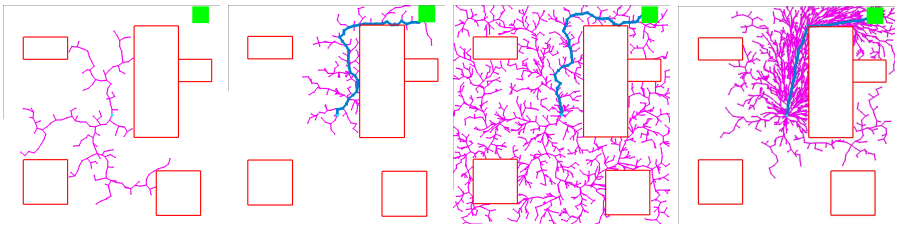

Fig. 2 The example of Karaman and Frazzoli after 235 iterations with unbiased RRT* (a) and f-biased RRT* (b) and after 2000 iterations (c) and (d).

to approximate its probability relative to the least probable node. An index into this array was then chosen uniformly at random. An alternative, less memory hungry, approach is to use binary search to select the node. To reduce the time spent by binary search, clustering can be used to group the abstract nodes into a small fixed number of equiprobable bins that can be searched very quickly.

In timing results with f-biasing, we do not include the time required to build the abstraction since it can be computed once for a given map and stored. Typically, the time required to build the abstraction was only a few seconds, most of which was spent testing if abstract nodes are blocked by obstacles in the configuration space. These tests can easily be performed in parallel, allowing the abstraction creation time to be greatly decreased with modern multicore hardware. Our timing results do include the time required to perform the Dijkstra shortest-path pre-computation step for each instance, because this must be performed for each individual motion query. Our implementation runs both Dijkstra searches in parallel as they are completely independent of one another. Regardless, this time was found to be quite insignificant.

5.2 Straight-line Vehicle

Our first set of experiments uses a very simple vehicle motion model from Karaman and Frazzoli [7] that we call the 'straight-line vehicle.' The straight-line vehicle moves straight and can instantly turn to any angle. The objective to minimize is the path length.

We begin by comparing unbiased RRT* with f-biased RRT* on a reproduction of the map used in Karaman and Frazzoli's Fig. 1 [7]. They used this simple map to show the benefits of RRT* over basic RRTs. Likewise, Fig. 2 uses this map to show the benefit of f-biasing. Figures 2(a) and 2(b) show unbiased RRT* and f-biased RRT* respectively after 235 samples, when f-biasing finds its first solution. Figures 2(c) and 2(d) show the state of both algorithms after 2000 samples. We can see that combining the sampling of RRTs with guidance from heuristic search

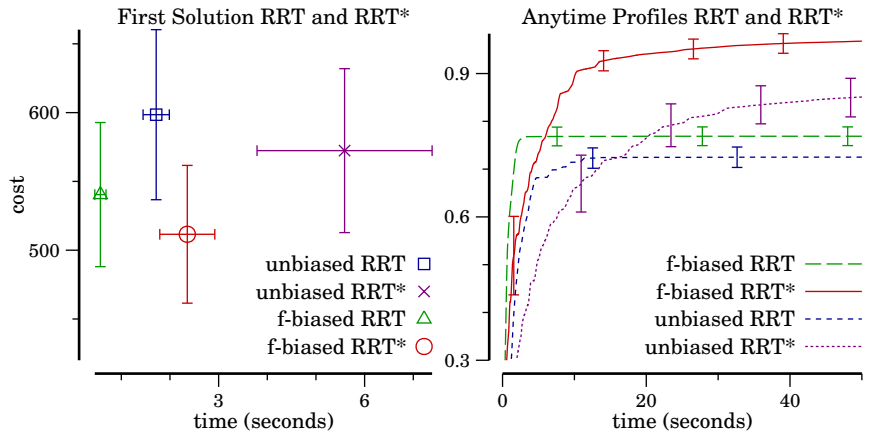

Fig. 3 Straight-line vehicle: f-biasing and RRT* improvement over unbiased RRT.

caused f-biasing to find its first solution more quickly and enabled it to decrease the solution cost more quickly too.

The map in Fig. 2 is very simple, so our next results are on a set of 100 path planning problems given by uniformly selected initial and goal locations on the more realistic map of Fig. 1. The abstraction used for f-biasing was a uniform eight-connected grid of resolution 12x10.

First, we look at the improvement of f-biasing over standard RRT compared to the improvement of RRT* over RRT. The left plot in Fig. 3 shows the first solution time and cost for RRT and RRT* with and without f-biasing. The x axis shows the first solution time in seconds and the y axis shows the first solution cost. Each glyph represents the mean over the 99 instances that were solved by all techniques with RRT and the 100 instances solved by all techniques with RRT* within a 90 second time limit. Error bars show the 95% confidence intervals on the mean. We can see from this plot that f-biased RRT actually found its first solution significantly more quickly than all alternatives and in addition, its first solution costs tended to be slightly cheaper than that of unbiased RRT*. f-biased RRT* gave the best solution cost and took only slightly longer than unbiased RRT.

RRT and RRT* are naturally anytime algorithms; they provide a stream of solutions of decreasing cost as they are given more time. One common way to compare anytime algorithms is by comparing their *anytime profile*, i.e., solution cost over time. We ran each biasing technique twice with the same random seed for 90 seconds with RRT and RRT* on each of our 100 instances. The first run computed the solution cost achieved at each sample. Because there are many iterations, this cost computation required a non-negligible amount of CPU time, so the second run measured the time at which each sample was taken without the cost computation. This data was used to build anytime profiles.

The right plot in Fig. 3 shows the anytime profiles for RRT and RRT* with and without f-biasing. The data points were computed in a paired manner by finding the

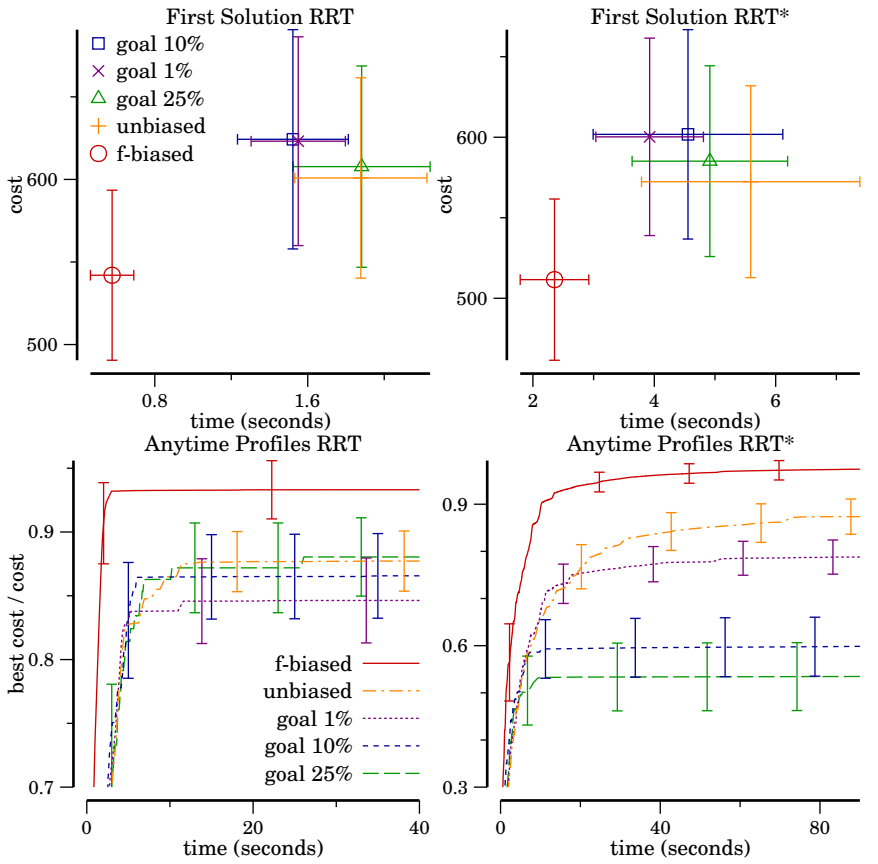

Fig. 4 Straight-line vehicle: first solution times and anytime profiles for RRT (left) and RRT* (right).

best solution found on each instance by the algorithms in the given plot and dividing this by the incumbent cost at each time value on the same instance. By initializing incumbent scores to infinity, this technique allows for comparison at times before all instances are solved. The lines show the mean over the instance set and the error bars show the 95% confidence interval on the mean. The plot shows that f-biased RRT and RRT* both find cheaper solutions faster than their unbiased counterparts.

Next, we compare f-biasing to both goal-biased and unbiased RRT and RRT*. The top row of Fig. 4 shows the time and cost of the first solution for f-biasing, goal-biasing with 1%, 10% and 25% of the samples being the goal configuration and unbiased RRT and RRT*. f-biasing found its first solutions significantly more quickly than the other techniques and the cost of its first solutions tended to be lower. The bottom row of Fig. 4 shows anytime profiles. From the left plot, we can see that when used in the RRT algorithm, f-biasing dominated the other techniques.

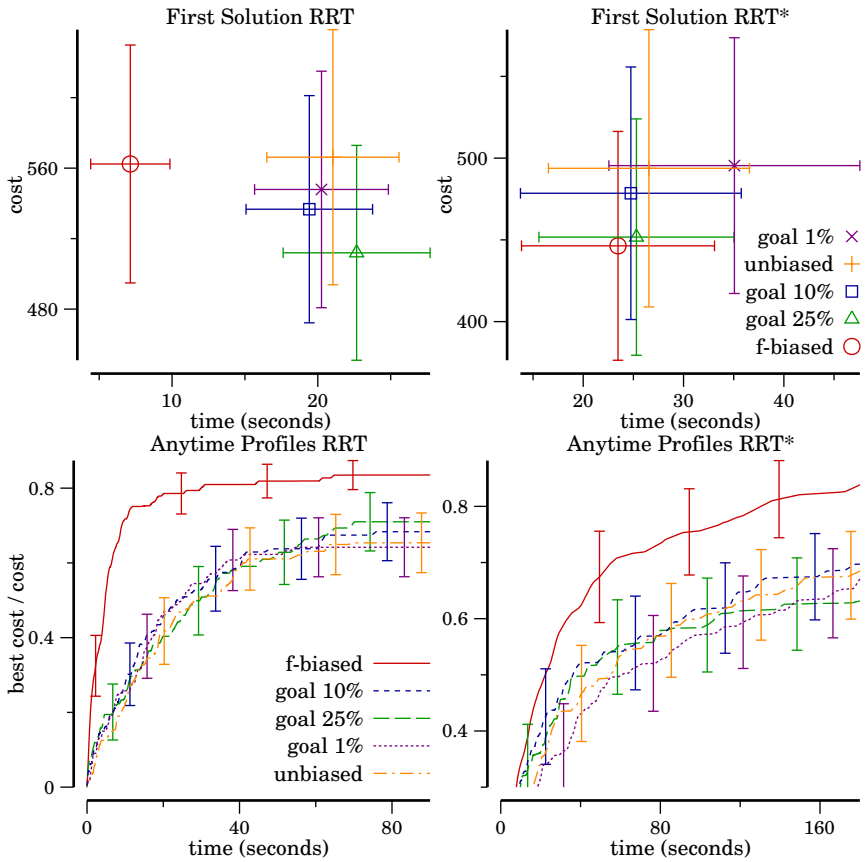

Fig. 5 Dubins car first solution times and anytime profiles for RRT (left) and RRT* (right).

In the right plot, we can see the same behavior for RRT* except that more time was required to approach the best cost solution. This is likely because of RRT*'s convergence to optimality: the best solution found by RRT* was much cheaper than the best found by RRT and more time was used to find it. Also, each iteration requires re-wiring.

5.3 Dubins Car

In this section, we evaluate the performance of f-biasing with the Dubins car [5], which has an x and y location and heading θ that is constrained by a fixed turning radius. The abstraction used by f-biasing on this domain used a uniform grid of discrete x, y and θ combinations with dimensions 75x65x4. In this set of experiments,

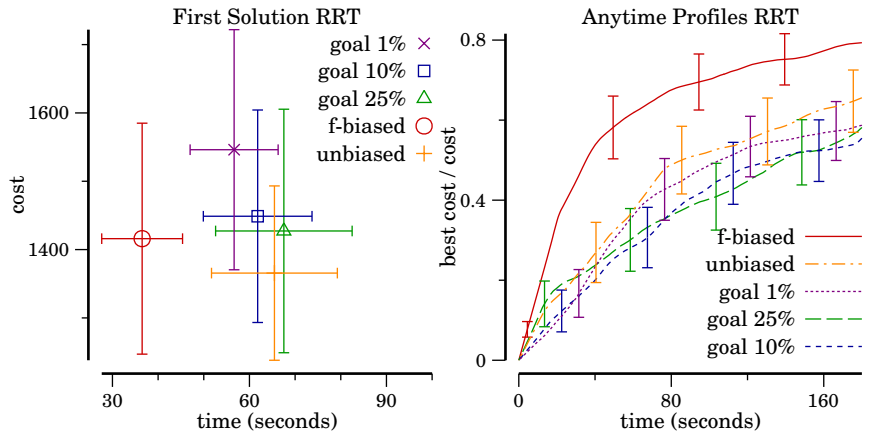

Fig. 6 Hovercraft first solution times and anytime profile for RRT.

we used the same instances that were used for the straight-line vehicle with a time limit of 90 seconds for RRT and 180 seconds for RRT*.

The top two plots in Fig. 5 show the time and cost of first solutions. For RRT, f-biasing found its first solutions significantly more quickly than the other techniques. For RRT*, however, none of the techniques found their first solution significantly faster than the others. f-biasing did not find its first solution faster in this setting because its biased samples created a very dense tree and so RRT* performed a lot more expensive re-wiring. The bottom two plots show anytime profiles. f-biasing had a better profile than all other techniques on both algorithms even though it performed fewer samples within the time limit for RRT*. This is because f-biasing both solved more instances and was able to find cheaper solutions with fewer samples than the other methods.

5.4 Hovercraft

The final domain that we present is path planning for a simple hovercraft. Each configuration consists of $\langle x, y, \theta, \delta x, \delta y, \delta\theta \rangle$. x, y and θ represent the craft's position and orientation. δx and δy represent the current translational rate in each respective direction and $\delta\theta$ represents the rotational velocity. This models a simple hovercraft with two fans: one propels the craft in the direction θ and the other applies rotational force in either direction. This domain has the largest dimensionality and presents the most difficult motion model of all domains considered in this paper.

For the experiments in this domain, we used 100 random start and goal configurations on the map from LaValle and Kuffner [14]. The abstraction used for f-biasing was the same as used for the Dubins car with dimensions 26x26x4. f-biased RRT solved 90% of all instances within a 180 second time limit whereas goal-biased RRT

with its best setting (1%) only solved 74% of the instances and unbiased RRT only solved 75%. The left plot in Fig. 6 shows the first solution costs and times for the 43 instances solved by all algorithms within the time limit. The first solution costs from f-biasing were not significantly different from that of the other techniques, however, it found these solutions significantly faster. The right plot shows the anytime profile, where we can see that f-biasing gave the best performance. Achieving good performance with such a basic abstraction for this complex domain suggests that f-biasing is robust to the choice of abstraction.

6 Discussion

As we point out in Section 5.3, f-biased RRT* is not able to generate samples as quickly as unbiased and goal-biased RRT* because it builds a denser tree and therefore requires more re-wiring at every sample. The sample speed of f-biasing can be increased in a couple of ways. First, Karaman and Frazzoli's [7] k-nearest technique can be used to fix the number of nodes tested for re-wiring at k, instead of checking all nodes within the ball. A second possibility is to chose the ball size used to test for re-wiring dynamically based on the sample density of the selected abstract node and its neighbors. Even without these optimizations, our results show that f-biasing performs favorably as it is able to find cheap solutions with fewer samples than alternative methods.

While the results presented in Fig. 6 show that f-biasing can give good performance even with a simplistic abstraction, it is worth noting that the choice of abstraction can be important. If the abstraction is too coarse, then it may not account for important obstacles in the planning problem. If this occurs, then the sampling can be biased toward regions of space that contain only infeasible plans due to the unaccounted obstacles. Given this, one might assume that a finer discretization of the abstract space will always perform better, as it is more informative, however, we have found that coarser discretizations actually tended to perform better in our experiments.

We have shown that f-biasing works well for constructing RRTs. We are also interested in trying to combine these ideas with other types of motion planning techniques. Probabilistic roadmaps (PRMs) [8] are a popular alternative to RRTs that work by constructing a roadmap of feasible paths between points that are sampled randomly from the configuration space. Once the roadmap has been constructed, motion planning queries can be performed by connecting the initial and goal configurations to any points on the roadmap and performing a fast discrete graph search.

As with RRTs, it is possible to bias the selection of locations used to create a PRM. One possibility for using the ideas presented in this paper in conjunction with PRM construction would be to compute the *betweenness centrality* [1] of nodes in an abstract graph. Betweenness centrality is a measure of the number of shortest paths upon which a node in a graph resides. Sampling from locations in the abstract graph with higher betweenness centrality may lead to more effective RPMs as the

nodes in the roadmap may reside in areas of the space that are used in many shortest paths.

7 Conclusion

We have presented f-biasing for RRTs, a new technique that combines guidance from heuristic search with sparse sampling techniques from robotics. f-biasing effectively focuses the growth of an RRT on areas of configuration space that are traversed by low-cost paths in an abstract representation of the problem. This allows f-biased RRTs to find cheaper motion plans more quickly than other sampling techniques. Our experimental results demonstrate that this new technique outperforms unbiased and goal-biased RRT and RRT* on three different vehicle motion models: a straight-line vehicle, a Dubins car, and a hovercraft. This work strengthens the connections between motion planning in the robotics community and heuristic search in artificial intelligence. We feel that there are many additional analogies that can be drawn between these two areas and we plan to explore them in future work.

Acknowledgments

We gratefully acknowledge support from NSF (grant IIS-0812141) and the DARPA CSSG program (grant HR0011-09-1-0021). We would also like to thank Jordan Thayer for his useful insight and help on an early draft of this paper.

References

[1] Brandes U (2001) A faster algorithm for betweenness centrality. The Journal of Mathematical Sociology 25(2):163–177
[2] Culberson JC, Schaeffer J (1998) Pattern databases. Computational Intelligence 14(3):318–334
[3] Dechter R, Pearl J (1988) The optimality of A*. In: Kanal L, Kumar V (eds) Search in Artificial Intelligence, Springer-Verlag, pp 166–199
[4] Dijkstra EW (1959) A note on two problems in connexion with graphs. Numerische Mathematik 1:269–271
[5] Dubins LE (1957) On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. American journal of mathematics 79:497–516
[6] Hart PE, Nilsson NJ, Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics SSC-4(2):100–107

[7] Karaman S, Frazzoli E (2011) Sampling-based algorithms for optimal motion planning. International Journal of Robotics Research 30:846–894
[8] Kavraki L, Svestka P, Latombe JC, Overmars M (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation 12(4):566–580
[9] Krammer L, Granzer W, Kastner W (2011) A new approach for robot motion planning using rapidly-exploring randomized trees. In: Proceedings of the Ninth IEEE International Conference on Industrial Informatics, pp 263 –268
[10] Kuffner JJ Jr, LaValle S (2000) RRT-connect: An efficient approach to single-query path planning. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-00), vol 2, pp 995 –1001
[11] LaValle SM (1998) Rapidly-exploring random trees: A new tool for path planning. Tech. rep.
[12] LaValle SM (2006) Planning Algorithms. Cambridge University Press, Cambridge, U.K., URL `http://planning.cs.uiuc.edu/`
[13] Lavalle SM, Kuffner JJ Jr (2000) Rapidly-exploring random trees: Progress and prospects. In: Proceedings of the Fourth International Workshop on Algorithmic Foundations of Robotics (WAFR-00), pp 293–308
[14] LaValle SM, Kuffner JJ Jr (2001) Randomized kinodynamic planning. International Journal of Robotics Research 20:378–400
[15] Likhachev M, Ferguson D (2009) Planning long dynamically feasible maneuvers for autonomous vehicles. The International Journal of Robotics Research 28(8):933–945
[16] Likhachev M, Stentz A (2008) R* search. In: Proceedings of the Twenty-third National Conference on Artificial Intelligence (AAAI-08)
[17] Nilsson NJ (1969) A mobile automaton: An application of artificial intelligence techniques. In: Proceedings of the First International Joint Conference on Artificial Intelligence (IJCAI-69)
[18] Remolina E, Kuipers B (2004) Towards a general theory of topological maps. Artificial Intelligence 152(1):47–104
[19] Song G, Amato NM (2001) Using motion planning to study protein folding pathways. In: Proceedings of the Fifth Annual International Conference on Computational Biology, pp 287–296
[20] Sturtevant N, Geisberger R (2010) A comparison of high-level approaches for speeding up pathfinding. Proceedings of the Fourth Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-10) pp 76–82
[21] Urmson C, Simmons R (2003) Approaches for heuristically biasing RRT growth. In: Proceedings of the IEEE/RSJ International Conference on Robotics and Systems (IROS-03)
[22] Vonásek V, Faigl J, Krajník T, Přeučil L (2009) RRT-path a guided rapidly exploring random tree. In: Robot Motion and Control, Lecture Notes in Control and Information Sciences, vol 396, pp 307–316