Efficient Sampling-based Motion Planning with Asymptotic Near-Optimality Guarantees for Systems with Dynamics

Zakary Littlefield, Yanbo Li, Kostas E. Bekris

Abstract—Recent motion planners, such as RRT*, that achieve asymptotic optimality require a local planner, which connects two states with a trajectory. For systems with dynamics, the local planner corresponds to a two-point boundary value problem (BVP) solver, which is not always available. Furthermore, asymptotically optimal solutions tend to increase computational costs relative to alternatives, such as RRT, that focus on feasibility. This paper describes a sampling-based solution with the following desirable properties: a) it does not require a BVP solver but only uses a forward propagation model, b) it employs a single propagation per iteration similar to RRT, making it very efficient, c) it is asymptotically near-optimal, and d) it provides a sparse data structure for answering path queries, which further improves computational performance. Simulations on prototypical dynamical systems show the method is able to improve the quality of feasible solutions over time and that it is computationally efficient.

I. INTRODUCTION

Traditionally, sampling-based methods are practical solutions for complex, high-dimensional planning problems, which quickly provide solutions [1], [2]. Tree-based algorithms, such as the Rapidly-exploring Random Tree (RRT) [3], are also effective under dynamics as they only require a forward propagation model and do not need a local planner, unlike roadmap methods. A local planner connects two states with a trajectory and corresponds to a solution to a two-point boundary value problem (BVP), which is not always available.

While very efficient in terms of state-space exploration, the RRT converges to a suboptimal solution [4], [5]. Recent progress, however, led to the development of RRT* that provides asymptotic optimality [5]. While a major breakthrough, RRT* still relies on an underlying roadmap and requires a local planner. This limits the capability of providing path quality guarantees for interesting robots with dynamics.

Another issue relates to the computational efficiency of the corresponding methods. While asymptotically RRT* has the same computational overhead as RRT in practice it is more expensive as it tries to connect to multiple neighbors per iteration instead of once. Furthermore, existing methods include every state space sample in the corresponding data structure and is not obvious when to stop sampling, resulting in increased space requirements. It is desirable to have solutions that provide both path quality guarantees and a return a sparse data structure that can be queried efficiently.

With the above properties in mind, this paper describes two modifications to the RRT. A previously proposed modification [6], called here RRT with BestNearest, is shown to have beneficial path quality properties. It chooses a low cost node within a predefined radius of a random sample to expand from. This allows low cost paths to be propagated forward. The approach is efficient as it needs only a single propagation per iteration but employs a more complex query from the auxiliary nearest-neighbor data structure.

The other modification, RRT with Drain, applies pruning so that only the node connected with the best path to the root is maintained within a predefined radius. This allows the improvement of paths with good quality. This variant significantly reduces the computational cost of performing nearest neighbor queries - the asymptotically dominant factor in these methods - as it works with a small number of nodes, which quickly converges to an equilibrium.

The overall new algorithm, called SPARSE-RRT, combines these two modifications and provides a simple and computationally efficient way to provide continual trajectory improvement over time for systems with dynamics. Experimental results suggest that better quality solutions can be reached than an existing heuristic variant of RRT* for systems with dynamics [7] with the computational efficiency of RRT. Figure 1 provides a comparison of RRT with the discussed variants in the case of an inverted pendulum where no local planner is employed.

II. BACKGROUND

Sampling-based tree planners, such as RRT [8] and Expansive Spaces [9], can be seen as extensions of search-based, kinodynamic planning [10]. They aim to evenly explore a state space $X$ by expanding a tree where nodes are states and edges are trajectories. Given a good metric, which is not always easy to define, the RRT exhibits a Voronoi bias, where larger unexplored sections of $X$ have higher probability of being explored first. Some methods aim to decrease the dependence on the metric by reducing the rate of failed node expansions [11]. Others guide the tree using local reachability information [12] or linearizing locally the dynamics to compute a metric [13]. Recent work focuses on applying such planners to challenging problems, such as under-actuated systems, manipulation and grasping [12], [14], [15]. Bidirectional versions improve performance but require a local planner. For certain robots, such as systems with symmetries, it is possible to address this issue [16], [17]. Some variants of RRT have employed heuristics to improve path quality but do not argue optimality [6], [18].
While RRT is effective in returning a solution quickly, it converges to a sub-optimal solution [5], [4]. The recently proposed RRT$^*$ [5] provides asymptotic optimality. Anytime [19] and lazy [20] variants of RRT$^*$ have been proposed. There are also approaches that provide asymptotic near-optimality [21]. All of the above methods require a BVP solver. A variation of RRT$^*$ utilizes a “shooting” approach to improve solutions over time without a local planner [7]. When a propagation from node $\alpha$ to node $\beta$ gets to state $\beta'$ within a small distance of $\beta$, a tree connection between $\alpha$ and $\beta$ is made. This results in a re-propagation of the subtree of $\beta$ from $\beta'$ to maintain the validity of each subsequent state, which may result in node prunings if collisions occur. This method does not provably achieve asymptotic optimality but can behave well together with numerical methods for decreasing the gap between $\beta$ and $\beta'$ [7]. This paper employs only random propagation and compares the performance of the proposed approaches and this existing RRT$^*$ variant.

Recent work provides local planners for systems that have linear dynamics or dynamics that can be linearly approximated [22]. This allows the use of RRT$^*$ for a wider set of systems. The proposed algorithms are applicable beyond systems with linear dynamics but could also be combined with this existing approach to provide more efficient asymptotically optimal solvers for systems with linear dynamics. A conservative estimate of the reachable region of a system can be constructed given results from sub-Riemannian geometry and using the Lie algebra of the dynamical system [23]. This reachable region helps to define appropriate metrics under dynamics, and can also be used in conjunction with the algorithms described in this paper.

Selective pruning has been used in the past to help recover regions of the state space that RRT claims early in execution [24]. It is claimed that this approach improves path quality, but no justification is provided. Another work stores a volume at each node of a tree instead of states [25]. These volumes represent claimed free regions in the space being searched and help focus search toward unexplored regions. Inspiration for the current method is drawn from approaches that reduce the size of the planning structure to achieve sparse representations and provide near-optimality [21].

III. PROBLEM SETUP

Consider a robot whose motion is governed by the differential equations:

$$\dot{x} = f(x,u), \quad g(x, \dot{x}) \leq 0$$

where $f,g$ are smooth; $x \in X$ is a state and $u \in U$ is a control. The set of collision-free states is $X_f \subset X$. The robot is in an initial state $x_0$ and must reach a goal region $X_G$ defined by a ball of radius $\delta$ around a goal state $B(x_{goal}, \delta)$.

Definition 1: [Trajectory] A trajectory of duration $T$ is a function $\pi : [0, T] \rightarrow X_f$ of bounded variation corresponds to piecewise-constant controls $u \in U$ and satisfies Eq. 1.

The set I is the set of all trajectories in $X_f$.

Definition 2: [Kinodynamic Motion Planning] Given the tuple $(X_f, U, x_0, X_G)$, find a trajectory $\pi \in I$ that is a solution, i.e., $\pi(0) = x_0$ and $\pi(T) \in X_G$ and satisfies Eq. 1.

The cost $c(\pi)$ of a trajectory is the cost to move from $\pi(0)$ to $\pi(T)$. The cost $c(x)$ of a node $x$ on a tree structure is the cost of the trajectory from the root $x_0$ to $x$. This work considers an additive cost function, such as time to traverse a trajectory. This cost function is used to evaluate solution paths and what the algorithm is optimizing over.

It is not easy to define the cost from $x_0$ to $x_1$ without a trajectory connecting them. Sampling-based algorithms, however, depend on a function that evaluates the proximity of states. This is typically defined in the task space and will be denoted as the metric $m(x_0, x_1)$. The best value for $m(x_0, x_1)$ is the optimum cost-to-go from $x_0$ to $x_1$ in the absence of obstacles but this is not available without solving optimally a motion planning problem. The function $m(x_0, x_1)$ may even be a symmetric function, even though the cost function is typically not for systems with dynamics. Thus, it is important to provide methods that are robust in the discrepancy between the metric and the cost function.

The objective is to provide methods with this property:

Definition 3: [Asymptotic Near-Optimality] An algorithm $\text{ALG}$ is asymptotically near-optimal for a kinodynamic motion planning problem $(X_f, U, x_0, X_G)$ and a cost function $c : I \rightarrow \mathbb{R}_{\geq 0}$ that admit a robustly optimal solution with finite cost $c^*$, when the probability that $\text{ALG}$ will find a solution of cost $c < tc^*$ for some factor $t \geq 1$ converges to 1 as the number of iterations approaches infinity.

Robustly optimal solutions are the optimal paths that have a certain minimum clearance from obstacles.
IV. METHODOLOGY

This section provides two modifications to the RRT algorithm that improve the quality of paths returned without requiring a local planner and using a single propagation per iteration. The proposed solution, called SPARSE-RRT, outlined in Algorithm 1, provides the benefits of both modifications, called BestNearest and Drain.

Algorithm 1: SPARSE-RRT($X_f, X_0, G, N, \Delta_{\text{drain}}, \Delta_{\text{near}}$)

1. $V_{\text{active}} \leftarrow \{x_0\}$, $V_{\text{inactive}} \leftarrow \emptyset$, $E \leftarrow \emptyset$, $i \leftarrow 0$;
2. $V \leftarrow V_{\text{active}} \cup V_{\text{inactive}}$; $G = \{\emptyset, E\}$;
3. while $i < N$ do
   4. $x_{\text{rand}} \leftarrow \text{Sample}(X)$;
   5. $x_{\text{nearest}} \leftarrow \text{BestNearest}(V_{\text{active}}, x_{\text{rand}}, \Delta_{\text{near}})$;
   6. $x_{\text{new}} \leftarrow \text{Propagate}(x_{\text{nearest}}, U, X_f)$;
   7. $V_{\text{active}} \leftarrow V_{\text{active}} \cup \{x_{\text{new}}\}$;
   8. $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$;
   9. $\text{Drain}(\Delta_{\text{drain}}, x_{\text{new}})$;
   10. $i \leftarrow i + 1$;

SPARSE-RRT resembles RRT in that a random state is sampled, a node is found in its proximity, a forward propagation step is executed, and an edge is added. The effects of the new input parameters $\Delta_{\text{drain}}$ and $\Delta_{\text{near}}$ will be clarified below. Typically, $\Delta_{\text{drain}}$ is in the order or smaller than the goal radius $\delta$, while $\Delta_{\text{near}}$ is greater than $\Delta_{\text{drain}}$. Special cases of the algorithm arise for different values of $\Delta_{\text{drain}}$ and $\Delta_{\text{near}}$. When both are 0, SPARSE-RRT becomes equivalent to the basic RRT. When either of the values are zero, only one of the modifications is used.

BestNearest: This function checks additional criteria when compared to a simple nearest neighbor selection. This modification was originally proposed in [6] and this work will analyze the effects it has in returning solutions.

Algorithm 2: BestNearest($V_{\text{active}}, x_{\text{rand}}, \Delta_{\text{near}}$)

1. $x_{\text{near}} \leftarrow \text{Near}(V_{\text{active}}, x_{\text{rand}}, \Delta_{\text{near}})$;
2. if $x_{\text{near}} = \emptyset$ then
   3. return Nearest($V_{\text{active}}, x_{\text{rand}}$);
3. else
   4. return $\arg\min_{x \in x_{\text{near}}} c(x)$;

An illustration of this operation can be seen in Figure 2. All nodes $x$ within a $\Delta_{\text{near}}$ radius according to the task space distance $m(x, x_{\text{rand}})$ are checked to find which node has the best path cost $c(x)$ from the start. The best node is propagated in Algorithm 1 since it has the best chance for providing a good quality path. In the event that the $\Delta_{\text{near}}$ radius contains no nodes, the closest node to $x_{\text{rand}}$ according to $m(x, x_{\text{rand}})$ is returned as in the regular RRT framework. Only nodes in $V_{\text{active}}$ are used to determine the closest nodes. The interaction of $V_{\text{active}}$ and $V_{\text{inactive}}$ are explained in regards to Drain.

Drain: This algorithm performs a local check to determine if the newly propagated node $x_{\text{new}}$ has any neighbors $x$ in the $\Delta_{\text{drain}}$ ball according to $m(x, x_{\text{new}})$ that can provide a better path cost from the start to that local region. If another node exists in $B(x_{\text{new}}, \Delta_{\text{drain}})$ with a better path cost, the new node $x_{\text{new}}$ is removed.

Algorithm 3: Drain($\Delta_{\text{drain}}, x_{\text{new}}, G$)

1. $x_{\text{near}} \leftarrow \text{Near}(V_{\text{active}}, x_{\text{new}}, \Delta_{\text{drain}})$;
2. if $c(x_{\text{new}}) \geq \arg\min_{x \in x_{\text{near}} \setminus x_{\text{new}}} c(x)$ then
   3. $V_{\text{elite}} \leftarrow V_{\text{elite}} \setminus \{x_{\text{new}}\}$;
3. else
   4. for $x_i \in x_{\text{near}} \setminus x_{\text{new}}$ do
      5. $V_{\text{active}} \leftarrow V_{\text{active}} \setminus \{x_i\}$;
      6. $V_{\text{inactive}} \leftarrow V_{\text{inactive}} \cup \{x_i\}$;
      7. $x_{\text{del}} \leftarrow x_i$;
   8. while isLeaf($x_{\text{del}}$) and $x_{\text{del}} \in V_{\text{inactive}}$ do
      9. $x_{\text{next}} \leftarrow \text{Parent}(x_{\text{del}})$;
      10. $V_{\text{bridge}} \leftarrow V_{\text{bridge}} \setminus \{x_{\text{del}}\}$;
      11. $E \leftarrow E \setminus \{(x_{\text{next}}, x_{\text{del}})\}$;
      12. $x_{\text{del}} \leftarrow x_{\text{next}}$;

In practice, this can be implemented by not adding the node in the first place until these conditions have been checked. If the new node provides a better path cost than all other nodes within $\Delta_{\text{drain}}$, then all other nodes are removed from $V_{\text{active}}$. If the removed nodes are not ancestors to any other paths in the $V_{\text{active}}$ set, then they are deleted from the tree completely as shown in Figure 3. The removal of nodes from the nearest neighbor selection process allows for nodes with better quality paths to be selected frequently for propagation, increasing the chance that a better path can be found. Furthermore, it results in a sparse data structure that quickly converges to a constant number of nodes in bounded spaces, which depends on the $\Delta_{\text{drain}}$ radius.
Propagate: This procedure randomly selects a sequence of
controls and a duration of propagation. This random duration
must be sufficient in order to make reasonable progress away
from \( x_{\text{nearest}} \). This is especially important when the Drain
function is used and nodes are pruned. In this case the
propagation must generate a state at least \( \Delta_{\text{drain}} \) away from
the propagating state according to the metric function. If
the resulting trajectory leads to a collision, a new random
sample is drawn. A number of attempts are executed until
a collision-free node can reach outside the \( \Delta_{\text{drain}} \) region of
the propagating node.

The cost from the start is stored on each node to make cost
comparisons between neighboring nodes quick. Similarly
each node is aware whether it is in the \( \mathcal{V}_{\text{active}} \) or \( \mathcal{V}_{\text{inactive}} \)
set. To facilitate quick solution gathering, the parent of each
node is stored in order to construct the path in the tree that
leads from the start to the goal.

V. PROPERTIES

This section aims to provide high-level explanations for
the good experimental performance of the BestNearest
variant and briefly discusses aspects of the Drain function.

BestNearest: Consider the version of SPARSE-RRT that
does not employ the Drain function (i.e. \( \Delta_{\text{drain}} = 0 \)). The
reasoning in this section follows the same assumption:

\textbf{Assumption 1:} For any optimum trajectory
\( \pi^*(x_{\text{start}}, x_{\text{target}}) \), there are values \( \epsilon, \delta \in \mathbb{R} \), so that
a random shooting process with piecewise constant controls
is able to eventually generate a trajectory \( \pi \) from state
\( x'_{\text{start}} \in B(x_{\text{start}}, \epsilon) \) to state \( x'_{\text{target}} \in B(x_{\text{target}}, \epsilon) \) so that:

\[
c(\pi(x'_{\text{start}}, x'_{\text{target}})) \leq c(\pi^*(x_{\text{start}}, x_{\text{target}})) + \delta
\]

\[
\pi(x'_{\text{start}}, x'_{\text{target}}) \leq \pi^*(x_{\text{start}}, x_{\text{target}})
\]

\[
c(\pi'(x'_{\text{start}}, x'_{\text{target}})) \leq c(\pi^*(x_{\text{start}}, x_{\text{target}})) + \delta
\]

\[
\pi'(x'_{\text{start}}, x'_{\text{target}}) \leq \pi^*(x_{\text{start}}, x_{\text{target}})
\]

\[
\text{Fig. 4. Assumption: it is possible to sample a } \delta \text{-optimal trajectory that starts and ends within } \epsilon \text{ distance of } x_{\text{start}} \text{ and } x_{\text{target}} \text{ respectively.}
\]

It will also be useful to define a reachability region:

\textbf{Definition 4:} The reachability region \( \mathcal{R}(x, t_1 : t_2) \subset X_f \)
is the set of all possible states in \( X_f \) that can be reached with
a trajectory of duration between \( t_1 \) and \( t_2 \) from \( x \), given all
possible controls that can be executed at \( x \).

Consider the maximum duration of propagation \( \tau \) em-
ployed by the algorithm. Then the reachability region
\( \mathcal{R}(x_0, 0 : \tau) \) of the initial state \( x_0 \) can be defined. Given the
BestNearest function, state \( x_0 \) is guaranteed to be visited
infinitely often, as every sample in the \( \Delta_{\text{near}} \) neighborhood
of \( x_0 \) results in the selection of \( x_0 \). This implies that as
the number of iterations \( N \) of the algorithm goes to infinity, the
number of attempts to propagate controls from \( x_0 \) also goes
to infinity. This fact, together with the assumption leads to
the following lemma:

\textbf{Lemma 1:} [Asymp. near-optimality of } \pi^*(x_{\text{start}}, x_{\text{target}}) \text{]
For each state \( x \in \mathcal{R}(x_0, 0 : T) \), RRT with BestNearest
will eventually generate a trajectory to a node \( x' \in B(x, \epsilon) \)
so as: \( c(\pi(x_0, x)) \leq c(\pi^*(x_0, x)) + \delta \).

The question arises of what happens with states beyond
\( \mathcal{R}(x_0, 0 : T) \) that are not going to be directly connected to
\( x_0 \). Consider \( x_2 \in \mathcal{R}(x_0, \tau : 2\tau) \), which can be reached in
time \( t_2 \) from the root given the optimum path as in Figure 5.
Moreover, consider states \( x_{\text{opt}} \) along \( \pi^*(x_0, x_2) \) that can
be reached from \( x_0 \) in the interval \([t_2 - \tau, \tau]\). Consider the
balls \( B(x_{\text{opt}}, \epsilon) \) for all \( x_{\text{opt}} \in X_{\text{opt}} \). Construct the set \( U_1 \) as
\( \bigcup x_{\text{opt}} \in X_{\text{opt}} B(x_{\text{opt}}, \epsilon) \).

As iterations go to infinity, the algorithm will gen-
erate trajectories from \( x_0 \) to neighboring states of \( x_{\text{opt}} \)
in \( U_1 \), which are within \( \delta \) of the corresponding optimum
path \( \pi^* \). Furthermore, the algo-
rithm will select infinitely often states from the set \( U_1 \)
and propagate them forward up to time \( T \). Notice that
there are states in \( U_1 \), which can reach an \( \epsilon \) ball around
\( x_2 \). Given Assumption 1, random shooting can gener-
ate near-optimal versions of such trajectories, i.e., trajec-
tories from states in \( U_1 \) to \( B(x_2, \epsilon) \) that are within \( \delta \)
cost differential of optimum from states along \( \pi^*(x_0, x_2) \) to \( x_2 \). Thus, the algorithm
converges to overall cost from \( x_0 \) to \( B(x_2, \epsilon) \) within \( 2\delta \) of
\( c^* \) with a positive probability. This can then be extended
to each subsequent state in the tree. This implies that when
a solution is found, its cost can be bounded based on the
number of edges between the start node and the end node:

\textbf{Lemma 2:} [Asymp. near-optimality of } \pi^*(x_{\text{start}}, x_{\text{target}}) \text{]
For each state \( x \) reachable with an
optimum path between the time interval \([kt, k(T + 1)]\)
from \( x_0 \), RRT with BestNearest will eventually
generate a trajectory to a node \( x' \in B(x, \epsilon) \) so that:
\( c(\pi(x_0, x)) \leq c(\pi^*(x_0, x)) + k \cdot \delta \).

Drain: As the algorithm progresses, there will be many
balls \( B(x_c, \Delta_{\text{drain}}) \), one for each node in the tree. Let \( X_c \)
denote the set of all such center states.

\textbf{Lemma 3:} [Uniqueness in Each Ball] For each center
state \( x_c \in X_c \), there does not exist another state in the
\( B(x_c, \Delta_{\text{drain}}) \) in \( \mathcal{V}_{\text{active}} \).

The Drain function guarantees that the above lemma
holds. The algorithm does not allow two states in one ball
region to exist for propagation purposes.

\textbf{Lemma 4:} [Finite Number of Balls] For a bounded \( X_f \)
region, the number of states in the set \( X_c \) is finite. Let
\( N(\Delta_{\text{drain}}, X_f) \) denote this maximum number.
With Lemma 3, Lemma 4 is immediate. The search space can only hold a maximum number of balls at any time if \( m \) is bounded. The authors believe that the \textit{Drain} function operates similarly to the \textit{BestNearest} one and provides near-optimality guarantees, where the bounds become worse as the optimum cost increases. This is also indicated by the experimental performance provided in the following section. Future research will aim to show this property, as well as the near-optimality properties of the integrated algorithm.

The proposed algorithm will not necessarily produce a trajectory to reach any state. Figure 6 illustrates a situation where \textit{SPARSE-RRT} has difficulties. There is a narrow passage and the chosen \( \Delta_{\text{drain}} \) is larger than the entrance of the narrow passage. The part of the space that is inside the narrow passage is hard to reach. A possible solution is to switch between \textit{RRT} and \textit{SPARSE-RRT}. The explorative properties of \textit{RRT} will be able to compensate for this deficiency.

The complexity of this approach is dominated practically by the two nearest neighbor queries, one for \textit{BestNearest} and one for \textit{Drain}. \textit{SPARSE-RRT} maintains a sparse data structure, so the amount of work to perform these queries is much smaller than other methods. Asymptotically, the complexity is equivalent to that of \textit{RRT}\textsuperscript{*} and its derivatives.

VI. EVALUATION

**Setup:** Experiments with systems that were modeled through numerical integration of dynamics equations were used to evaluate the proposed algorithm in the following setups:

- A second-order car moving among obstacles. This is a 5-dim state space (Cartesian coordinates, orientation, velocity and steering angle) and a 2-dim control space (acceleration and rate of change of steering angle). The task space metric \( m \) is the time to move between each \((x,y)\) location with the average velocity between the two states.
- A one-link pendulum with undamped motion. The state space of the system is 2-dim (one angle and its velocity). The control space is 1-dim (torque). Manhattan distance is used for \( m \).
- A 1-dim double integrator. The state space of this system is 2-dim (position and velocity) and the control space is 1-dim (force). \( m \) is the traditional Euclidean distance using both the position and velocity elements of the state.
- A two-link Acrobat (passive “shoulder”, active “elbow”). The state space of the system is 4-dim (two angles and their velocities). The control space is 1-dim (torque on the second joint). \( m \) is defined by the angle and velocity of the end-effector as if it was a one link pendulum (the angle formed by the end effector and the distance from the passive joint).

While there are alternatives to planning for some of these systems, they are standard benchmarks for non-linear dynamics and under-actuation. They can be used to evaluate the performance of these modifications under a regime where no BVP solver is employed. All combinations of the two modifications have been tested on these systems. The approach used in [7] is employed as a comparison point. The algorithms were compared in terms of the number of nodes in each tree and the cost of the solution after a fixed amount of time (10 minutes for these experiments).

To perform nearest neighbor queries, a structure based on PRM\textsuperscript{*} [5] is used. Internally, a node keeps track of a number of neighbors based on the metric function required in order to asymptotically provide the shortest path between two states, a result from percolation theory. This graph can then be traversed to determine nearest neighbors within a radius as required by all the algorithms being evaluated. This metric imposes a larger cost on addition of nodes, while making removal and queries very fast.

**Experiments:** Table I shows the solution lengths returned by the algorithms averaged over all experiments. For the car-like system, the goal was to reach a certain location in a simple maze-like environment. For the Acrobat and pendulum, the goal was to swing up toward its unstable fixed point. The goal for the double integrator is to move to the origin position.

<table>
<thead>
<tr>
<th>System</th>
<th>Algorithm</th>
<th>Iterations</th>
<th>Nodes</th>
<th>Length (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double Int.</td>
<td>RRT</td>
<td>299749</td>
<td>299750</td>
<td>7.86</td>
</tr>
<tr>
<td></td>
<td>Drain</td>
<td>511137</td>
<td>35979</td>
<td>6.67</td>
</tr>
<tr>
<td></td>
<td>BestNearest</td>
<td>299744</td>
<td>30684</td>
<td>5.86</td>
</tr>
<tr>
<td></td>
<td>SPARSE-RRT</td>
<td>580187</td>
<td>35195</td>
<td>6.6675</td>
</tr>
<tr>
<td></td>
<td>RRT\textsuperscript{*} w/ Shooting</td>
<td>66438</td>
<td>66439</td>
<td>7.76</td>
</tr>
<tr>
<td>Pendulum</td>
<td>RRT</td>
<td>521582</td>
<td>521529</td>
<td>4.785</td>
</tr>
<tr>
<td></td>
<td>Drain</td>
<td>761252</td>
<td>761252</td>
<td>7.428</td>
</tr>
<tr>
<td></td>
<td>BestNearest</td>
<td>160856</td>
<td>160857</td>
<td>2.5275</td>
</tr>
<tr>
<td></td>
<td>SPARSE-RRT</td>
<td>668167</td>
<td>38699.5</td>
<td>2.385</td>
</tr>
<tr>
<td></td>
<td>RRT\textsuperscript{*} w/ Shooting</td>
<td>310289</td>
<td>310290</td>
<td>6.3825</td>
</tr>
<tr>
<td>Acrobat</td>
<td>RRT</td>
<td>191081</td>
<td>154482</td>
<td>8.775</td>
</tr>
<tr>
<td></td>
<td>Drain</td>
<td>301122</td>
<td>24451</td>
<td>5.22</td>
</tr>
<tr>
<td></td>
<td>BestNearest</td>
<td>184367</td>
<td>148824</td>
<td>5.8475</td>
</tr>
<tr>
<td></td>
<td>SPARSE-RRT</td>
<td>295923</td>
<td>23862</td>
<td>4.245</td>
</tr>
<tr>
<td></td>
<td>RRT\textsuperscript{*} w/ Shooting</td>
<td>313807</td>
<td>313807</td>
<td>6.67</td>
</tr>
<tr>
<td>2nd-order Car</td>
<td>RRT</td>
<td>114200</td>
<td>83981</td>
<td>76.22</td>
</tr>
<tr>
<td></td>
<td>Drain</td>
<td>129721</td>
<td>3672.5</td>
<td>32.995</td>
</tr>
<tr>
<td></td>
<td>BestNearest</td>
<td>119090</td>
<td>76062</td>
<td>38.175</td>
</tr>
<tr>
<td></td>
<td>SPARSE-RRT</td>
<td>135741.5</td>
<td>2967</td>
<td>35.655</td>
</tr>
<tr>
<td></td>
<td>RRT\textsuperscript{*} w/ Shooting</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**TABLE I**

Solution length achieved by different algorithms for the same problem averaged over all experiments.

The results in Table I compare the relative size of the data structure for each algorithm after a set amount of time. \textit{RRT} will not generally improve its solution over time, so the number of nodes reported is the amount of nodes that are able to be added in the time noted. The second-order car experiments for \textit{RRT}\textsuperscript{*} with Shooting provided some difficulty in providing results that improved upon \textit{RRT}. For this reason, the numbers are omitted from this table.

\textit{RRT} converges to a random suboptimal value in terms of path length. The overall result is that \textit{Drain} provides a much sparser data structure with some improvement of
Fig. 7. An example of the solution returned. The system shown is a two-link Acrobot, where the objective is to reach the stable fixed point where the system is balanced, without hitting the two obstacles. The path of the end-effector is shown. The region at the top is considered the goal region.

path quality. BestNearest provides similar amount of iterations as RRT while focusing on improving path costs. Also, all combinations of these methods allow for more iterations than RRT* with Shooting.

Figure 8 illustrates the rate of improvement for all versions of the algorithm. Regardless of the initial position, all of the algorithms show improvement of solutions over time. SPARSE-RRT is able to provide similar convergence properties as RRT* with Shooting while being much simpler to implement and requiring a much smaller data structure.

Figure 9 shows the rate of improvement for each algorithm throughout the entire tree. Due to RRT not improving any path costs over time, the average cost for all nodes will only increase. If the algorithm quickly finds a solution, then it can return a shorter one. For SPARSE-RRT and its other versions, the improvement over time is apparent, both due to removal of high cost nodes and generation of better nodes.

Remarks: The relationship between $\Delta_{\text{drain}}$ and $\Delta_{\text{near}}$ requires further exploration. During experimentation, several of these parameters were tested and depending on the underlying system, resulted in interesting behavior. Some combinations would make the algorithm able to quickly find good initial solutions. These experiments were executed with manually selected parameters.

VII. DISCUSSION

This work introduces an extension of RRT that can improve path quality over time for kinodynamic problems where a BVP solver is unavailable. Restricting the nodes that can be added to the tree to only those that can help improve path cost helps to focus the search to parts of the space where improvements need to be made. Also, by allowing nodes to stake claim to a region surrounding itself, SPARSE-RRT only needs to maintain a small data structure, and therefore can run more efficiently than its alternatives.

For future work, a more rigorous analysis of the approach will be conducted. This analysis needs to provide a definite bound on the solution quality that can be returned by both of these methods. The necessity of the assumptions will also be evaluated in an attempt to find the weakest set of requirements in order to argue about asymptotic near-optimality.

Fig. 8. Comparison of Improvement Rates for Acrobot and second order car. The plots display the relative path cost over time. The path costs are relative to the best path found.
The benefit of integrating with other works, which provide improved supporting structures, such as local planners and nearest neighbor structures [22], [23], is also a promising direction to pursue. These additions should provide a means of improving the results of this method to provide faster convergence.

Another possible direction is anytime planning or replanning. The sparse nature of the resulting data structure appears to be appropriate to provide such primitives.

REFERENCES