Hierarchical Rejection Sampling for Informed Kinodynamic Planning in High-Dimensional Spaces

Tobias Kunz, Andrea Thomaz and Henrik Christensen

***Abstract*— We present hierarchical rejection sampling (HRS) to improve the efficiency of asymptotically optimal sampling-based planners for high-dimensional problems with differential constraints. Pruning nodes and rejecting samples that cannot improve the currently best solution have been shown to improve performance for certain problems. We show that in high-dimensional domains this improvement can be so large that rejecting samples becomes the bottleneck of the algorithm because almost all samples are rejected. This contradicts general wisdom that collision checking is always the bottleneck of sampling-based planners.**

Only samples in the informed subset of the state space can potentially improve the current solution. For systems without differential constraints the informed subset forms an ellipsoid, which can be parameterized and sampled directly. For systems with differential constraints the informed subset is more complicated and no such direct sampling methods exist. HRS improves the efficiency of finding samples within the informed subset without parameterizing it explicitly. Thus, it can also be applied to systems with differential constraints for which a steering method is available. In our experiments we demonstrate efficiency improvements of an RRT* planner of up to two orders of magnitude.

I. Introduction

Sampling-based planners have been successfully applied to difficult, high-dimensional motion planning problems. One popular example is the Rapidly Exploring Random Tree (RRT) [1]. While RRTs are probabilistically complete for problems considered here, they do not return an optimal solution. Karaman and Frazzoli [2] introduced asymptotically optimal sampling-based algorithms, including PRM*, RRG* and RRT*. Others have proposed RRT$^{\#}$ [3], FMT* [4], BIT* [5]. All of these algorithms optimally connect nodes within a given distance of each other. This distance shrinks as the number of nodes increases, avoiding large numbers of connection attempts and improving the efficiency of the algorithms.

In order to achieve a near-optimal solution, all these planners need to densely fill the state space with samples. However, in high-dimensional spaces this is very inefficient. In order to improve the efficiency and not fill the whole state space with samples, additional information can be used to fill only those parts of the state space that can improve the current solution. Following [6], we call this the *informed subset* of the state space. Assume we can calculate a lower bound on the optimal cost to move from the start through a sample and to the goal. If this lower bound is larger than the current solution, the sample cannot improve the current solution, is not part of the informed subset and can be ignored.

At the time this research was conducted the authors were with the Institute for Robotics and Intelligent Machines at the Georgia Institute of Technology, Atlanta, GA 30332, USA. Email: tobias@gatech.edu, athomaz@cc.gatech.edu, hic@cc.gatech.edu

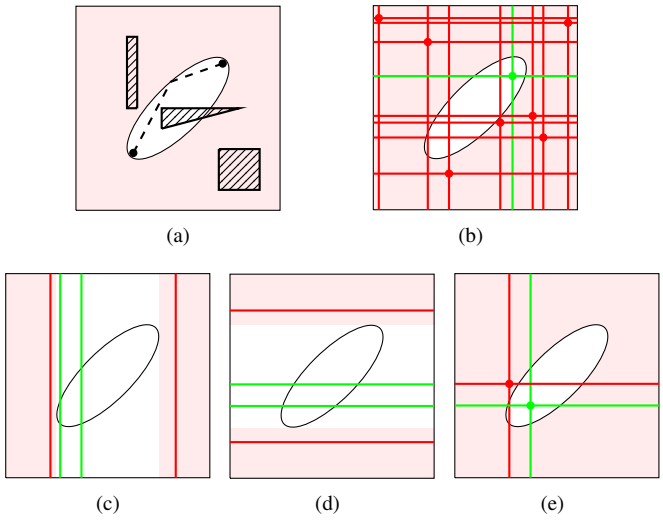

Fig. 1. Illustration of hierarchical rejection sampling using a simple example (2 dimensions, no differential constraints): (a) current solution trajectory and informed subset, (b) standard rejection sampling, (c) - (e) hierarchical rejection sampling (HRS).

For systems without differential constraints and the Euclidean distance as cost function, the informed subset is ellipsoidal [7]. This ellipsoidal subset can be parameterized and sampled directly [6]. For systems with differential constraints, however, the informed subset is not ellipsoidal but more complex. To date no methods to sample directly within that space exist. Instead, samples within the informed subset can be found using rejection sampling, i.e. sampling the full state space and rejecting those samples that cannot improve the current solution. However, especially in high-dimensional spaces, the informed subset might only be a tiny fraction of the whole state space. In this case most samples get rejected and rejection sampling is very slow. In our experiments up to 99.99% of all samples get rejected. This can lead to a sampling-based planner spending most of its time rejection sampling.

To improve the efficiency of informed sampling without the need to explicitly parameterize the informed subset, we introduce *hierarchical rejection sampling (HRS)*. HRS hierarchically combines partial samples into larger samples, making accept/reject decisions at every level. When a partial sample is rejected, only the partial information needs to be resampled.

We demonstrate the benefit of HRS by using it in combi-

nation with an RRT* planner. We plan acceleration-limited trajectories for robot arms with up to 7 DOFs. The RRT* planner plans in the 14-dimensional state space consisting of joint positions and velocities and uses a steering method. HRS speeds up the convergence of the RRT* planner by up to two orders of magnitude. The largest efficiency improvements are observed for close-to-optimal solutions.

HRS is primarily intended for systems with an available steering method, which can efficiently and optimally solve the boundary-value problem and, thus, connect any two states in the absence of obstacles. Such steering methods are for example available for double integrators, Dubins cars, Reeds-Shepp cars and linear-quadratic problems. This steering method can be used to calculate a lower bound on cost and, thus, to reject samples. We also assume that it is possible to apply the steering method to partial state samples to obtain a lower bound on cost. We expect the latter to be a weak assumption. We are not aware of any steering method that cannot not be applied to partial samples. However, the tightness of the obtained bound and, thus, the effectiveness of HRS might vary for different systems.

II. Related Work

A. Informed Graph Pruning

Informed graph pruning [8] is closely related to informed sampling. But instead of rejecting samples or areas of the state space, it rejects nodes in the graph. Thus, it cannot reject samples until they have been added to the graph. In contrast, informed sampling can reject samples before adding them to the graph. Informed graph pruning uses the graph as additional information. Instead of using a lower bound on the optimal cost-to-come, it uses the cost-to-come along the graph. This is advantageous because the graph is more informed as it considers obstacles. However, the cost along the graph gives an upper bound on the optimal cost-to-come instead of a lower bound. Thus, there is a chance that pruned nodes could have contributed to the optimal solution once the graph becomes closer to optimal.

The samples rejected by informed sampling are a subset of those rejected by informed graph pruning. Therefore, hierarchical rejection sampling can be combined with graph pruning. The benefit of hierarchical rejection sampling is actually larger when also employing graph pruning. This is because graph pruning reduces the number of collision checks, making it more likely that rejection sampling instead of collision checking becomes the bottleneck of the algorithm. In out experiments in Sec. VIII we evaluate the performance gain of hierarchical rejection sampling with and without employing informed graph pruning.

B. Biased Sampling

There has been a lot of work on improving the efficiency of sampling-based planners through adaption of the sampling distribution. The distribution has been biased toward the current solution trajectory [9, 10] or areas of the state space with a high difficulty, because of low manipulability [11] or narrow passages [12, 13].

Unlike informed sampling, sample biasing still accepts samples that cannot improve the solution. These sample schemes might lead to better or worse performance. They include a parameter to control the strength of the bias, which affects performance and has to be tuned. Informed sampling in contrast is parameter-free. It might not always lead to better performance, but it never causes significantly worse performance. Biased sampling schemes that uniformly sample the state space and then reject certain samples [9–11] can be combined with hierarchical rejection sampling. Sampling schemes that use a completely different method to generate samples [12, 13] cannot.

III. Background: Informed Sampling

We want to find the optimal trajectory from the start state $\boldsymbol{x}_{\text{start}}$ to the goal state $\boldsymbol{x}_{\text{goal}}$ according to some given cost function. The trajectory must completely lie in the obstacle-free space X_{free} and potentially satisfy differential constraints of the form $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ with $\boldsymbol{u} \in U$. Note that for a more concise presentation we assume a single goal state. The approach can easily be extended to a finite set of goal states. In our experiments we actually have multiple goal states.

Let c_{best} be the cost of the current solution or infinite if no solution has been found yet. Informed sampling rejects samples that cannot improve c_{best}.

Let $c^*(\boldsymbol{x}_1, \boldsymbol{x}_2)$ be the cost of the optimal trajectory from $\boldsymbol{x}_1$ to $\boldsymbol{x}_2$. A sample can only improve the current solution if the cost of the optimal solution trajectory through $\boldsymbol{x}$ is less than c_{best}, i. e.

$$c^*(\boldsymbol{x}_{\text{start}}, \boldsymbol{x}) + c^*(\boldsymbol{x}, \boldsymbol{x}_{\text{goal}}) < c_{\text{best}} \tag{1}$$

$c^*(\boldsymbol{x}_1, \boldsymbol{x}_2)$ is generally unknown. Instead we use a heuristic estimate $c(\boldsymbol{x}_1, \boldsymbol{x}_2)$. c is called admissible if

$$\forall \boldsymbol{x}_1, \boldsymbol{x}_2 \in X : c(\boldsymbol{x}_1, \boldsymbol{x}_2) \leq c^*(\boldsymbol{x}_1, \boldsymbol{x}_2) \tag{2}$$

Informed sampling uses such an admissible heuristic estimate to restrict samples to the subset of the state space that can potentially improve the current solution. Following [6], we call this the *informed subset* of the state space. The set includes all states $\boldsymbol{x}$ satisfying

$$c(\boldsymbol{x}_{\text{start}}, \boldsymbol{x}) + c(\boldsymbol{x}, \boldsymbol{x}_{\text{goal}}) < c_{\text{best}} \tag{3}$$

Because of Eq. 2, Eq. 1 implies Eq. 3, i.e. the informed subset includes all states that can improve the current solution.

If a steering method is available, the cost of the trajectory returned by the steering method can be used as the heuristic cost estimate. The cost returned by the steering method is a lower bound on the optimal cost, since the steering method returns the optimal trajectory for the relaxed problem without obstacles.

IV. A Simple Example

Fig. 1 uses a very simple example to demonstrate hierarchical rejection sampling and why it leads to more efficient sampling. The example uses a two-dimensional problem without differential constraints. Note that we use this simple example only to visualize hierarchical rejection sampling. Hierarchical rejection sampling is not needed for this problem, because the

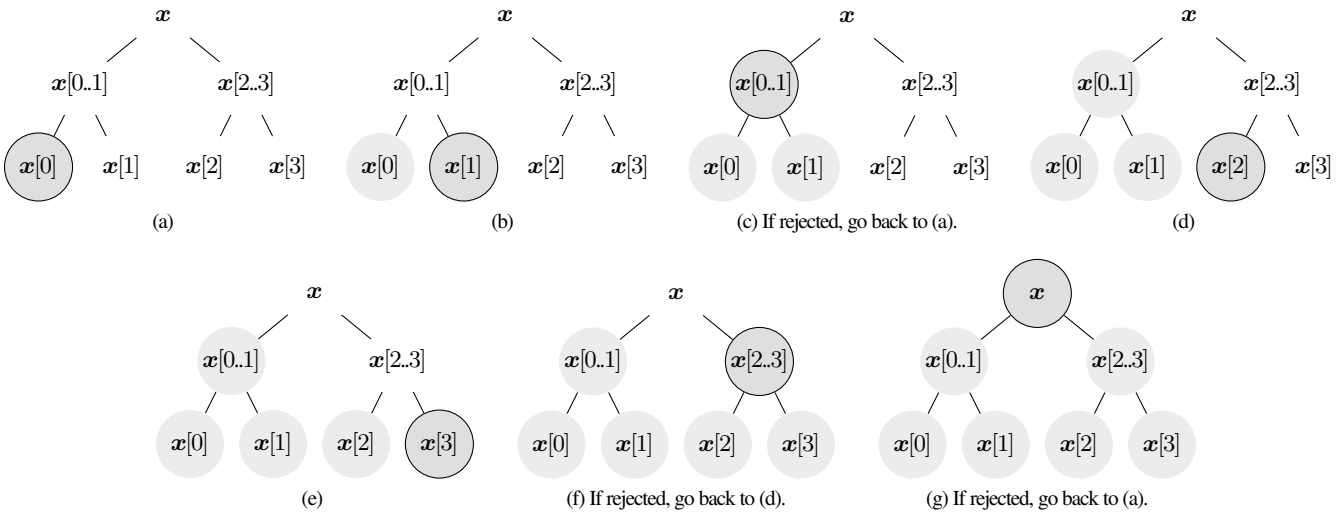

Fig. 2. Hierarchical rejection sampling: The sample gets built bottom-up along a tree structure. Whenever a node rejects part of a state, only its subtree gets resampled. Parts that have already been sampled are shown in gray. The active node is circled.

informed subset can be represented explicitly and sampled directly.

Fig. 1(a) shows a planning problem with start and goal states and obstacles. It also shows the current, suboptimal solution trajectory. The white ellipse marks the informed subset, in which a sample has to lie in order to be able to improve the solution trajectory. The informed subset covers 1/8 of the state space. The axis-aligned bounding box around it is a square and covers 1/4 of the state space.

Fig. 1(b) shows samples produced by standard rejection sampling. Since the informed subset covers 1/8 of the whole state space, we need on average 8 samples to find one valid sample (shown in green). Since the state space is 2-dimensional, these 8 samples consist of 16 scalar samples (shown as lines).

Fig. 1(c)-(e) visualize hierarchical rejection sampling. In Fig. 1(c) only the horizontal dimension is considered. There are 4 partial, scalar samples shown. Even though only partial samples are considered, 2 out of 4 samples can be rejected. This is because no matter what the value of the vertical dimension is, those partial samples can never result in a full sample that lies within the informed subset. Fig. 1(c) shows the same for the vertical dimension with 2 out of 4 samples being rejected. Fig. 1(e) combines the 4 partial, scalar samples that were accepted in Fig. 1(c) and 1(d) into 2 full samples. Only one of the 2 full samples lies within the informed subset. In Fig. 1(c) and 1(d) a total of 8 scalar samples are generated.

In this simple example hierarchical rejection sampling generates half as many scalar samples as standard rejection sampling on average. Admittedly, this is not a large improvement. However, the improvement grows larger when increasing the number of dimensions.

To illustrate the effect of a high-dimensional search space imagine that, like in the 2-dimensional example above, the current solution cost results in 50 % of all scalar samples being rejected. Now, imagine the problem is 7-dimensional. This results in a probability of $\left(\frac{1}{2}\right)^7$ or less than 1 % that a complete 7-dimensional sample is accepted. That means, on average, we need to generate more than 100 7-dimensional samples to get one sample that can improve the current solution. That is more than 700 scalar samples. Hierarchical rejection sampling, in contrast, requires only 2 scalar samples on average per dimension. That is a total of 14 scalar samples, much less than 700. In reality, HRS does not only make accept/reject decisions for partial scalar and complete 7-dimensional samples, but also for partial 2-, 3- or 4-dimensional samples. This further increases the benefit of HRS.

V. Algorithm

Fig. 2 visualizes the hierarchical rejection sampling algorithm using a 4-dimensional problem. At every node reject/accept decisions are made based on partial state information. $\boldsymbol{x}[i]$ represents the ith element of the state vector and $\boldsymbol{x}[i..j]$ the segment from element i to j. Leaf nodes sample a single dimension until a scalar sample is accepted. Interior nodes combine two partial samples resulting in a larger partial sample consisting of elements i to j. Interior nodes only consider elements i - j for calculating a lower bound on the cost of a trajectory through a sample and for making an accept/reject decision. If the sample is rejected, the node's whole subtree gets resampled.

Algorithm 1 shows a recursive implementation of hierarchical rejection sampling. The parameters i and j specify the currently considered segment $\boldsymbol{x}[i..j]$ of the state vector $\boldsymbol{x}$. The start state $\boldsymbol{x}_{\text{start}}$, goal state $\boldsymbol{x}_{\text{goal}}$ and the cost c_{best} of the best solution trajectory found so far are provided to the HRS algorithm. The algorithm returns $\boldsymbol{x}$, c_{start} and c_{goal}. $\boldsymbol{x}$ is the state sample that is being generated. In a simple implementation c_{start} and c_{goal} are lower bounds on the cost to move from the start state to the sample and from the sample to the goal state respectively. In a more efficient implementation c_{start} and c_{goal} might be more complex data

Algorithm 1: HRS($i, j, \boldsymbol{x}$)

```
1  if i = j then
2      repeat
3          x[i] ← SampleLeaf(i);
4          c_start ← CalculateLeaf(x_start, x, i);
5          c_goal ← CalculateLeaf(x, x_goal, i);
6          n[2i] ← n[2i] + 1;
7      until Cost(c_start) + Cost(c_goal) < c_best;
8  else
9      m ← ⌊(i+j)/2⌋;
10     repeat
11         (x, c_start, c_goal) ← HRS(i, m, x);
12         (x, c'_start, c'_goal) ← HRS(m+1, j, x);
13         c_start ← Combine(x_start, x, i, m, j, c_start, c'_start);
14         c_goal ← Combine(x, x_goal, i, m, j, c_goal, c'_goal);
15         n[2m+1] ← n[2m+1] + 1;
16     until Cost(c_start) + Cost(c_goal) < c_best;
17 return (x, c_start, c_goal);
```

structures containing additional cached results of intermediate calculations.

Lines 9 - 16 handle interior nodes, while lines 2 - 7 handle leaf nodes. Interior nodes divide the considered state vector segment in two parts. Line 9 calculates the split point. Lines 11 and 12 recursively call the algorithm to generate two partial samples. Using only the partial information available from dimensions i - j, lines 13 and 14 calculate a lower bound on the costs to move from the start to the sample and from the sample to the goal. If the sum of the lower bounds is larger than c_{best}, the sample is rejected and the process is repeated. The process for leaf nodes is similar. But instead of recursively sampling two partial samples, it just samples a scalar in line 3.

The algorithm shown here also updates a vector $\boldsymbol{n}$ of a size equal to the number of tree nodes in the sampling hierarchy: $2d-1$. Each element of $\boldsymbol{n}$ is associated with one node in the sampling hierarchy and stores the number of samples drawn by that node from its children. We call these the numbers of *explicit samples*. The numbers of explicit samples are updated in lines 6 and 15. These two lines are only necessary if one wants to keep track of the sample density and can otherwise be omitted. For more details on the use of $\boldsymbol{n}$ see Sec. VII.

VI. Example Implementations

In this section we provide two example implementations of hierarchical rejection sampling by providing implementations of the primitive procedures used by HRS in Algorithm 1.

A. Geometric

Algorithms 2, 3 and 4 provide an example implementation of HRS for geometric problems with a Euclidean cost function. Here c_{start} and c_{goal} are scalars. To reduce the number of computations, c_{start} and c_{goal} are not the costs but the squared costs. However, HRS still incurs a small overhead calculating the Euclidean distance since the square root has to be calculated multiple times: once for each partial sample.

Algorithm 2: CalculateLeaf_Geometric($\boldsymbol{x}_1, \boldsymbol{x}_2, i$)

```
1 return (x_1[i] − x_2[i])^2;
```

Algorithm 3: Combine_Geometric($\boldsymbol{x}_1, \boldsymbol{x}_2, i, m, j, c, c'$)

```
1 return c + c';
```

Algorithm 4: Cost_Geometric(c)

```
1 return √c;
```

B. Double Integrators Minimum Time

Algorithms 5, 6 and 7 provide an example implementation for the double-integrators minimum-time (DIMT) problem [14]. The system is a set of double integrators with the state consisting of joint positions and velocities and the control input of accelerations. Velocities and accelerations are bounded. Cost is defined as the minimum time required to move between two states.

For a simpler presentation we so far assumed leaf nodes to sample a scalar. However, the algorithm can easily be adapted to leaf nodes sampling a higher-dimensional sample. In our implementation of HRS for the DIMT problem leaf nodes sample both, position and velocity of a single joint. Also, i and j do not refer to a range of vector elements but a range of joints, each represented by two elements of the state vector.

Here, c_{start} and c_{goal} each represent a set of feasible times to move between two states. Algorithm 5 calculates the set of all feasible times at which a single double integrator can reach $\boldsymbol{x}_2$. We represent this set as a minimum time plus optionally an infeasible time interval. Algorithm 6 combines two partial samples by calculating the intersection of the two feasible sets. HRS does not cause a lot of overhead here, because even for normal sampling we would need to calculate the minimum time and infeasible time intervals for each individual joint and then find the minimum time feasible for all joints. For more information on the DIMT problem and how to calculate the minimum times and infeasible intervals see [14].

Algorithm 5: CalculateLeaf_DIMT($\boldsymbol{x}_1, \boldsymbol{x}_2, i$)

```
1 t_min ← MinimumTime(x_1[i], x_2[i]);
2 T_infeasible ← InfeasibleInterval(x_1[i], x_2[i]);
3 return {t : t ≥ t_min} \ T_infeasible;
```

Algorithm 6: Combine_DIMT($\boldsymbol{x}_1, \boldsymbol{x}_2, i, m, j, c, c'$)

```
1 return c ∩ c';
```

Algorithm 7: Cost_DIMT(c)

```
1 return min c;
```

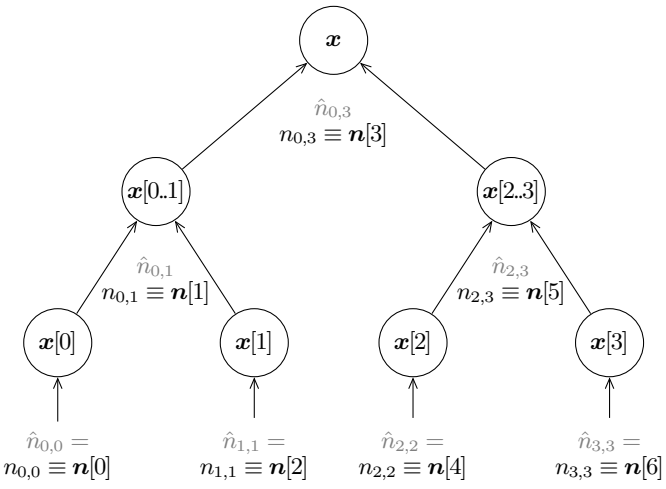

Fig. 3. Visualization of numbers of explicit (n) and implicit ($\hat{n}$) samples for every node.

VII. Implicit Samples

This section introduces *implicit samples* as a concept to measure sample density inside the informed subset. Unfortunately we cannot measure sample density inside the informed subset directly because we do not know its volume. Instead, we estimate the number of samples that would have been necessary to cover the full state space with the same sample density as the informed subset. We call this the number of implicit samples.

The sample density is necessary when using certain algorithms, e.g. the standard RRT*, which uses the sample density to control the connection threshold. We chose not to shrink the connection threshold in our experiments and thus do not need this information. Nonetheless, this section makes clear that HRS can be used in combination with such algorithms. In Sec. VIII-C we use the number of implicit samples to explain why HRS finds better solutions faster.

During the sampling process we collect information that allows us to calculate the number of implicit samples generated since the last time the informed subset changed, i.e. the solution cost decreased. The total number of implicit samples is the sum of the implicit samples generated during each phase of constant solution cost. The rest of this section only considers the number of implicit samples since the last solution cost decrease and thus assumes the informed subset does not change.

For each node in the sampling hierarchy that samples elements i to j, we store the number $n_{i,j}$ of *explicit samples* it has generated. This means the number of times it has queried its two children for a sample and made an accept/reject decision. This is also the number of times its two children have accepted a sample. Note that both children accept the same number of samples, because they both get queried the same number of times for a sample. This is visualized in Fig. 3.

We can also assign a number $\hat{n}_{i,j}$ of *implicit samples* to each node. This is the number of samples that would have been required to achieve the same number of accepted partial samples $\boldsymbol{x}[i..j]$ with standard rejection sampling. The number of implicit samples mentioned in the first paragraph of this section is the number of implicit samples of the root node.

The numbers $\hat{n}_{i,j}$ do not need to be stored since they can be calculated from the numbers of explicit samples $n_{i,j}$ of all nodes. The number $\hat{n}_{i,j}$ of implicit samples of a node can be calculated recursively from the number $n_{i,j}$ of explicit samples and the numbers $\hat{n}_{i,m}$ and $\hat{n}_{m+1,\,j}$ of implicit samples of its child nodes.

For both child nodes we can calculate the probability that one accepts an implicit sample. These probabilities $p_{i,m}$ and $p_{m+1,\,j}$ are estimated from the number of accepted and implicit samples seen so far as:

$$p_{i,m} = \frac{n_{i,j}}{\hat{n}_{i,m}} \quad \text{and} \quad p_{m+1,\,j} = \frac{n_{i,j}}{\hat{n}_{m+1,\,j}} \tag{4}$$

where $m = \lfloor \frac{i+j}{2} \rfloor$. The number $\hat{n}_{i,j}$ of implicit samples of the current node is then given as the number of times this node queried its child nodes for a sample divided by the probability that an implicit sample is accepted by both child nodes:

$$\hat{n}_{i,j} = \frac{n_{i,j}}{p_{i,m} \cdot p_{m+1,\,j}} \tag{5}$$

Combining Eq. 4 and 5 we obtain

$$\hat{n}_{i,j} = \frac{\hat{n}_{i,m} \cdot \hat{n}_{m+1,\,j}}{n_{i,j}} \tag{6}$$

For leaf nodes with $i = j$ the number of implicit samples equals the number of explicit ones.

$$\hat{n}_{i,i} = n_{i,i} \tag{7}$$

Combining the recursive formula with the base case for leaf nodes, we can write the number of implicit samples of the root node in closed form.

$$\hat{n}_{0,d-1} = \frac{\prod\limits_{\boldsymbol{x}[i] \text{ is leaf node}} n_{i,i}}{\prod\limits_{\boldsymbol{x}[i..j] \text{ is interior node}} n_{i,j}} \tag{8}$$

To store the values of $n_{i,j}$ we can flatten the two indices i and j into a single one: the index of the node is the in-order traversal of the tree, i.e.

$$\text{For } i \neq j: \quad n_{i,j} \equiv \boldsymbol{n}[2m+1] \tag{9}$$

$$\text{For } i = j: \quad n_{i,j} \equiv \boldsymbol{n}[2i] \tag{10}$$

with $m = \lfloor \frac{i+j}{2} \rfloor$. Eq. 8 giving the total number of implicit samples can then be written as

$$\hat{n}_{0,d-1} = \frac{\prod\limits_{k=0}^{d-1} \boldsymbol{n}[2k]}{\prod\limits_{k=0}^{d-2} \boldsymbol{n}[2k+1]} \tag{11}$$

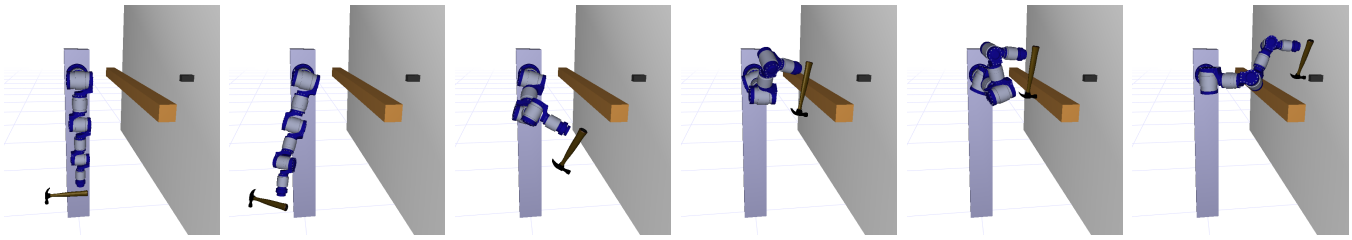

Fig. 4. Hammering: Solution trajectory after 60 s of computation time

(a) 0.0 s (b) 1.4 s (c) 2.8 s (d) 4.2 s (e) 4.4 s

Fig. 5. Batting: Solution trajectory after 60 s of computation time

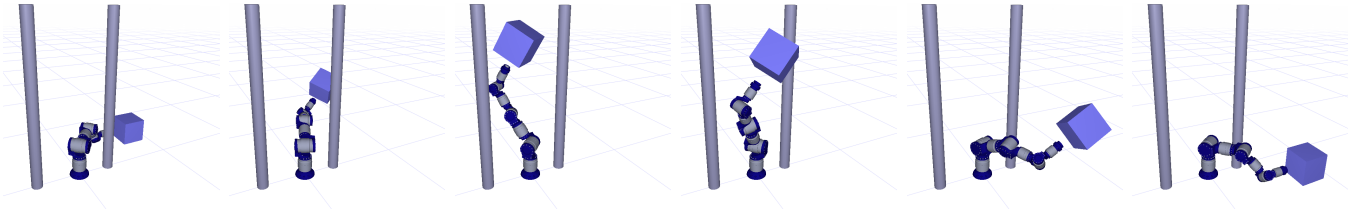

Fig. 6. Pick and place: Solution trajectory after 60 s of computation time

VIII. Experiments

A. RRT* Algorithm

To evaluate HRS we apply it to an RRT* algorithm [2] with a steering method. We deviate from the standard RRT* in two ways. First, we grow the tree all the way to the sampled state instead of making a small step toward it, since this is more efficient in high-dimensional spaces. Second, we do not shrink the connection threshold for rewiring as the number of tree nodes increases. Instead, we use an infinite connection threshold and attempt to rewire a new node to and from all nodes in the tree.

We use an infinite connection threshold because the standard geometric connection threshold [2] only guarantees asymptotic optimality for systems without differential constraints. An infinite connection threshold might be less efficient, but guarantees asymptotic optimality. Using a more efficient connection threshold would only increase the positive effect of HRS. Karaman and Frazzoli also presented a kinodynamic RRT* [15]. However, their paper leaves some details unexplained. For example it is unclear how to choose the connection threshold such that the neighborhood contains a ball of a given size. Other authors using the kinodynamic RRT* seem to have similar problems as their papers deviate from [15] without mentioning or explaining it. [16] and [17] use the geometric connection threshold for systems with differential constraints. This results in asymptotic optimality not being guaranteed. Similar to this paper [18] uses a non-shrinking infinite or constant finite connection threshold.

B. Example Problems

We evaluate our planner on three example problems, which are described below and shown in Fig. Fig. 4 - 6. All problems use a robot arm where each joint is modeled as a double integrator as described in Sec VI-B. HRS relies on an ordering of the joints. Since we do not expect the ordering of the different joints to have a major effect, we just use the physical ordering of the joints within the arm. Table I shows parameters of the problems. All problems assume goal states to be given in joint space. We automatically generate a set of joint-space goals from a given workspace goal. This conversion is not part of the algorithms and evaluation presented in this paper.

1) Problem 1: Hammering: A simulated 7-DOF robot arm is given the task to hit a nail at a given velocity while avoiding an obstacle. Because of the required non-zero goal velocity, the problem cannot be solved by a geometric planner. The state space, consisting of joint positions and velocities, is 14-dimensional. To the best of our knowledge the highest-dimensional space a kinodynamic RRT* has been applied to before is 10-dimensional [18].

TABLE I

Parameters of the example problems

task	DOF / dimensions	goal states	end-eff. goal vel.	vel. limit	accel. limit
hammering	7 / 14	100	0.6 m/s	90 °/s	45 °/s^2
batting	5 / 10	50	5.0 m/s	90 °/s	60 °/s^2
pick and place	7 / 14	2	0.0 m/s	90 °/s	45 °/s^2

2) Problem 2: Batting: Unlike the first problem, the second example problem involves a real robot. We use a KUKA KR 210 robot arm. Again, the goal is to hit an object, a ball in this case, at a velocity of 5 m/s. A stack of boxes is placed between the robot and the ball such that the shortest trajectory is in collision and the robot needs to plan around the obstacle. While the robot has 6 DOF, we only make use of 5 of them. Thus, the planning problem has 10 dimensions.

3) Problem 3: Pick and Place: While the previous two example problems involve non-zero goal velocities, in this problem, both, start and goal velocity are zero, which is typical for pick-and-place operations. The robot has to move a box through two vertical obstacles. There are two goal states, which are almost identical. Only the first joint differs by 360 degrees.

C. Results

Below we evaluate the convergence of the RRT* algorithm when using HRS in comparison to using uninformed sampling and standard rejection sampling. Uninformed sampling does not reject any samples. Standard rejection sampling samples a complete state and then makes a accept/reject decision. For a fair comparison we implemented standard rejection sampling as efficiently as possible. Even standard rejection sampling might make reject decisions early before having finished calculating the lower bound on cost. To do that it passes the current solution cost to the steering method to allow the steering method to abort the calculations as soon as the current solution cost is exceeded. However, unlike HRS, standard rejection sampling always rejects and resamples the complete sample. We compare the performance of standard and hierarchical rejection sampling with and without additional graph pruning. All results shown below are averages over 100 runs on a single core of an Intel Xeon E5-1620 CPU (3.6 GHz, released 2012). The graphs also show standard deviations as vertical bars.

Fig. 7 shows the convergence of the RRT* algorithm for the three example problems. While informed standard rejection sampling improves over uninformed sampling, HRS improves efficiency even further, leading to the planner finding lower-cost solution trajectories faster. This is the case with or without graph pruning. For the hammering and pick-and-place problems the improvement is significant, while for the lower-dimensional batting problem the improvement is small.

To measure the efficiency improvement, in Table II we compare the average computation time required by different sampling strategies to achieve the same solution cost. The efficiency gain factor is the quotient of the time required by standard rejection sampling and by HRS. The improvement grows larger as the solution cost shrinks, since more and more samples are rejected and a larger and larger fraction of time is spent sampling. HRS improves efficiency by up to two orders of magnitude.

Table III gives additional insight into why HRS improves performance. The number of implicit samples is directly correlated to the solution cost. More implicit samples lead to the state space being filled more densely and thus to a better solution trajectory. While standard rejection sampling accepts fewer samples than uninformed sampling and thus can generate more implicit samples in the same amount of time, it creates a new bottleneck since the algorithm now spends most of its time sampling. HRS reduces the amount of time spent sampling and thus can create even more implicit samples in the same amount of time. This also results in more accepted samples and more nodes. The RRT* with HRS generates up to 35 billion implicit samples in 60 seconds. I.e. the samples fill the informed subset as densely as if we had sampled 35 billion samples in the whole state space. After applying HRS, sampling is not the bottleneck anymore for the batting and pick-and-place problems. However, for the hammering problem it still is. Graph pruning reduces the time spent collision checking. Therefore, a larger fraction of the time is spent on sampling and the efficiency improvement of HRS is larger when combined with graph pruning.

TABLE II

AVERAGE COMPUTATION TIME REQUIRED TO REACH GIVEN SOLUTION COSTS

		without graph pruning				with graph pruning		
task	solution cost	uninformed sampling	rejection sampling	HRS	eff. gain factor	rejection sampling	HRS	eff. gain factor
hammering	7.0 s	22.4 s	0.9 s	1.0 s	0.9	0.5 s	0.5 s	0.9
	6.5 s	176.9 s	1.3 s	1.5 s	0.9	0.7 s	0.7 s	1.0
	6.0 s		2.6 s	2.3 s	1.1	1.6 s	1.2 s	1.4
	5.5 s		23.3 s	5.3 s	4.4	21.4 s	3.5 s	6.0
	5.2 s		216.2 s	12.5 s	17.3	206.3 s	9.8 s	21.0
	5.0 s		1793.2 s	40.4 s	44.4	1797.1 s	32.7 s	**55.0**
batting	5.5 s	6.06 s	0.45 s	0.46 s	1.0	0.30 s	0.32 s	1.0
	5.0 s	77.48 s	0.58 s	0.59 s	1.0	0.38 s	0.38 s	1.0
	4.5 s		1.18 s	0.93 s	1.3	0.77 s	0.57 s	1.4
	4.26 s		36.18 s	7.73 s	4.7	31.51 s	5.05 s	**6.2**
pick and place	8.0 s	95.3 s	9.6 s	8.3 s	1.2	3.1 s	2.3 s	1.3
	7.0 s		16.1 s	14.7 s	1.1	3.9 s	3.2 s	1.2
	6.0 s		27.1 s	21.9 s	1.2	7.0 s	4.0 s	1.7
	5.5 s		62.5 s	34.4 s	1.8	33.3 s	4.8 s	6.9
	5.2 s		327.4 s	75.9 s	4.3	284.5 s	6.8 s	42.1
	5.0 s		3059.0 s	354.4 s	8.6	2828.8 s	17.5 s	**161.4**

IX. Conclusion

We showed that applying standard rejection sampling to an asymptotically optimal sampling-based planner can sometimes cause rejection sampling to become the bottleneck of the algorithm. We presented hierarchical rejection sampling (HRS) to improve the efficiency of rejection sampling. Unlike existing work, HRS does not require an explicit representation of the informed subset and can thus also be applied to systems with differential constraints. Using a system of double integrators we demonstrated an efficiency improvement by HRS of up to two orders of magnitude. Future work includes evaluating HRS on more systems and a formal analysis under what conditions HRS provides a significant efficiency improvement.

Acknowledgments

This work was supported in part by ONR grant N00014-14-1-0120.

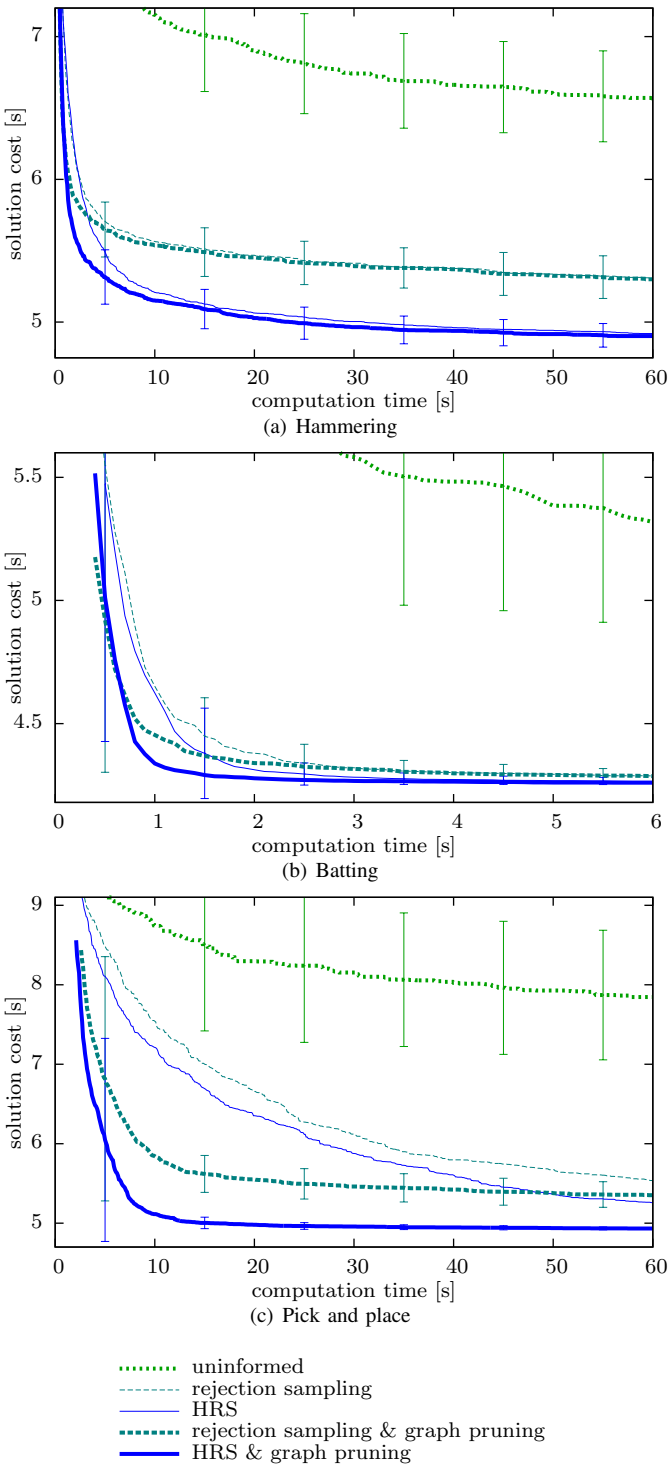

Fig. 7. Convergence of the RRT* for different sampling strategies (lower is better)

TABLE III

AVERAGE NUMBER OF SAMPLES AND NODES GENERATED WITHIN 60 S

task	graph pruning	sampling	(implicit) samples	accepted samples	nodes	time spent sampling
hammering		uninformed	7,338	7,338	3,847	0.0 %
		rejection	11,640,000	406	363	95.0 %
		HRS	1,226,000,000	998	849	88.5 %
	X	rejection	12,100,000	408	156	98.0 %
	X	HRS	1,491,000,000	1,014	344	95.7 %
batting		uninformed	15,183	15,183	1,616	0.0 %
		rejection	23,744,000	2,894	1,745	78.6 %
		HRS	105,640,000	11,294	7,355	8.2 %
	X	rejection	25,570,000	3,091	1,723	85.2 %
	X	HRS	143,770,000	15,212	9,597	11.0 %
pick and place		uninformed	26,145	26,145	1,556	0.0 %
		rejection	40,399,000	9,952	927	50.7 %
		HRS	1,184,700,000	15,364	1,497	0.3 %
	X	rejection	72,628,000	10,370	109	90.3 %
	X	HRS	35,741,000,000	87,062	4,887	3.9 %

REFERENCES

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

[2] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The Int. Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[3] O. Arslan and P. Tsiotras, "Use of relaxation methods in sampling-based algorithms for optimal motion planning," in *IEEE Int. Conf. on Robotics and Automation*, 2013.

[4] L. Janson and M. Pavone, "Fast marching trees: a fast marching sampling-based method for optimal motion planning in many dimensions," in *Int. Symposium on Robotics Research*, 2013.

[5] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "BIT*: Batch informed trees for optimal sampling-based planning via dynamic programming on implicit random geometric graphs," *CoRR*, vol. abs/1405.5848, 2014.

[6] J. Gammell, S. Srinivasa, and T. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2014.

[7] D. Ferguson and A. Stentz, "Anytime RRTs," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2006.

[8] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT*," in *IEEE Int. Conf. on Robotics and Automation*, 2011.

[9] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2011.

[10] J. Nasir, F. Islam, U. Malik, Y. Ayaz, O. Hasan, M. Khan, and M. S. Muhammad, "RRT*-SMART: A rapid convergence implementation of RRT*," *Int. Journal of Advanced Robotic Systems*, vol. 10, 2013.

[11] P. Leven and S. Hutchinson, "Using manipulability to bias sampling during the construction of probabilistic roadmaps," *IEEE Trans. on Robotics and Automation*, vol. 19, no. 6, pp. 1020–1026, 2003.

[12] H. Kurniawati and D. Hsu, "Workspace importance sampling for probabilistic roadmap planning," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2004.

[13] J. P. van den Berg and M. H. Overmars, "Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners," *The Int. Journal of Robotics Research*, vol. 24, no. 12, pp. 1055–1071, 2005.

[14] T. Kunz and M. Stilman, "Probabilistically complete kinodynamic planning for robot manipulators with acceleration limits," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2014.

[15] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *IEEE Conf. on Decision and Control*, 2010.

[16] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics," in *IEEE Int. Conf. on Robotics and Automation*, 2012.

[17] G. Goretkin, A. Perez, R. Platt, and G. Konidaris, "Optimal sampling-based planning for linear-quadratic kinodynamic systems," in *IEEE Int. Conf. on Robotics and Automation*, 2013.

[18] D. J. Webb and J. van den Berg, "Kinomdynamic RRT*: Optimal motion planning for systems with linear differential constraints," in *IEEE Int. Conf. on Robotics and Automation*, 2013.