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) 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 reasons over 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.

Work by the authors has been supported by NSF CNS 0932423. Any conclusions expressed here are of the authors and do not reflect the views of the sponsors. The authors are with the Computer Science Department, Rutgers University, 110 Frelinghuysen Road, 08854, Piscataway, NJ, USA, email: kostas.bekris at cs.rutgers.edu.

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 $\mathbb{X}$ 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 $\mathbb{X}$ 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].

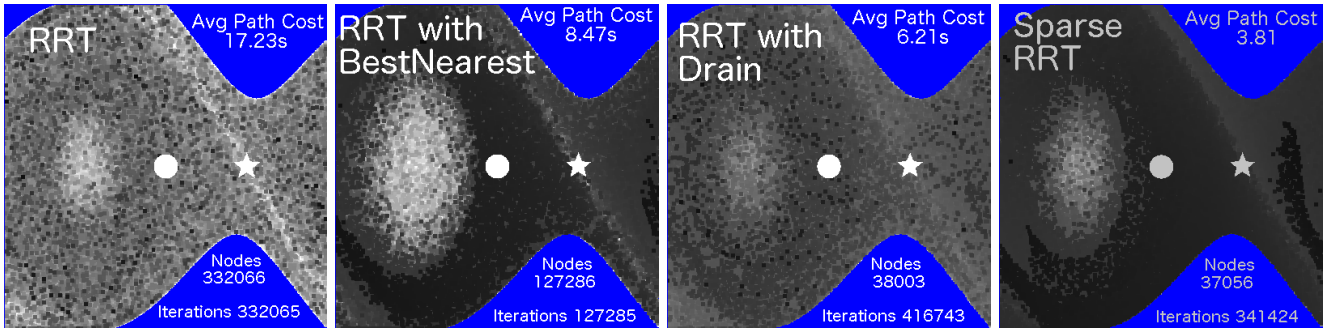

Fig. 1. Phase plots that show best path cost at each point in the one-link pendulum state space for each of the proposed modifications. x-axis: pendulum angle, y-axis: velocity. Blue corresponds to unexplored regions of the state space. The circle is state $\{0, 0\}$, a horizontal placement of the pendulum, the star is state $\{\frac{\pi}{2}, 0\}$, an upward configuration. Colors are computed by dividing the best path cost to a state in a pixel by a predefined value (20.0 for RRT and 10.0 for the other methods) and then mapping the result to the range [0,255]. All algorithms were executed for the same amount of time (5 min). For the last two methods that provide a sparse representation, each state is coloring a 3x3 local neighborhood. The best path cost for each pixel is displayed.

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 α to node β gets to state β' within a small distance of β, a tree connection between α and β is made. This results in a re-propagation of the subtree of β from β' to maintain the validity of each subsequent state, which may result in node pruning if collisions occur. This method does not provably achieve asymptotic optimality but can behave well together with numerical methods for decreasing the gap between β and β' [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: $\dot{x} = f(x, u), \quad g(x, \dot{x}) \leq 0$ (1) where f, g are smooth; $x \in \mathbb{X}$ is a state and $u \in \mathbb{U}$ is a control. The set of collision-free states is $\mathbb{X}_f \subset \mathbb{X}$. The robot is in an initial state x_0 and must reach a goal region $\mathbb{X}_G$ defined by a ball of radius δ around a goal state $\mathcal{B}(x_{goal}, \delta)$.

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

The set Π is the set of all trajectories in $\mathbb{X}_f$.

Definition 2: [Kinodynamic Motion Planning] Given the tuple ($\mathbb{X}_f$, $\mathbb{U}$, x_0, $\mathbb{X}_G$), find a trajectory $\pi \in \Pi$ that is a solution, i.e., $\pi(0) = x_0$ and $\pi(T) \in \mathbb{X}_G$ 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 ALG is asymptotically near-optimal for a kinodynamic motion planning problem ($\mathbb{X}_f$, $\mathbb{U}$, x_0, $\mathbb{X}_G$) and a cost function c: $\Pi \to \mathbb{R}_{\geq 0}$ that admit a robustly optimal solution with finite cost c^*, when the probability that 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-RRTand outlined in Algorithm 1, provides the benefits of both modifications, called BestNearest and Drain.

Algorithm 1: SPARSE-RRT($\mathbb{X}_f$,$\mathbb{U}$,x_0,$\mathbb{X}_G$,N,Δ_{drain},Δ_{near})

$\mathbb{V}_{active} \leftarrow \{x_0\}$,$\mathbb{V}_{inactive} \leftarrow \emptyset$, $\mathbb{E} \leftarrow \emptyset$, $i \leftarrow 0$;
$\mathbb{V} = \mathbb{V}_{active} + \mathbb{V}_{inactive}$; $G = \{\mathbb{V}, \mathbb{E}\}$;
while $i < N$ **do**
  $x_{rand} \leftarrow$ Sample($\mathbb{X}$);
  $x_{nearest} \leftarrow$ BestNearest($\mathbb{V}_{active}, x_{rand}, \Delta_{near}$);
  $x_{new} \leftarrow$ Propagate($x_{nearest}$, $\mathbb{U}$, $\mathbb{X}_f$);
  $\mathbb{V}_{active} \leftarrow \mathbb{V}_{active} \cup \{x_{new}\}$;
  $\mathbb{E} \leftarrow \mathbb{E} \cup \{(x_{nearest}, x_{new})\}$;
  Drain($\Delta_{drain}, x_{new}, G$);
  $i \leftarrow i + 1$;

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 Δ_{drain} and Δ_{near} will be clarified below. Typically, Δ_{drain} is in the order or smaller than the goal radius δ, while Δ_{near} is greater than Δ_{drain}. Special cases of the algorithm arise for different values of Δ_{drain} and Δ_{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 search. This modification was originally proposed in [6] and this work will analyze the effects it has in returning solutions.

Algorithm 2: BestNearest($\mathbb{V}_{active}$,x_{rand},Δ_{near})

$X_{near} \leftarrow$Near($\mathbb{V}_{active}, x_{rand}, \Delta_{near}$);
if $X_{near} = \emptyset$ **then**
  return Nearest($\mathbb{V}_{active}, x_{rand}$);
else
  return $\text{argmin}_{x \in X_{near}}\, c(x)$;

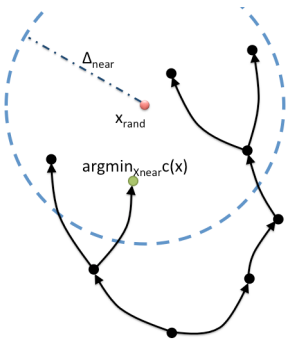

Fig. 2. The selection of the best neighbor in BestNearest. The best path cost node in $\mathcal{B}(x_{rand}, \Delta_{near})$ is selected.

An illustration of this operation can be seen in Figure 2. All nodes x within a Δ_{near} radius according to the task space distance $m(x, x_{rand})$ 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 Δ_{near} radius contains no nodes, the closest node to x_{rand} according to $m(x, x_{rand})$ is returned as in the regular RRT framework. Only nodes in $\mathbb{V}_{active}$ are used to determine the closest nodes. The interaction of $\mathbb{V}_{active}$ and $\mathbb{V}_{inactive}$ are explained in regards to Drain.

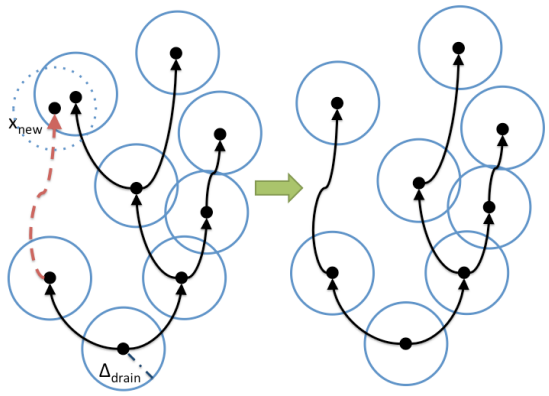

Fig. 3. Illustration of the tree before and after Drain. When a new node is created, the surrounding Δ_{drain} region is checked to see if any other nodes with better cost exist. If none exist, the new node is added and other nodes are moved to $\mathbb{V}_{inactive}$ and possibly deleted.

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

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

$X_{near} \leftarrow$Near($\mathbb{V}_{active}, x_{near}, \Delta_{drain}$);
if $c(x_{new}) \geq \text{argmin}_{x \in X_{near} \setminus x_{new}}\, c(x)$ **then**
  $\mathbb{V}_{elite} \leftarrow \mathbb{V}_{elite} \setminus \{x_{new}\}$;
else
  for $x_i \in X_{near} \setminus x_{new}$ **do**
    $\mathbb{V}_{active} \leftarrow \mathbb{V}_{active} \setminus \{x_i\}$;
    $\mathbb{V}_{inactive} \leftarrow \mathbb{V}_{inactive} \cup \{x_i\}$;
    $x_{del} \leftarrow x_i$;
    while IsLeaf(x_{del}) **and** $x_{del} \in \mathbb{V}_{inactive}$ **do**
      $x_{next} \leftarrow$Parent(x_{del});
      $\mathbb{V}_{bridge} \leftarrow \mathbb{V}_{bridge} \setminus \{x_{del}\}$;
      $\mathbb{E} \leftarrow \mathbb{E} \setminus \{(x_{next}, x_{del})\}$;
      $x_{del} \leftarrow x_{next}$;

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 Δ_{drain}, then all other nodes are removed from $\mathbb{V}_{active}$. If the removed nodes are not ancestors to any other paths in the $\mathbb{V}_{active}$ 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 Δ_{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 $x_{nearest}$. 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 Δ_{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 Δ_{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 $\mathbb{V}_{active}$ or $\mathbb{V}_{inactive}$ 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. $\Delta_{drain} = 0$). The reasoning in this section employs the following assumption:

Assumption 1: For any optimum trajectory $\pi^*(x_{start}, x_{target})$, 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 π from state $x'_{start} \in \mathcal{B}(x_{start}, \epsilon)$ to state $x'_{target} \in \mathcal{B}(x_{target}, \epsilon)$ so that:

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

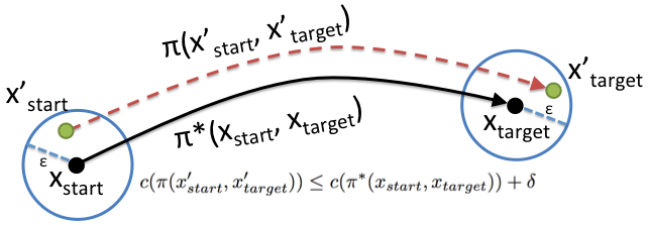

Fig. 4. Assumption: it is possible to sample a δ-optimal trajectory that starts and ends within ϵ distance of x_{start} and x_{target} respectively.

It will also be useful to define a reachability region:

Definition 4: The reachability region $\mathcal{R}(x, t_1 : t_2) \subset \mathbb{X}_f$ is the set of all possible states in $\mathbb{X}_f$ 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 τ employed 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 Δ_{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:

Lemma 1: [Asymp. near-optimality of x_0 neighborhood] For each state $x \in \mathcal{R}(x_0, 0 : T)$, `RRT` with `BestNearest` will eventually generate a trajectory to a node $x' \in \mathcal{B}(x, \epsilon)$ 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_{opt} along $\pi^*(x_0, x_2)$ that can be reached from x_0 in the interval $[t_2 - \tau, \tau]$. Consider the balls $\mathcal{B}(x_{opt}, \epsilon)$ for all $x_{opt} \in X_{opt}$. Construct the set U_1 as $\bigcup_{x_{opt} \in X_{opt}} \mathcal{B}(x_{opt}, \epsilon)$.

As iterations go to infinity, the algorithm will generate trajectories from x_0 to neighboring states of $x_{opt} \in U_1$, which are within δ of the corresponding optimum path π^*. Furthermore, the algorithm 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 ϵ ball around x_2. Given Assumption 1, random shooting can generate near-optimal versions of such trajectories, i.e., trajectories from states in U_1 to $\mathcal{B}(x_2, \epsilon)$ that are within δ 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 $\mathcal{B}(x_2, \epsilon)$ within 2δ 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:

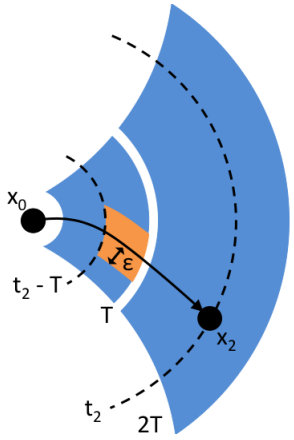

Fig. 5. The reachability regions of x_0 with respect to time. The orange region is defined by the union of $\mathcal{B}(x, \epsilon)$ for all x on the optimum trajectory π^* between time $t_2 - \tau$ and T. This region has a positive probability of being selected to propagate toward x_2.

Lemma 2: [Asymp. near-optimality of `RRT` with `BestNearest`] 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 $x' \in \mathcal{B}(x, \epsilon)$ 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 $\mathcal{B}(x_c, \Delta_{drain})$, one for each node in the tree. Let X_c denote the set of all such center states.

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

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.

Lemma 4: [Finite Number of Balls] For a bounded $\mathbb{X}_f$ region, the number of states in the set X_c is finite. Let $N(\Delta_{drain}, X_{free})$ 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 is is bounded. The authors believe that the Drain function operates similarly to the 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 SPARSE-RRT has difficulties. There is a narrow passage and the chosen Δ_{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 RRT and SPARSE-RRT. The explorative properties of RRT will be able to compensate for this deficiency.

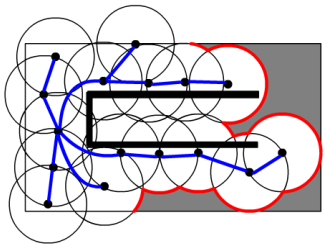

Fig. 6. When Δ_{drain} is chosen to be large, narrow passages may become difficult to explore.

The complexity of this approach is dominated practically by the two nearest neighbor queries, one for BestNearest and one for Drain. 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 RRT* 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 Acrobot (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* [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 Acrobot 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.

System	Algorithm	Iterations	Nodes	Length (s)
Double Int.	RRT	299749	299750	7.86
	Drain	511137	35979	6.67
	BestNearest	269244	269245	6.87
	SPARSE-RRT	508187	35495	6.6675
	RRT* w/ Shooting	66438	66439	7.76
Pendulum	RRT	521528	521529	4.785
	Drain	761252	39215	2.5275
	BestNearest	160856	160857	2.6825
	SPARSE-RRT	668167	38699.5	2.385
	RRT* w/ Shooting	310289	310290	6.3825
Acrobot	RRT	191081	154482	8.775
	Drain	301122	24451	5.22
	BestNearest	184367	148824	5.8475
	SPARSE-RRT	295923	23862	4.245
	RRT* w/ Shooting	313806	313807	6.67
2nd-order Car	RRT	114200	83981	76.22
	Drain	129721	3672.5	32.995
	BestNearest	119090	76062	38.175
	SPARSE-RRT	135741.5	2967	35.655
	RRT* w/ Shooting	N/A	N/A	N/A

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. 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 RRT* with Shooting provided some difficulty in providing results that improved upon RRT. For this reason, the numbers are omitted from this table.

RRT converges to a random suboptimal value in terms of path length. The overall result is that Drain provides a much sparser data structure with some improvement of

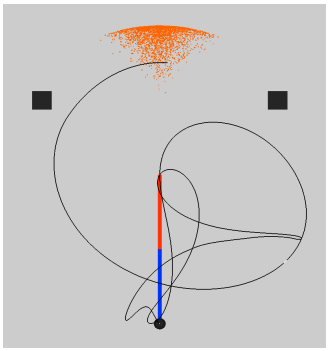

Fig. 7. An example of the solution returned. The system shown is a two-link Acrobot, where the objective is to reach the stable fixed point where the system is balanced , without hitting the two obstacles. The path of the end-effector is shown. The region at the top is considered the goal region.

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 Δ_{drain} and Δ_{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.

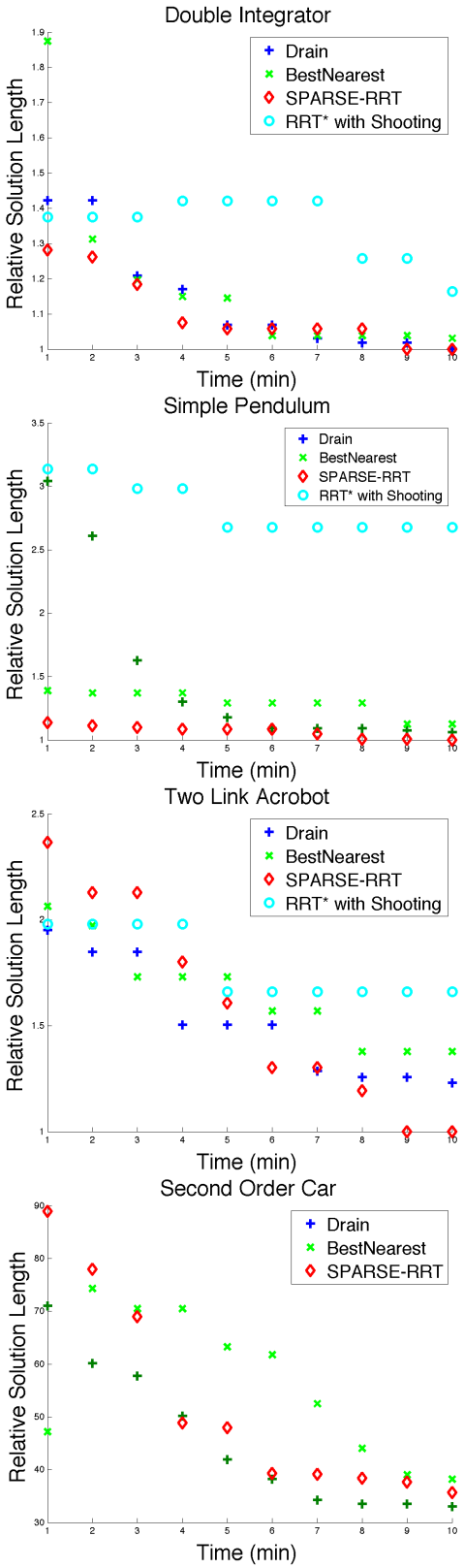

Fig. 8. Comparison of Improvement Rates for Acrobot and second order car. The plots display the relative path cost over time. The path costs are relative to the best path found.

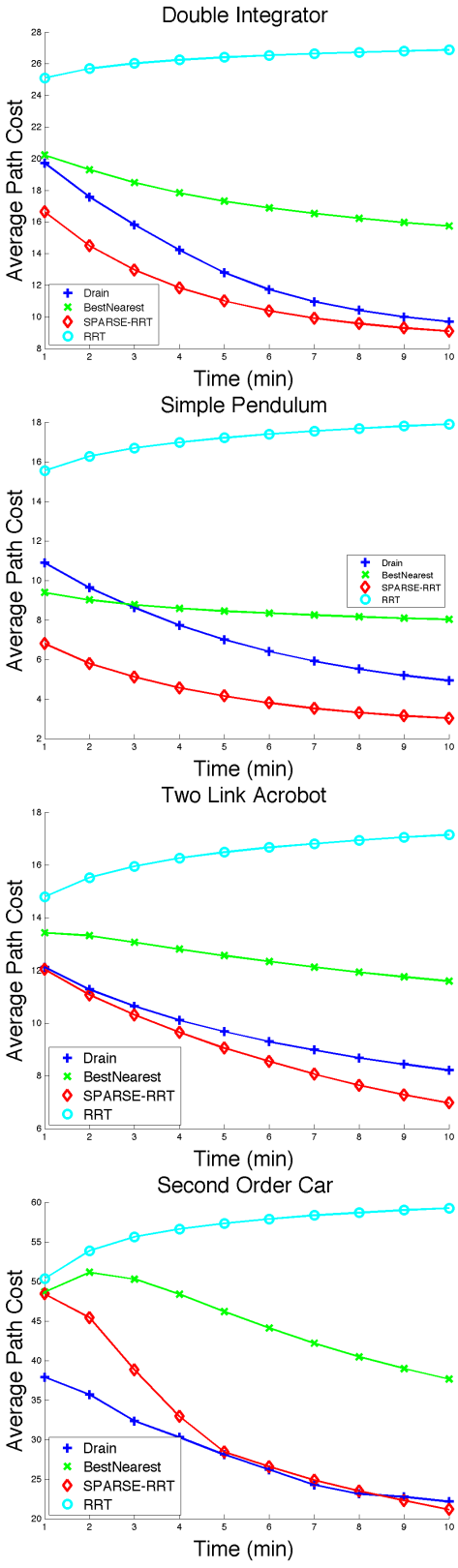

Fig. 9. Comparison of average path cost of all nodes over time. `RRT` node costs will continue to increase since no effort to improve path costs is made and new nodes continue to be added with worse path costs. The combination of `Drain` and `BestNearest` in `SPARSE-RRT` provide a large improvement in path cost over time.

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

[1] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion*. The MIT Press, 2005.

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

[3] S. LaValle and J. Kuffner. Rapidly exploring random trees: Progress and prospects. In *WAFR*, pages 293–308, 2001.

[4] O. Nechushtan, B. Raveh, and D. Halperin. Sampling-Diagrams Automata : a Tool for Analyzing Path Quality in Tree Planners. In *WAFR*, 2010.

[5] S. Karaman and E. Frazzoli. Sampling-based Algorithms for Optimal Motion Planning. *IJRR*, 30(7):846–894, June 2011.

[6] C. Urmson and R. Simmons. Approaches for Heuristically Biasing RRT Growth. In *IEEE/RSJ IROS*, pages 1178–1183, 2003.

[7] J.-H. Jeon, S. Karaman, and E. Frazzoli. Anytime Computation of Time-Optimal Vehicle Maneuvers using the RRT*. In *CDC*, 2011.

[8] S. M. LaValle and J. J. Kuffner. Randomized Kinodynamic Planning. *IJRR*, 20(5):378–400, May 2001.

[9] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock. Randomized Kinodynamic Motion Planning with Moving Obstacles. *IJRR*, 21(3):233–255, March 2002.

[10] B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic Motion Planning. *Journal of the ACM*, 40(5):1048–1066, 1993.

[11] P. Cheng and S. M. LaValle. Reducing Metric Sensitivity in Randomized Trajectory Design. In *IEEE/RSJ IROS*, pages 43–48, 2001.

[12] A. Shkolnik, M. Walter, and R. Tedrake. Reachability Sampling for Planning under Differential Constraints. In *IEEE ICRA*, 2009.

[13] E. Glassman and R. Tedrake. A Quadratic Regulator-based Heuristic for Rapidly Exploring State Space. In *IEEE ICRA*, 2010.

[14] W. V. Weghe, D. Ferguson, and S. Srinivasa. Randomized Path Planning for Redundant Manipulators without Inverse Kinematics. In *IEEE Int. Conf. on Humanoid Robots*, 2007.

[15] M. Stilman. Task Constrained Motion Planning in Robot Joint Space. In *IEEE/RSJ IROS*, pages 3074–3081, 2007.

[16] P. Cheng, E. Frazzoli, and S. M. LaValle. Improving the Performance of Sampling-based Planners by using a Symmetry-Exploiting Gap Reduction Algorithm. In *IEEE ICRA*, 2004.

[17] F. Lamiraux, E. Ferre, and E. Vallee. Kinodynamic Motion Planning: Connecting Exploration Trees using Trajectory Optimization Methods. In *IEEE ICRA*, pages 3987–3992, 2004.

[18] D. Ferguson and A. Stentz. Anytime RRTs. In *IEEE/RSJ IROS*, pages 5369–5375, October 2006.

[19] S. Karaman, M.R. Walter, A. Perez, E. Frazzoli, and S. Teller. Anytime Motion Planning using the RRT. In *IEEE ICRA*, May 2011.

[20] R. Alterovitz, S. Patil, and A. Derbakova. Rapidly-Exploring Roadmaps: Weighing Exploration vs. Refinement in Optimal Motion Planning. In *IEEE ICRA*, 2011.

[21] J. D. Marble and K. E. Bekris. Asymptotically Near-Optimal is Good Enough for Motion Planning. In *ISRR*, 2011.

[22] D. Webb and J. van Den Berg. Kinodynamic RRT*: Asymptotically Optimal Motion Planning for Robots with Linear Differential Constraints. In *IEEE ICRA*, 2013.

[23] S. Karaman and E. Frazzoli. Sampling-Based Optimal Motion Planning for Non-holonomic Dynamical Systems. In *IEEE ICRA*, 2013.

[24] Y. Abbasi-Yadkori, J. Modayil, and C. Szepesvari. Extending Rapidly-exploring Random Trees for Asymptotically Optimal Anytime Motion Planning. In *IEEE/RSJ IROS*, 2010.

[25] A. Shkolnik and R. Tedrake. Sample-Based Planning with Volumes in Configuration Space. arxiv:1109.3145v1, MIT, 2011.