Utility-Guided Random Trees

Technical Report 06-29

Brendan Burns Oliver Brock
Laboratory for Perceptual Robotics
Department of Computer Science
University of Massachusetts Amherst

June 2006

Abstract

Random trees planning has been used to solve connectivity problems in a variety of problem domains. In particular, these approaches are some of the most successful approaches to robotic motion planning. In this work we examine the broad array of random-tree that have been proposed for motion planning. From this survey we distill a general algorithmic framework for random-tree planning that abstracts the shared functionality used by all planners. Examination of this algorithm shows the parts of random-tree planning that have been explored in depth and those that have received little attention. Using this analysis, we show that two functions: exploration length and connecting growth have been largely unexplored. To address this, we propose utility-guided implementations of these functions. These functions are incorporated into a complete utility-guided random-tree motion planner. Experimental results with this planner in challenging artificial and real world planning problems indicate that this approach is significantly more efficient and reliable.

1 Introduction

The instantaneous state of an evolving system can be conveniently represented as a point in the associated state space. In such a representation, one commonly encountered problem is determining if a system can transition from its current state to a desired goal state. Expressed in state space: Is it possible to connect the current state to a desired state by a sequence of allowable state transitions?

Problems of this kind occur in a variety of domains, such as robot motion planning. Given two configurations of a robot and a geometric description of its environment, we would like to determine if the robot can move between the two configurations without colliding with objects in the environment. We can cast this problem as a connectivity problem in state space. In this case, the configuration space of the robot is considered is the state space in which we determine a sequence of collision-free transitions that connect the robot's current configuration to the desired goal configuration.

The introduction of sampling-based motion planners [12] has lead to the development of a large number of algorithmic techniques to solve this type of state space connectivity problem. These techniques are collectively referred to as single-query motion planners. Such planners can solve state space connectivity problems for many problems encountered in robotics. For example, they are able to generate collision-free motion of humanoid robots [6, 14], to determine disassembly motions for manufacturing [8], to plan behaviors for modular robots [23], or to simulate the motion of flexible molecules [5]. These planners can also solve problems in state spaces that incorporate kinodynamic constraints of the system [16], such as the

docking of space vessels [4, 19]. They are also able to address state spaces with second-order nonlinear dynamic constraints, such as drift and under-actuation [15].

Single-query planning methods have also proven effective for solving general connectivity problems in state spaces. The techniques used to generate the motion for robots and molecules can easily be adapted for the generation of hybrid control systems [2] and to validate the correctness of such systems [7], even in the context of composite state space, representing a multitude of independently evolving, interacting systems [13].

In this paper, we propose an abstract framework for the tree-based exploration of state spaces and argue that existing approaches can be viewed as specific instantiations. The framework exposes a number of parameters that influence the effectiveness of state space exploration. We introduce utility-guided rapidly-exploring random trees (RRTs) as a novel method of solving connectivity problems in state spaces. Our method differs from existing approaches in that the exploration "strategy" is adapted in response to information obtained during the exploration. Adaptation is performed by adjusting the parameters exposed by the abstract framework. We present experimental evidence in the context of robot motion planning to demonstrate that utility-guided RRTs increase the effectiveness of exploration significantly.

2 Exploration With Random Trees

Tree-based single-query motion planners are well suited to solving connectivity problems in high-dimensional spaces. These planners grow trees of valid state transitions to determine if two states can be connected. One tree is rooted at the initial state, the other at the goal state. The two trees are grown towards each other by adding branches obtained with sampling techniques. A path in these trees represents a valid sequence of state transitions through state space. If the two trees can be connected, the planner has determined a solution to the connectivity problem.

Though random-tree algorithms are capable of exploring arbitrary spaces, in the following we restrict our examination of these algorithms to robot motion planning, and more specifically to bi-directional single-query planning in configuration spaces. This restriction serves to focus the analysis, but the results extend to general state spaces and tree-based algorithms that only grow a single tree.

To facilitate the analysis of random-tree planning we distill the common elements of the many existing random tree planners into a general algorithmic framework for random-tree state space exploration. This generalized algorithm exposes the individual modular components that compose a tree-based planning algorithm. First, we present this generalized algorithm and identify among the modular components those that have been explored in some depth by previous work. We also identify those components that have largely remained unexplored. In subsequent sections we propose specific implementations for these under-examined components and empirically show that these novel implementations significantly improve planner performance.

2.1 Generalized bi-directional random-tree planning

Bi-directional tree-based planners solve the single-query problem by growing two configuration space trees, one rooted at the start configuration and one rooted at the goal. The trees are alternately grown until they are connected to each other. Tree growth occurs in one of two different modes: connecting growth or exploratory growth. Connecting growth attempts to connect the two trees together. Exploratory growth attempts to expand a tree's representation of configuration space connectivity. In the following we describe how these two types of growth are coordinated to implement a complete planning algorithm.

The generalized tree-based algorithm begins with an attempt to connect the trees together. While this greedy initial growth only succeeds in extremely simple configuration spaces, it often directs the exploratory

BiDirectionalRandomTreePlanner(q_s,q_g)

1: $T_a = T(q_s), T_b = T(q_g)$
2: Connect(T_a, T_b)
3: **while** (no path is found)
4: q_n = SelectNode(T_a)
5: $\vec{v}$ = SelectDirection(T_a, q_n)
6: d = SelectLength($T_a, q_n, \vec{v}$)
7: Extend($T_a, q_n, \vec{v}, d$)
8: Connect(T_a, T_b)
9: Swap(T_a, T_b)

Figure 1: The generalized bi-directional random tree motion planning algorithm

growth toward the region of configuration space most relevant to the particular query. Several different algorithms for connecting growth have been proposed [17, 21]. Regardless of the implementation, connecting growth attempts to use the available representation of configuration space connectivity to identify a successful path.

When a connecting growth fails, exploratory growth is necessary to expand a random-tree's approximation of configuration space connectivity. Exploratory growth initially selects a node in the random tree as the root of the new tree branch. Once a node has been selected, the planner uses its location in configuration space to select an exploratory direction for the expansion. This direction may be a simple linear path through configuration space or it may be a more complex movement through state space. Once a node and direction have been selected the final step is to select the length of the growth. When all parameters are selected, the planner attempts the expansion. If the state transition is valid, the tree is expanded to include the transition. The planner can now attempt additional connecting growth to find a path. The process of alternating connecting and exploratory growth continues for both trees until they can be connected and a valid path or sequence of state transitions is identified. This general algorithm is given in pseudo code in Figure 1.

2.2 Related work in random-tree planning

The performance of a single-query planner is defined by the speed with which it can discover a collision-free path connecting two specified configurations. Achieving maximal efficiency requires careful implementations of each function in the generalized planning algorithm given in Figure 1. There has been considerable research into tree-based planning that provide implementations for some or all of these functions. A summary of these implementations is shown in Table 1. In the following, we discuss these approaches in detail.

The most widely used tree-based planners are the rapidly-growing random tree (RRT) family. The original RRT-Connect [17] algorithm simultaneously selects a node for expansion and a direction of expansion using the Voronoi bias that directs expansion toward large unexplored regions of configuration space. More recent research has refined this Voronoi bias using the dynamic domain [22] and the adaptive dynamic domain [10]. Once a node and an expansion direction have been selected, all of these methods use a static length for the expansion. OBRRT [20] uses the Voronoi bias to select nodes for expansion, but uses a hybrid approach for expansion direction and distance. OBRRT uses weighted random selection to choose between six different methods of selecting an expansion method. Once an exploratory expansion has occurred all four approaches attempt to connect the newly added node to the closest node in the other configuration

Algorithm	Node Selection	Expl. Trajectory	Expl. Length	Connection
RRT-Connect	Voronoi bias	Voronoi bias	constant	nearest node
DD-RRT	DD V. bias	DD-V. bias	constant	nearest node
ADD-RRT	ADD V. bias	ADD-V. bias	constant	nearest node
OBRRT	Voronoi bias	hybrid	hybrid	nearest node
Exp. Spaces	tree density	tree density	$<$ constant	$<$ constant
Guided Exp. Spaces	heuristic	heuristic	$<$ constant	$<$ constant
Adapt. Single-Query	none	none	none	heuristic

Table 1: A table summarizing different tree-based planners in the context of the generalized framework for tree-based motion planning. In the table, DD refers to the Dynamic Domain [22] and ADD refers to the Adaptive Dynamic Domain [10]. In the context of motion planning, a state transition corresponds to

space tree.

The expansive space approach to tree-based planning [9] selects nodes for expansion based upon the density of nodes in the configuration space. Nodes that are in sparsely sampled areas are more likely to be selected. Rather than a single expansion, the selected node is expanded in multiple directions. These directions are selected with a bias toward directions that lead to areas of the configuration space sparsely covered by the tree. The distance of expansion must be less than a neighborhood threshold. After each expansion, connection is attempted between every pair of nodes in the two trees whose distance is less than a threshold. The expansive space approach has also been extended [19] to use a node selection function that incorporates a heuristic cost function and the degree of the node in addition to tree density to bias node selection.

The adaptive framework for single-query planning [21] is a hybrid approach to tree-based planning. It focuses solely on connecting trees together. Only connecting growth is used to expand the trees. In each iteration, a pair of nodes, one in each tree, and a path planning algorithm for connecting them is selected. These nodes and algorithm are selected using a heuristic scoring function that weighs local configuration space properties of the start and goal nodes, global properties of the space between the configurations and planner characteristics.

It can be seen that these previous research in tree-based planning can be viewed as concrete implementations of the same generalized algorithm. When viewed through this generalization, comparisons between existing tree-based methods are simplified. Through this comparison it can be seen that the largest amount of research has explored the selection of the node to expand and to a lesser extent, the direction of this expansion. Relatively little research has examined the role of the exploration length or the algorithm for connecting growth. Interestingly, the one approach that examines connecting growth [21] does not use exploratory expansion. The following section explores an application of the general utility-guided framework for selecting the exploration length and directing connecting growth.

3 Utility-Guided RRTs

The generalized algorithm for random-tree planning identifies a set of modular components that make up a random-tree planner. Though the implementation of all of these modules affects the performance of the planner, two of these components, exploration length and connecting growth, are largely unexplored in previous research. In the following we adapt the utility-guided framework [3] to develop novel implementations of these components.

3.1 Utility-guided planning

Fundamentally, motion planning is search in the configuration space for a successful path. Utility-guided planning is a general framework for structuring this search toward expansions with maximal expected utility. The utility of an expansion step is estimated based on a probability distribution over possible outcomes of the expansion, such as immediately discovering it to be obstructed, or some portion of the state space that provides a partial solution to the connectivity problem.

Let the set E contain a set of possible outcomes resulting from some expansion of a random tree. The expected utility [11] of that expansion is given by:

$$\text{ExpectedUtility}(E) = \sum_{e \in E} P(e)\text{Utility}(e)$$

This formulation of expected utility exploits two sources of structure in the planning problem to guide exploration. $P(e)$, the probability of a successful expansion, is estimated by an approximate model built from past experience. This model will be described in Section 3.4. The utility $\text{Utility}(e)$ of that outcome is a task-specific function that captures the "goodness" of the outcome with respect to the specific planning problem. Selecting exploratory expansions with maximal expected utility, produces expansions that provide maximal progress toward the planner's goals. The following presents an application of the utility-guided framework to generalized random-tree planning.

3.2 Utility of exploration length

The task of exploration is to improve the random tree's approximate representation of the configuration space. Because of this, small movements through configuration space are of limited value to the planner. They are unlikely to significantly improve the representation of the connectivity. We model this heuristically in the utility function by a constant: τ_{min}. Explorations shorter than this distance have no utility to our planner. At first examination this might seem to limit the planner's ability to solve problems with passages that are narrower than τ_{min}, however this is not the case. τ_{min} restricts the length of any particular exploration but does not limit the granularity of the resulting plan. In fact, τ_{min} often improves planning through narrow passages because it eliminates repeated "zig-zagging" in favor of longer connections aligned with the narrow passage.

Long configuration space expansions also have little utility. The purpose of any exploration is to expand the tree's coverage of configuration space to new connectivity regions. When an trajectory has moved a sufficient distance in configuration space it has expanded the visibility of the tree sufficiently to enable a novel attempt at connecting the two trees. Further exploration requires significant computation for little or no improvement likelihood that the subsequent connection attempt will be successful. Because the ultimate goal of the planner is efficiency, the utility of wasted computation is negative. A graphical example of this is given in Figure 2. Once the exploration proceeds beyond the "shadow" of the obstacle and has a direct connection to the other tree, further exploration (shown as the dotted portion of the line) does not provide further improvement. This maximum distance is modeled in the utility function by the upper threshold τ_{max}.

When the length of an exploration is between these thresholds, its utility is proportional to its length. All else being equal, this biases exploration toward quicker coverage of the configuration space. The formal definition of this utility function for some trajectory t is:

$$\text{Utility}_e(t, M) = \begin{cases} 0 & \text{if } |t| < \tau_{min} \\ P(t = \text{free}|M)|t| & \text{if } \tau_{max} > |t| \geq \tau_{min} \\ \tau_{max} - |t| & \text{if } |t| \geq \tau_{max} \end{cases}$$

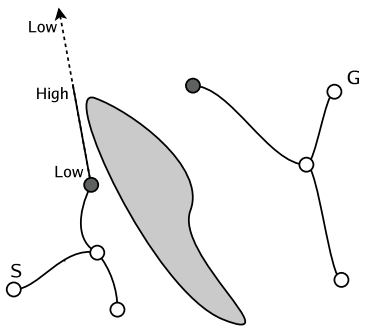


Figure 2: An illustration of the utility function for expansion distance. Expansion beyond the shadow of the obstacle does not increase the utility of the expansion.

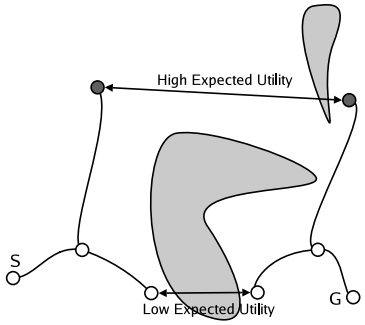


Figure 3: An illustration of the utility function for connection attempts. Expected utility of a connection is a function of the probability that the connection is obstructed.

3.3 Utility of connection attempt

Following each exploration, the connection step attempts to use the new information provided by exploratory expansion to discover a direct connection between the two trees. Even when these connection attempts fail, connecting growth often shortens the distance between the two trees and reduces subsequent exploration required to compute a path.

This is illustrated graphically in Figure 3. Though the bottom nodes in the two trees are quite close, connection is not attempted since the probability of a successful connection is low, due to the obstacle in between the nodes. The presence of this obstacle has been discovered in previous steps of the planner and the resulting information is stored in a configuration space model, which will be described in Section 3.4. This model also contains some indication that the direct connection between the two top nodes will fail, but this failure is less likely, given the information in the model. Hence, a connection attempt is made to connect the top two nodes. Though it does not succeed, it expands the coverage of the are between the two trees into parts of the configuration space likely to be free of obstructions, thus making progress towards connecting the two trees. The newly discovered information about obstacles in the configuration space will be added to the model and is available to guide future steps of the algorithm.

The utility of expanding configuration space coverage has previously been noted in the context of other single-query planners [3, 15]. We define connection utility as a linear function of distance and define the expected utility of some trajectory t as:

$$\text{Utility}_c(t, M) = P(t = \text{free}|M)|t|.$$

Connecting growth increases planner efficiency by minimizing the exploration required to solve a planning problem, while exploratory expansion adds new knowledge to the planner. Connecting growth is a greedy attempt to solve the problem with the information at hand.

Greedy search often leads to pathological behavior when the greedy path diverges significantly from the true solution path. In motion planning this causes planners to repeatedly attempt the expansion of trees into obstructed regions. This problem has been identified as a hazard for several single-query planners [1, 17]. Fortunately, a planner that is greedy with respect to *expected* utility generally avoids such pathologic behavior. The expected utility calculation considers not only the utility of the expansion but also the probability that the expansion is obstructed. Since obstructed expansions have no utility, pathological

expansions through obstructed configuration space regions are not chosen. If an obstructed expansion is chosen, information from that obstructed expansion is incorporated into the predictive model improve its predictions of future expansions.

3.4 Modeling configuration space

Estimating the expected utility of tree growth requires predicting the probability of each possible outcome of the exploration. This is the $P(e)$ term in the expected utility equation in Section 3.1. This prediction enables the selection of useful explorations prior to expending significant computational resources. These predictions are provided to the utility-guided motion planner by an approximate configuration space model, which is built incrementally from the information obtained during the explorations.

Modeling configuration space can be viewed as a machine learning problem [3]. The learner is trained on a number of configurations and their label (obstructed or free) and is asked to construct a model that can predict the label of unobserved configurations. For utility-guided RRT we use a simple nearest neighbor model. This model has several attractive features. Because it does not attempt to fit a global model to the data it is quite capable of capturing the complex shapes of an arbitrary configuration space. Additionally, inserting data into the model is a constant time operation, only querying the model requires any real computation. When using a nearest neighbor model, the key to the model's performance is the distance metric chosen. The distance metric is particular to different classes of configuration spaces. For some configuration spaces, such as the bugtrap worlds described in Section 4.1, a simple Euclidean distance metric is sufficient. For configuration spaces involving articulated robots we *reference point distance* a metric first suggested for motion planning by Leven, et al. [18]. The reference point metric is defined in terms of a number of reference points located on the body of the articulated robot. The actual location of these reference points depends on the kinematics of the robot. The Cartesian location of these points when the robot is in a particular configuration q are given by the functions $p_1(q) \ldots p_n(q)$. Given this set of functions P, we use the maximum reference point distance:

$$\mathrm{D}(q_1, q_2) = Max_{p \in P}\mathrm{EuclideanDistance}(p(q_1), p(q_2))$$

3.5 Utility-guided RRT planning

The estimates of utility presented previously are used to instantiate a utility-guided implementation of the general algorithm presented in Section 2. We choose to use the traditional Voronoi bias for selecting nodes and expansion direction. We anticipate in the future developing utility-guided implementations of these functions as well, however estimating the utility of a direction is still an open question. For the implementations of exploration length and connecting growth we use an incremental algorithm for utility-guided expansion (Figure 4). This algorithm explores along a trajectory through configuration space by extending the exploration in increments of length α until the expected utility of an extension is less than or equal to some threshold μ. In practice, incremental stepping improves the ability of the model to estimate the probability that an edge is obstructed or free. Long edges are more likely to be partially free and partially obstructed which makes the model's prediction significantly less accurate. While the expected utility is greater than the threshold, each incremental step is interpolated along the trajectory to ensure that it is free to some resolution ϵ. If the incremental step is free, it is added to the expansion of the tree. In the following, the edge between two configurations is denoted by $q \rightarrow q'$.

In the context of the Table 1, the utility-guided random-tree algorithm for single-query motion planning is summarized as:

Algorithm	Node Selection	Exp. Trajectory	Exp. Length	Connection
Util-RRT	Voronoi Bias	Voronoi Bias	Util. Exp.	Util. Exp.

UtilityExtend(T, q_s, q_e, M)
1: $q_i = q_s$
2: **do**
3: $q_i' = q_i + \alpha$
4: **if**(Utility$(q_i \rightarrow q_i') < \mu$)
5: break
6: if (Free$(q_i \rightarrow q_i')$)
7: AddToTree(T, q_i')
8: $q_i = q_i'$
9: **while** (q_i != q_e)

Figure 4: The utility-guided extension algorithm used for both exploration and connection

Note that this algorithm does not make any assumptions specific to motion planning. It can be applied to arbitrary state spaces. The notion of a state space model can incorporate hard constraints, such as forbidden regions, as well as soft constraints that express optimality criteria for a desired solution. Such soft constraints can be expressed as preferred regions of state space.

4 Experiments

The goal of bi-directional random-tree planning is to quickly identify a sequence of state transtitions, reaching from the intial to the final state. In the context of motion planning, this corresponds to a collision free path connecting a pair of configurations. Depending on how state transitions are chosen, this path will satisfy the kinodynamic constraints of the robot. Through the generalization of the random-tree algorithm we have identified several components in this planning process that affect the efficiency and reliability of the planner that have previously been unexplored. We have also proposed utility-guided implementations of these functions. In the following we present planning experiments that compare the performance of the utility-guided random-tree planner with adaptive dynamic-domain RRT a state of the art planner.

4.1 Bugtrap experiments

Researchers [22, 10] have identified so-called "bugtraps" as a challenging configuration space feature for traditional RRT methods. Previously, researchers have only considered bugtraps in two-dimensional configuration spaces. Because challenging motion planning problems occur in higher dimensional configuration spaces, we define bugtraps in configuration spaces of arbitrary dimensionality. Our bugtrap consists of a single hyper-sphere obstacle; the wall of this hyper-sphere has some "thickness" associated with it. The hyper-sphere is pierced by a hyper-cylinder oriented along the x axis and extending from the origin to the edge of the hyper-sphere. If a configuration is within this walls of the hyper-sphere or hyper-cylinder it is considered obstructed, otherwise it is free. Two-dimensional examples of such a bugtrap are shown in Figure 7.

We performed planning experiments in bugtrap worlds with two, three, four and five dimensions. In each world, each planner was asked to compute a path between a random point inside the bugtrap to a random point outside the bugtrap. The length of time to compute this path was recorded. The average results for one hundred random path queries are shown graphically in Figure 6. They are also shown numerically in Table 2.

The results show that the utility-guided algorithm significantly reduces the average planning time required by the planner for all dimensions when compared to adaptive dynamic-domain RRT. The standard

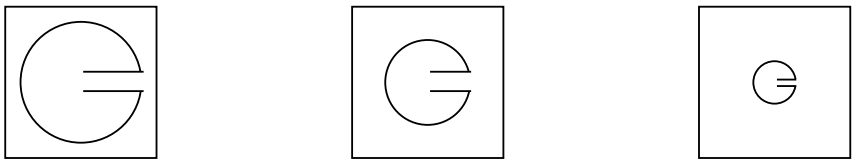


Figure 5: Large, medium, and small bugtrap; sizes are relative to the size of the considered configuration space.

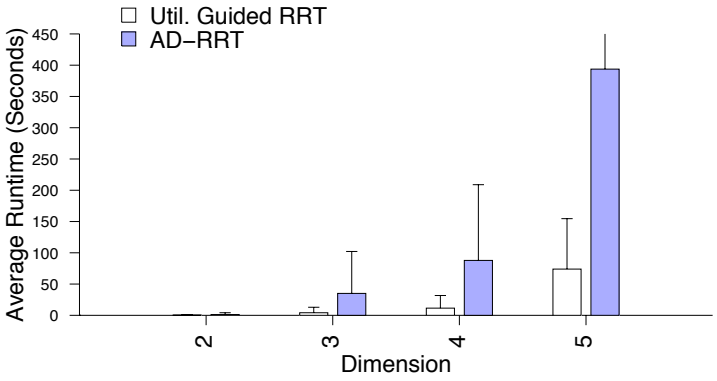


Figure 6: Results comparing planning algorithms for bugtrap worlds of varying dimensionality

DOF	A.D. RRT	Util. RRT
2	1.28 (2.88)	0.37 (0.53)
3	35.08 (67.04)	4.11 (8.81)
4	87.83 (121.08)	11.49 (20.20)
5	393.71 (555.47)	73.95 (80.64)

Table 2: The average runtime, with standard deviations in parenthesis, for adaptive domain RRT and utility guided RRT for bugtrap worlds of varying dimension

deviation of utility-guided runtimes is also smaller, indicating that the performance of the planner is more predictable as well.

In the previous experiments, the configuration was only slightly larger than the radius of the hyper-sphere defining the exterior of the bugtrap. We also investigated the performance of random-tree planners when the size of the configuration space relative to the bugtrap varied. This experiment evaluates performance variations that may occur when the amount of tree-refinement has to be balanced with configuration space exploration. To perform this evaluation, we reduced the size of the bugtrap relative to the size of configuration space. The two-dimensional environments for a large, medium, and small bugtrap are shown in Figure 5. Results showing the performance of the utility-guided RRT planner and teh adaptive dynamic-domain RRT for different sized bugtraps of varying dimensionality are shown in Figure 7. In these experiments, all planners were allowed to run for a maximum of five minutes. If the planner failed to provide a motion plan after five minutes, the particular run was recorded as a failure. The graphs in the left column in Figure 7 report the average runtime of successful motion plans. The graphs in the right column show the fraction of path planning attempts that succeeded for different sizes and dimensions. All graphs represent the average of one hundred path queries.

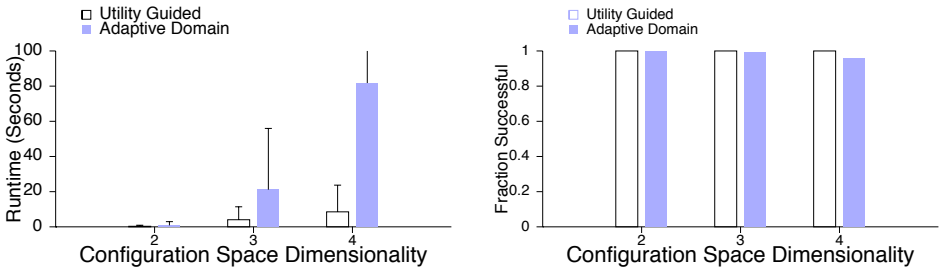
As before, utility-guided random-tree planning is more efficient and more reliable. Both planners suffer decreased performance as the relative size of the configuration space increases. However, this degradation is significantly more severe for adaptive dynamic-domain RRT. It is important to note that in some cases, the average runtime of both planners is equal. However in these cases, adaptive dynamic-domain RRT fails much more often. Thus the *expected* runtime of a successful query is significantly higher. The failure rate also increases with configuration space size for both planners, however it is also much less drastic for utility-guided RRT.

Jaillet et al. [10] report that the performance of adaptive dynamic-domain RRT performance is invariant to the size of the bugtrap. The experiments reported in [10] refer to the expansion of a single tree, rooted inside teh bugtrap. Our experiments indicate that for bi-directional tree-based planners the performance does vary with the relative size of the bugtrap.

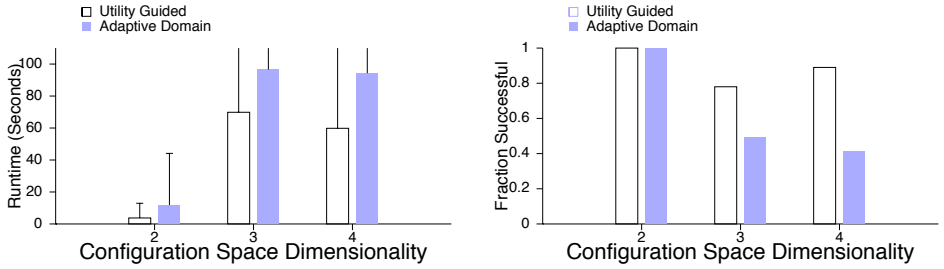
4.2 Real-world experiments

The experiments in bugtrap worlds demonstrate that utility-guided random-tree planning can improve performance in challenging configuration spaces. However, from a practical perspective it is important that the algorithm improve the performance of real-world planning as well. Thus we examined planner performance in the context of two related real-world tasks for a 14-dof humanoid torso (Figure 8). Both tasks examine the assembly of pipes in an obstructed environment. These experiments correspond with the overall goal of this humanoid platform performing assembly and service tasks on the exterior of the international space station. The motion planning problems consist of a start position with the arms extended on the outside of the box where the assembly will take place. The goal position is inside the box with the pipes oriented and ready for a lower-level force-controller to orchestrate the final assembly. To experiment with our motion planner on problems of varying difficulty, we performed experiments with and without a lid on the frame that serves as the obstacle. Images of the start and goal for both of these experiments are shown in Figure 8. For each of these experiments we performed 10 different requests for the same path query to planners seeded with different random seeds. The average runtime for each planner for each task are given in Table 4.2.

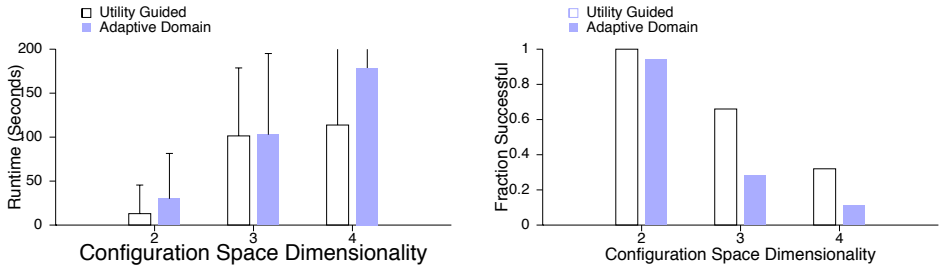
In both cases, the results show that utility-guided RRT is at least twice as fast as adaptive-domain RRT motion planning. Without the lid on the frame in which the assembly occurs, the motion planning problem is relatively simple. Even though both planners are fast at solving the problem, the two-times speed-up that utility-guided RRT provides is significant. At six seconds, the run time is fast enough to enable the robot to motion plan in dynamic worlds that change at a slow to moderate pace. Thus the development of the



(a) Large bugtrap



(b) Medium bugtrap



(c) Large bugtrap

Figure 7: Average planner runtime for different ratios of configuration to bugtrap size in configuration spaces of varying dimensionality.

World	Adaptive Domain RRT	Utility Guided RRT
Box w/o Lid	12.10 (6.50)	6.77 (4.00)
Box w/ Lid	66.67 (41.74)	24.62 (17.96)

Table 3: The average runtime, with standard deviations in parenthesis, for adaptive domain RRT and utility guided RRT for two related real-world environments.

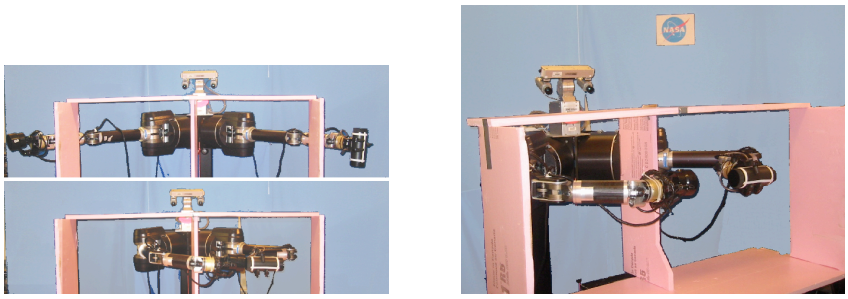


Figure 8: The humanoid robot Dexter in its start and goal configuration (left) and while enacting a successful motion plan (right).

utility-guided RRT algorithm is an important step toward dynamic motion planning. With the lid on the frame (the world shown in Figure 8) the problem is significantly harder. While it takes the adaptive-domain planner more than five times longer to solve the problem compared with the experiment without the lid, utility-guided RRT is only four times slower. This result are consistent with the slower growth in runtimes observed in the bug trap worlds. Additionally, the standard deviation of utility-guided RRT is significantly smaller than that of adaptive-domain RRT, indicating that the utility-guided planner is more robust as well. Combined with the results from the bugtrap worlds, these results demonstrate that utility-guided RRT's performance degrades more gracefully than adaptive-domain RRT as motion planning problems become increasingly more challenging.

5 Conclusions

State space exploration based on random trees has been applied successfully in a variety of problem domains. In the context of motion planning, random-tree planners have emerged as an an important class of sampling-based, single-query motion planners. In the preceding we have examined a comprehensive suite of existing random-tree planners and formulated a generalized algorithm that abstracts the characteristics shared by them all. This general algorithm identifies different functions in the planner that have an impact performance. Two of these functions—exploration length and connecting growth—have received little attention in previous research. We have proposed utility-guided implementations for these functions that result in an application of the utility-guided framework to random-tree motion planning. Empirical experiments with this utility-guided RRT planner demonstrate that it is capable of improving performance over existing state-of-the-art planners in challenging artificial and real-world environments. We believe the abstract algorithmic framework for random-tree planning presented herein enables advances in reasoning about random-tree planning and consequently further improvements in planner performance.

Acknowledgment

This work is supported in part by the National Science Foundation (NSF) under grants CNS-0454074, IIS-0545934, and MIT/NASA cooperative agreement NNJ05HB61A.

References

[1] R. Bohlin and L. E. Kavraki. Path planning using lazy PRM. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 521–528, San Francisco, USA, 2000.

[2] M. S. Branicky, M. M. Curtiss, J. Levine, and S. Morgan. RRTs for nonlinear, discrete, and hybrid planning and control. In *Proceedings of the IEEE Conference on Decision and Control*, Maui, USA, December 2003.

[3] B. Burns and O. Brock. Toward optimal configuration space sampling. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005.

[4] P. Cheng, Z. Shen, and S. M. LaValle. RRT-based trajectory design for autonomous automobiles and spacecraft. *Archives of Control Science*, 11(304):167–194, 2001.

[5] J. Cortés, T. Siméon, V. Ruiz de Angulo, D. Guieysse, M. Remaud-Siméon, and V. Tran. A path planning approach for computing large-amplitude motions of flexible molecules. In *Proceedings of the International Conference on Intelligent Systems for Molecular Biology (ISMB)*, Detroit, USA, 2005.

[6] E. Drumwright, M. Kallmann, and M. Matarić. Towards single-arm reaching for humanoids in dynamic environments. In *Proceedings of the IEEE International Conference on Humanoid Robots*, Santa Monica, USA, 2004.

[7] J. Esposito, J. Kim, and V. Kumar. Adaptive RRTs for validating hybrid robotic control systems. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics (WAFR)*, Utrecht/Zeist, The Netherlands, 2004.

[8] E. Ferré and J.-P. Laumond. An iterative diffusion algorithm for part disassembly. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3149–3154, New Orleans, USA, April 2004.

[9] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *International Journal of Computational Geometry and Applications*, 9(4):495–512, 1999.

[10] L. Jaillet, A. Yershova, S. M. LaVallen, and T. Siméon. Adaptive tuning of the sampling domain for dynamic-domain RRTs. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Edmonton, Canada, 2005.

[11] N. E. Jensen. Introdution to Bernoullian utility theory. *Swedish Journal of Economics*, 69(4):163–183, 1967.

[12] 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*, 12(4):566–580, 1996.

[13] J. Kim, J. M. Esposito, and V. Kumar. An RRT-based algorithm for testing and validating multi-robot controllers. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, 2005.

[14] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Motion planning for humanoid robots. In *Proceedings of the International Symposium of Robotics Research*, Siena, Italy, 2003.

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

[16] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Detroit, USA, 1999.

[17] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics (WAFR)*, pages 293–308, 2000.

[18] P. Leven and S. Hutchinson. Toward real-time path planning in changing environments. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2000.

[19] J. M. Phillips, N. Bedrossian, and L. E. Kavraki. Guided expansive spaces: A search strategy for motion- and cost-constrained state spaces. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3968–3973, New Orleans, USA, April 2004.

[20] S. Rodriguez, X. Tang, J.-M. Lien, and N. M. Amato. An obstacle-based rapidly-exploring random tree. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Orlando, USA, May 2006.

[21] D. R. Vallejo, C. Jones, and N. M. Amato. An adaptive framework for single shot motion planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1722–1727, Takamatsu, Japan, 2000.

[22] A. Yershova, L. Jaillet, T. Siméon, and S. M. LaValle. Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Barcelona, Spain, 2005.

[23] E. Yoshida, H. Kurokawa, A. Kamimura, K. Tomita, S. Kokaji, and S. Murata. Planning behaviors of a modular robot: an approach applying a randomized planner to coherent structure. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sendai, Japan, 2004.