IEEE Transactions on Robotics, 2005, vol. 21(4), pp. 587–608

Sampling-Based Roadmap of Trees for Parallel Motion Planning

Erion Plaku, Kostas E. Bekris, Brian Y. Chen, Andrew M. Ladd, Lydia E. Kavraki

***Abstract*— This paper shows how to effectively combine a sampling-based method primarily designed for multiple query motion planning (Probabilistic Roadmap Method - PRM) with sampling-based tree methods primarily designed for single query motion planning (Expansive Space Trees, Rapidly-Exploring Random Trees, and others) in a novel planning framework that can be efficiently parallelized. Our planner not only achieves a smooth spectrum between multiple query and single query planning but it combines advantages of both. We present experiments which show that our planner is capable of solving problems that cannot be addressed efficiently with PRM or single-query planners.**

A key advantage of our planner is that it is significantly more decoupled than PRM and sampling-based tree planners. Exploiting this property, we designed and implemented a parallel version of our planner. Our experiments show that our planner distributes well and can easily solve high-dimensional problems that exhaust resources available to single machines and cannot be addressed with existing planners.

***Index Terms*— Motion planning, sampling-based planning, parallel algorithms, roadmap, tree, PRM, EST, RRT, SRT.**

I. Introduction

HIGH-DIMENSIONAL problems such as those arising in planning with flexible objects [35], [37], [45], reconfigurable robots [56], complex planning instances [52], manipulation planning [51], and computational biology search problems [6], [7] test the limits of current motion planner implementations. One important avenue for solving such problems is to effectively use parallelism in motion planning. Our work describes a robust planner, which provides a smooth transition from single query to multiple query planners and can be used for problems that are beyond the capabilities of current planners. The planner can be used in a sequential implementation or a powerful parallel implementation.

Sampling-based planners have been used extensively during the last decade for multiple query or single query motion planning [1], [3], [9], [10], [25], [30], [32], [34], [35], [40], [47], [49], [52]. In multiple query motion planning, typically a roadmap is built during a preprocessing phase in order to quickly respond to on-line queries [10], [32], [37]. Alternatively, in single query planning, there is no preprocessing phase and all computations occur during query resolution. Such planners typically explore the space using a single or a bi-directional tree [8], [25], [26], [39], [40], [49], [52]. Recent papers (e.g., [25], [40]) contain extensive references to sampling-based motion planners.

Manuscript received April 13, 2004; revised November 23, 2004. Work on this paper by the authors has been partially supported by NSF 0308237, NSF ITR 0205671, a Texas ATP grant, and a Sloan Fellowship to L. Kavraki. A. M. Ladd is also partially supported by an FCAR grant. Experiments were run on equipment supported by AMD and NSF EIA 0216467. Preliminary versions of this work have appeared in International Symposium on Robotics Research, 2003 and International Conference on Intelligent Robots and Systems, 2003.

The authors are with Rice University, Department of Computer Science, Houston, TX 77005 USA, email: {plakue, bekris, brianyc, aladd, kavraki}@cs.rice.edu. The corresponding author is Lydia E. Kavraki.

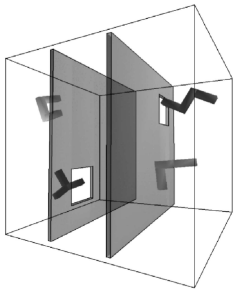

Fig. 1. A scene from our benchmarks. In problem "narrow4h2", each robot must go through two very narrow passages.

The Probabilistic Roadmap Method (PRM) is an efficient and easy to implement planner primarily designed for multiple query motion planning problems [29]–[32], [47], [54]. PRM operates by sampling configurations in the free configuration space, $\mathcal{C}_{\text{free}}$, and connecting them using a local planner. Although a typical implementation uses a very simple local planner and uniform pseudo-random sampling, it has been shown that a variety of alternate approaches ranging in sophistication and cost can be applied without sacrificing correctness in hopes of obtaining a faster planner [20]. Indeed, two of the key issues in the context of PRM are the power of the local planner [4], [11], [28], [32] and the way sampling is performed [3], [12], [13], [20], [21], [23], [24], [27], [32], [38], [55].

In this paper we replace the local planner of PRM with a single query sampling-based motion planner. Among the single query planners that have been developed recently, Expansive Space Trees (ESTs) [1] [25], [26] and Rapidly-Exploring Random Trees (RRTs) [39], [40] have been very successful and are used in our work. However, other sampling-based tree planners can be used (e.g., [35]).

The idea of using multiple trees has been mentioned in [39] and used in [41], [46], [53]. In [41], a single-query method is developed that takes advantage of the exploration done in solving previous queries. Initially a bi-directional RRT creates two trees to answer a single query. At the end, the trees are not discarded but kept to answer subsequent queries. The planner proceeds by generating random configurations and attempting

[1]The acronym EST to describe Expansive Space Trees does not appear in the original papers, but is used in this paper for convenience.

to connect each new configuration to every tree created as a result of answering previous queries. In [46], a roadmap is initially constructed using PRM and then enhanced using RRTs to connect different connected components of the roadmap. The approach taken in [53] is for single-query planning. The method keeps some of the samples that the initial RRT could not connect to as possible roots from where to grow other trees. Attempts are made to connect neighboring roots together to form local trees or to connect the roots to already formed local trees and the trees rooted at the initial and goal configurations.

Our planner creates a roadmap of trees that integrates the global sampling properties of PRM with the local sampling properties of single query tree-based planners. Earlier versions of our planner [1], [9] were termed Probabilistic Roadmap of Trees (PRT), but as suggested in [13], [42], the probabilistic sampling could be replaced with other sampling schemes. To emphasize the importance of sampling, we name our planner Sampling-based Roadmap of Trees (SRT).

Our work is important in many respects. We propose an integration scheme that fully exploits successful sampling-based methods. Firstly, we obtain a planner which is faster than PRM and more robust than the tree planners that we used, namely ESTs and RRTs. SRT provides a smooth spectrum between single query and multiple query planning that combines the advantages of both. In our work, we take advantage of recent very effective sampling methods employed by ESTs and RRTs and provide a new sampling scheme for PRM. It should be noted that the proposed overall sampling of SRT is in the spirit of non-uniform sampling and refinement techniques used in earlier work of PRM. Secondly, the local exploration of the configuration space can be done independently by using tree planners such as ESTs and RRTs. This property makes SRT significantly more decoupled than PRM and tree planners such as ESTs and RRTs and allows for an efficient parallelization. Although many subroutines of PRM can be run effectively in a highly distributed fashion, efficient coordination of various processing resources requires significant additional algorithmic design. By increasing the power of the local planner and by using more complex milestones, SRT can distribute its computation almost evenly among processors, requires little communication, and allows us to solve very high-dimensional problems and problems that exceed the resources available to the sequential implementation.

This paper presents experiments with up to 72 degrees of freedom (DOFs) where SRT obtains a solution at a fraction of the running time needed by PRM, EST, or RRT. Fig. 1 shows an example with 24-DOFs. We were able to obtain nearly linear speedup for parallel SRT. As with other motion planners, generalizations of SRT to many other kinematic planning problems are straightforward and the use of RRTs and ESTs as subroutines gives a natural way to extend the planner to kinodynamic planning instances [18], [25], [40].

In section II we describe the SRT algorithm. Section III describes the parallelization of the SRT algorithm. In section IV we describe the experimental setup, the set of benchmarks used to test the efficiency of our planner, and the results obtained. We conclude in section V with a discussion on SRT and possibilities for future work.

TABLE I
SAMPLING-BASED ROADMAP OF TREES (SRT) ALGORITHM.

Input: K, number of milestones.
Output: A roadmap $G_T = (V_T, E_T)$.

1: $V_T \leftarrow \emptyset$, $E_T \leftarrow \emptyset$, $Q \leftarrow \emptyset$, $E_C \leftarrow \emptyset$.
2: **while** $|V_T| < K$ **do**
3: $T \leftarrow$ build tree rooted at a collision-free random config.
4: $V_T \leftarrow V_T \cup \{T\}$.
5: $Q \leftarrow Q \cup \{q_T\}$, where q_T is the representative of T.
6: **for all** $T \in V_T$ **do**
7: $S_{\text{close}} \leftarrow$ a set of n_{c} closest $q_{T'} \in Q$ to q_T.
8: $S_{\text{rand}} \leftarrow$ a set of n_{r} random $q_{T'} \in Q$ to q_T.
9: $E_C \leftarrow E_C \cup \{(T, T') : q_{T'} \in S_{\text{close}} \cup S_{\text{rand}}\}$.
10: **for all** $(T_1, T_2) \in E_C$ **do**
11: **if not** CONNECTED(T_1, T_2) **and** CONNECT(T_1, T_2) **then**
12: $E_T \leftarrow E_T \cup \{(T_1, T_2)\}$.

II. SRT PLANNER

SRT constructs a roadmap aimed at capturing the connectivity of $\mathcal{C}_{\text{free}}$ and then uses the roadmap to answer multiple queries [1], [9]. SRT is designed primarily as a multiple query planner, but it should be noted that it is most effective for difficult planning problems, as our experiments in section IV suggest. For such problems, there is no clear distinction between single versus multiple query methods.

The nodes of the roadmap are not single configurations but trees, which are referred to as milestones. Connections between milestones are computed by sampling-based tree planners. The tree planners that we have used are RRTs [39], [40] and ESTs [25], [26]. The pseudocode for SRT is in Table I.

A roadmap is an undirected graph $G = (V, E)$ over a finite set of configurations $V \subset \mathcal{C}_{\text{free}}$ and each edge $(q', q'') \in E$ represents a local path from q' to q''. The undirected graph $G_T = (V_T, E_T)$ is an induced subgraph of the roadmap which is defined by partitioning G into a set of trees $T_1, \ldots, T_K$ and contracting them into the vertices of G_T. In other words, $V_T = \{T_1, \ldots, T_K\}$ and $(T_i, T_j) \in E_T$ if there exist configurations $q_i \in T_i$ and $q_j \in T_j$ are connected by a local path. As shown in Table I, the roadmap construction proceeds in three stages: milestone computation (lines 1–5), edge selection (lines 6–9), and edge computation (lines 10–12).

A. Milestone Computation

In SRT, the trees of the roadmap G_T are computed by sampling their roots uniformly at random in $\mathcal{C}_{\text{free}}$ and then using a sampling-based tree planner to explore the region around the root configuration. If the initial, q_{init}, and the goal, q_{goal}, configurations of a query are known in advance, they should be used as roots to make SRT even more efficient. Each tree T is incrementally extended, where at each iteration a new random configuration, q_{rand}, is generated and a local planner, e.g., straight-line planner, attempts to connect some configuration $q \in T$ to q_{rand}. If the local planner succeeds, then the configuration q_{rand} and the edge (q, q_{rand}) are added to T (see [25], [40]).

B. Edge Selection

Each tree T defines a representative configuration q_T which is computed as an aggregate of the configurations in T. Our implementation uses the centroid. If $Q = \{q_{T_1}, \ldots, q_{T_K}\}$ is the set of representatives, then for each $q_{T_i} \in Q$, we determine n_{c} closest and n_{r} random representatives q_{T_j} and add each (T_i, T_j) to the graph of candidate edges $G_C = (V_T, E_C)$. A distance metric defines closeness and the closest neighbors are found using kd-trees [17]. Random neighbors are used to offset any problems with the distance metric.

C. Connected Component Heuristic

The objective of our planner is to determine the existence of a path. To this end, we avoid computing candidate edges that would create cycles. Since a query would never succeed due to an edge that is part of a cycle, it is indeed sensible not to consume time and space computing and storing such edges. In some cases, however, the absence of cycles may lead the query phase to construct unnecessarily long paths. This drawback can be mitigated by applying post-processing techniques, such as smoothing, on the resulting path.

D. Edge Computation

Candidate edges are computed by a sampling-based tree planner. For each candidate edge (T_i, T_j), n_{p} close pairs of configurations of T_i and T_j are quickly checked with a fast local planner. Equally spaced points along the straight line between two configurations are tested for collision using bisection, which increases the chances of quick rejection of paths in collision [20]. If any local path is found, no further computation takes place. Otherwise, a more complex tree-connection algorithm is executed, e.g., bi-directional `RRT` or `EST`. During the tree connection, additional configurations are typically added to the trees T_i and T_j. If the tree-planner is successful, the edge (T_i, T_j) is added to E_T and the graph components to which T_i and T_j belong are merged into one.

E. Queries

Queries are solved by connecting q_{init} and q_{goal} to the roadmap and proceeding by graph search. Two trees, T_{init} and T_{goal} rooted at q_{init} and q_{goal}, respectively, are grown for few iterations and added to the roadmap. Neighbors of T_{init} and T_{goal}, denoted S_{init} and S_{goal}, respectively, are computed as a union of n_{c} closest and n_{r} random milestones. The tree-connection algorithm alternates between attempts to connect T_{init} to each milestone in S_{init} and T_{goal} to each milestone in S_{goal}. A path is found if at any point T_{init} and T_{goal} lie on the same connected component of the roadmap. The quality of the path is improved by applying path smoothing.

F. Parameters

`SRT` has several parameters which we now summarize: K, the number of milestones used in the construction of the roadmap; m, the number of configurations used in the generation of a milestone; n_{c}, the number of closest neighbor milestones; n_{r}, the number of random neighbor milestones; n_{p}, the number of close pairs to check with a straight-line planner before running the tree-connection algorithm; n_{i}, the number of iterations to run the tree-connection algorithm.

A nice feature of `SRT` is that by setting these parameters differently, `SRT` can behave exactly as `PRM`, `RRT`, or `EST` as illustrated in Table II.

TABLE II
OTHER PLANNERS AS INSTANCES OF `SRT`.

	K	m	n_{c}	n_{r}	n_{p}	n_{i}
`PRM`	any	1	any	any	1	0
`RRT`	0	0	1	0	0	any
`EST`	0	0	1	0	0	any

III. PARALLEL `SRT` PLANNER

High-dimensional problems arising from complex robotic systems test the limits of current motion planners and require the development of efficient parallelized motion planners that take full advantage of all the available resources. Despite the need for fast solutions for such problems, little work exists on parallel motion planners, especially when contrasted with the work on sequential motion planners. In [43], a parallel algorithm for 6-DOFs manipulators is developed based on the property that the configuration space obstacle for a union of objects is the union of the configuration space obstacle of the individual objects. In [15], [16], a parallel version of the randomized path planner [8] is proposed that uses the OR paradigm, i.e., different processors compute the same algorithm and as soon as a solution is found, the computation stops. The work in [22] discretizes and then decomposes the configuration space into hypercubes and cyclically assigns the exploration of the hypercubes to the available processors. The method is impracticable for high-dimensional problems due to the discretization of the configuration space. The works in [5] and [14] focus on embarrassingly parallel algorithms for `PRM` and `RRT`, respectively, which avoid any interprocess communication and in the context of `PRM` and `RRT` are limited to memory-shared systems.

In this section, we describe the design and implementation of a parallel version of `SRT` designed for solving very high-dimensional problems that exceed the resources available to single machines and that cannot be efficiently addressed with existing planners. The efficiency of our parallel planner stems from its hierarchical structure that allows the division of computation into large blocks. Our algorithm can be applied to memory-shared or message-passing systems. The description of the algorithm is in terms of message-passing systems.

A. Data and Control Flow Dependencies

Before relating the details, we discuss data and control flow dependency in each stage of the `SRT` algorithm. During milestone computation, there are no dependencies. Each milestone can be processed in parallel. Additional parallelization is stymied by the sampling scheme we use to generate milestones and would be considerably more involved. Random edge

selection can be done in parallel; however, the distribution of the closest edge selection is more difficult since it requires the construction of a search structure that depends on the representatives of the milestones. Finally, edge computations are not entirely independent of each-other. Since milestones can change after an edge computation as a result of adding new configurations to the milestones and since computing an edge requires direct knowledge of both milestones, the edge computations cannot be efficiently parallelized without some effort. Furthermore, computation pruning due to component analysis (see section II-C) entails control flow dependencies throughout the computation of the edges. Our experiments with the sequential implementation revealed that the bulk of the run time occurs in milestone and edge computation.

B. Computation of Parallel SRT

We have chosen a scheduler–processor architecture for our parallel implementation. The processors are responsible for milestone and edge computations. The scheduler arbitrates milestone ownership, handles edge selection, assigns edge candidates to processors, and manages the connected component data structure. Parallel SRT is described in Table III.

1) Milestone Computation: The milestone computation is described in Table III under COMPUTE MILESTONES. Each processor P_i computes a set T_{P_i} of milestones and sends to the scheduler their representatives until K milestones have been computed. The set T_{P_i} is owned by P_i and stored locally in P_i, while the set of the representatives is stored in the scheduler. The communication during this stage is limited, non-blocking, and occurs only between the scheduler and the processors. The scheduler maintains a map σ, such that $\sigma(k) = P_i$, if the k-th milestone representative is sent to the scheduler by P_i.

2) Edge Selection: The scheduler computes the graph $G_C = (V_T, E_C)$ of candidate edges as described in the sequential case Table I. There is no parallelization of this stage since it is only 1–2% of the total computation time and requires complex search structures. Each set T_{P_i} induces a subgraph $L_{P_i} = (T_{P_i}, E_{P_i})$ of G_C referred to as the local graph. Since each milestone of L_{P_i} is owned by and stored locally in P_i, computation of the edges of L_{P_i} requires no communication.

3) Edge Computation: The edge computation is described in Table III under COMPUTE EDGES. For each processor P_i, the scheduler selects an edge $e_i = (T', T'')$ uniformly at random from L_{P_i}, deletes e_i from G_C and L_{P_i}, and assigns the computation of e_i to P_i. The scheduler sends to P_i the indices of the two milestones T' and T''. In response, P_i runs the tree-connection algorithm on T' and T'' and if the connection is successful, it sends to the scheduler the indices of two configurations $q' \in T'$ and $q'' \in T''$ that are connected by a local path. In that case, the scheduler adds e_i to G_T and all edges $(T_i, T_j) \in G_C$ such that T_i and T_j lie in the same connected component of G_T are deleted from G_C as they will not change the connected component structure of G_T. The above steps are repeated until there are no more edges in G_C. At each step, certain L_{P_i}'s may be empty due to edge deletions and cause some of the processors, say $P_1, \ldots, P_\ell$, to become idle. Our implementation handles this situation by repartitioning the milestones owned by these processors.

4) Partition Computation: The partitioning is described in Table III under COMPUTE PARTITIONS. Given the graph G_C, the problem of finding "good" partitions is formulated as an optimization problem: determine a partition $T_{P_1}, \ldots, T_{P_\ell}$ of the milestones that maximizes $\sum_{i=1}^{\ell} |E_C \cap E_{P_i}|$. This is an instance of the graph partition into ℓ parts problem which is known to be NP-hard for $\ell \geq 2$. Graph partition problems, however, arise in many computational tasks, notably in distributed computing, and effective heuristic approaches have been found [44]. We use the classical Kernighan-Lin algorithm [33] which is a greedy local optimization approach. Once the partitions are computed, they must be assigned to the processors in such a way that the number of milestones that need to be exchanged is minimized. This is an instance of the maximum bipartite matching problem and can be solved efficiently with the Hungarian algorithm [48].

The scheduler recomputes the map σ so that it reflects the changes due to partitioning and the assignment operation. Denote the updated map by σ', i.e., $\sigma'(k) = P_i$ if the k-th milestone is assigned to P_i. Then the scheduler sends the updated map to all the processors involved in the partitioning. Processors are now responsible for implementing the partitioning and the assignment operation. In particular, P_i needs to send the k-th milestone to P_j if $\sigma(k) = P_i$ and $\sigma'(k) = P_j$, and P_i needs to receive the k-th milestone from P_j if $\sigma(k) = P_j$ and $\sigma'(k) = P_i$. The communication between P_i and P_j is non-blocking. Each processor posts the requests for its send and receive operations and continues immediately with edge computations. The communication becomes blocking only if computing a particular edge $e = (T', T'')$ requires the completion of one or two receive operations on T' or T''.

IV. EXPERIMENTS AND RESULTS

The experiments in this paper were chosen for two purposes: to test SRT on problems that cannot be efficiently solved by PRM and single-query planners and to evaluate the performance of the parallel SRT compared to the sequential implementation. In this way, we hope to gain insight on the difficult and largely unsolved problem of sampling schemes for motion planning. It must be noted that the authors made an effort to choose difficult benchmarks and representative problems given the limited amount of space that can be devoted to experimental setup. This task is particularly difficult due to the absence of benchmark sets for path planning.

A. Benchmarks

We ran our experiments on a set of benchmarks chosen to vary in type and in difficulty. An illustration of our benchmarks can be found in Fig. 1 and Fig. 2.

Problem "narrow4h2" consists of four non-convex parts (robots) that must wiggle their way through two small holes as they exchange places from one side of a wall (obstacle) to the other side of a second wall (obstacle), as shown in Fig. 1. This benchmark tests how SRT handles narrow-passage problems, which are known to be difficult for RRT, EST, and PRM.

Problems "narrow6" and "narrow8" are similar to "narrow4h2", except they have six and eight non-convex parts

TABLE III
PARALLEL SAMPLING-BASED ROADMAP OF TREES (SRT) ALGORITHM.

Scheduler		**Processor** P_i
1: INVOKE COMPUTE MILESTONES. 2: INVOKE COMPUTE EDGES.	PARALLEL SRT	1: INVOKE COMPUTE MILESTONES. 2: INVOKE COMPUTE EDGES.
1: $Q \leftarrow \emptyset$, $i \leftarrow 0$. 2: **while** $i < K$ **do** 3: Wait for some q_T to arrive. 4: $Q \leftarrow Q \cup \{q_T\}$; $i \leftarrow i + 1$. 5: Broadcast `finish` to processors.	COMPUTE MILESTONES	1: $T_{P_i} \leftarrow \emptyset$. 2: Post request for message from scheduler. 3: **while** `finish` has not been received **do** 4: $T \leftarrow$ generate a milestone; $T_{P_i} \leftarrow T_{P_i} \cup \{T\}$. 5: Send representative q_T to the scheduler.
1: $G_C = (V_T, E_C) \leftarrow$ graph of candidate edges. 2: $L_{P_i} = (V_i, E_i) \leftarrow$ local graph, for all P_i. 3: $W = \{P_1, \ldots, P_n\}$. 4: **while** unprocessed edges remain in G_C **do** 5: INVOKE COMPUTE PARTITIONS. 6: **for** i: $P_i \in W$ **and** $\lvert E_i \rvert > 0$ **do** 7: $e_i \leftarrow$ randomly selected from E_i; send e_i to P_i 8: $E_i \leftarrow E_i - \{e_i\}$; $W \leftarrow W - \{P_i\}$. 9: **if** computed edges have arrived **then** 10: $W \leftarrow W \cup \{P_i : P_i$ computed $e_i\}$. 11: Update connected components, G_C and L_{P_i}'s. 12: Broadcast `finish` to processors.	COMPUTE EDGES	1: Post request for message from scheduler. 2: **while** `finish` has not been received **do** 3: **while** no message has been received **do** 4: Complete a pending send operation. 5: Complete a pending receive operation. 6: **if** partition message has been received **then** 7: INVOKE COMPUTE PARTITIONS. 8: **if** $e_i = (v_1, v_2)$ has been received **then** 9: Complete pending receives (if any) on T_{v_1}, T_{v_2}. 10: Try to connect T_{v_1} and T_{v_2}. 11: Send result to scheduler. 12: Post request for message from scheduler.
1: $S = \{P_i : E_i = \emptyset\}$. 2: Compute $G'_C = (V_S, E_S)$, where $V_S = \bigcup_{P \in S} V_P$ 3: and $E_S = \{(v_1, v_2) \in E : v_1, v_2 \in V_S\}$. 4: Partition G'_C into L_{P_i}'s for $P_i \in S$. 5: **for** i: $P_i \in S$ **do** 6: $\sigma(v) \leftarrow P_i$ for all $v \in V_{P_i}$. 7: Send σ to P_i for all $P_i \in S$.	COMPUTE PARTITIONS	1: Complete all pending send/receive operations. 2: Receive σ from server. 3: **for** $i = 1$ to K **do** 4: **if** $T_i \in T_{P_i}$ **and** $P_i \neq \sigma(i)$ **then** 5: Post request to send T_i to $\sigma(i)$. 6: **if** $T_i \notin T_{P_i}$ **and** $P_i = \sigma(i)$ **then** 7: Post request to receive T_i from $\sigma(i)$.

(robots), respectively, and a single wall (obstacle) with a small square hole in it. These benchmarks test how efficiently motion planners solve high-dimensional narrow-passage problems.

Problem "pentomino" [2] consists of twelve pieces (robots) filling a $3 \times 4 \times 5$ box, as shown in Fig. 2(a). The objective is to disassemble the initial configuration by moving each piece an arbitrary distance away from all the other pieces. This benchmark tests the efficiency of motion planners in solving high-dimensional problems in uncluttered environments.

Problems "tunnel1" and "tunnel2" consist of one and two non-convex parts (robots), respectively, that must go through a long and narrow tunnel (obstacle), as shown in Fig. 2(b). These benchmarks further test how efficiently motion planners solve narrow-passage problems.

Problems "fence1", "fence2", and "fence4" consist of one, two, and four non-convex parts (robots), respectively, placed in a box split by a regular fence-like wall (obstacle), as shown in Fig. 2(c). Each robot must go from one side of the fence to the opposite side. These benchmarks test how efficiently `SRT` solves problems which its building blocks cannot solve efficiently. Using benchmarks with different number of robots also tests the efficiency of the motion planner as the number of DOFs increases.

Problem "combo1" consists of a single non-convex part (robot) that must go through a fence (obstacle) and then through a long and narrow tunnel (obstacle), as shown in Fig. 2(d). This benchmark combines the "fence" and "tunnel" benchmarks in order to test the efficiency of `SRT` in solving problems which are known to be difficult for `PRM`, `RRT`, and `EST`. The holes in the fence have different dimensions to further highlight the opportunistic nature of `RRT` and `EST` and increase their likelihood of failure by making some of the holes so small that the robot cannot wiggle its way through. The tunnel is also narrow, barely allowing the robot to move through it, making it difficult for `PRM` with uniform sampling to generate many configurations inside the tunnel.

Problem "rooms1" consists of four rooms forming a 2×2 grid, as shown in Fig. 2(e). Every pair of rooms is separated by fences (obstacles), except the first and the fourth ones, which are separated by a wall (obstacle). The objective is to move one non-convex part (robot) from the first room to the fourth room. Problem "rooms2" is similar except it has two robots. As the "fence" benchmarks, the "rooms" benchmarks further test the efficiency of `SRT` in solving problems which its building blocks cannot solve easily.

Problems "random4" and "random-chain" consist of four non-convex parts (robots) and a 12-DOFs articulated arm (robot), respectively, in a box filled with random polyhedral objects (obstacles), as shown in Fig. 2(f). These benchmarks test how efficient motion planners are in solving problems with

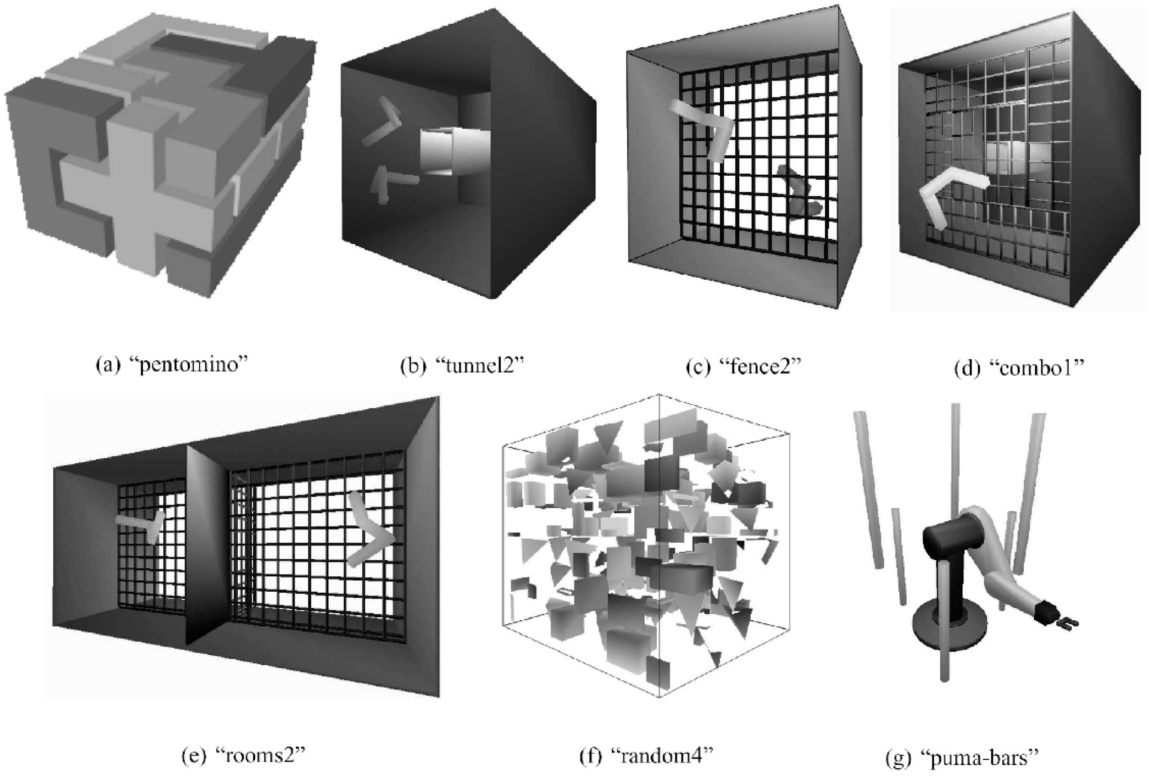

Fig. 2. Path planning problems.

cluttered environments and a high number of collision checks.

Problem "puma-bars" consists of a 6R articulated limb (robot) similar to a Puma560 surrounded by several vertical bars (obstacles), as shown in Fig. 2(g). The movements of the puma robot are severely constrained making it difficult to solve any queries. This benchmark tests how well SRT distributes its computation for problems that require long computation times.

B. Motion Planning for Multiple Robots

In our experiments, we considered multiple non-convex, rigid bodies and multiple open kinematic chains operating in a three dimensional workspace with rigid, non-convex static obstacles. As with other motion planners, generalizations to many other kinematic planning problems are straightforward and the use of RRTs and ESTs as subroutines gives a natural way to extend SRT to kinodynamic planning instances [18], [25], [40]. We applied several optimizations to the multi-robot case for rigid bodies and open kinematic chains that involved heuristic replanning of robots in collision [1], [9].

C. Hardware and Software Setup

The implementation was carried out in ANSI C/C++ using the GNU compilers and libraries. Additionally, we made use of the SWIFT++ collision detection library [19], the MPICH/MPI for communication and OpenGL for visualization. The processing nodes consisted of eleven dual AMD Athlon 1900MPs with one gigabyte of memory each. The scheduler node was an AMD Athlon 1800XP with 500 megabytes of memory. The network topology was switched 100Mbps for the processing nodes with a 1Gbps backbone to the scheduler node. All of the nodes ran Debian Linux with kernel 2.4.21.

D. Sampling Methods

In our experiments, we used sampling-based motion planners. There is extensive research on how sampling should be done, especially in the context of PRM. In the original formulation of PRM [32], uniform random sampling is employed. In recent years, other sampling methods have been developed that attempt to improve the performance of the motion planners [3], [12], [13], [21], [23], [24], [27], [55], especially in the presence of narrow passages. We expect that any sampling method that improves PRM, RRT, or EST, would also improve SRT, since PRM, RRT, and EST are the building blocks of SRT. Furthermore, SRT retains the global sampling property of PRM and uses it to offer to RRT and EST more opportunities to explore different parts of the configuration space.

A comprehensive testing with all available sampling methods is just not possible, given the space restrictions of this paper. In our experiments, we compare the performance of SRT to PRM, RRT, and EST using several sampling methods. The results of our experiments indicate that any sampling method that improves the performance of the building blocks of SRT, also improves the performance of the overall SRT.

In addition to the uniform random sampling [32], we present experiments where gaussian [12], bridge [27], and two variants of obstacle-based [3] sampling, denoted obstacle1 and obstacle2, are used. A summarized description of many of these methods can be found in [12].

TABLE IV
PARAMETER SETTING FOR SRT.

	K	m	n_c	n_r	n_p	n_i
cat1	400-500	10-50	7-10	4-6	15-25	25-35
cat2	750-1000	50-100	12-15	6-9	25-35	40-70
cat3	1500-2000	100-150	15-18	8-10	50-60	100-125

E. Parameter Setting

Through extensive experimentation on many different benchmarks, we have found three different categories of parameter values that yield good performance for SRT on *cat1*, *cat2* and *cat3* benchmarks, as indicated in Table IV. The performance of SRT on these categories depends more on the number of milestones, K, and the number of configurations per milestone, m, than it does on the values of the other parameters. That is, for the parameters n_c, n_r, n_p and n_i, the values in the *cat3* category can be used to solve the problems in the *cat1* and *cat2* categories with only a negligible increase in the running time of SRT (see Section II-F for a summary of SRT parameters). There is clearly a tradeoff between K and m. The larger m is, the easier it is for tree-connections to succeed. On the other hand, generating many configurations per milestone requires computation time that could be more useful if spent in tree-connections. Since SRT provides a smooth spectrum between PRM and RRT or EST, larger values for K and smaller values for m make SRT behave more like PRM and less like RRT or EST, and vice-versa. Our experiments indicate that SRT should use more milestones and a smaller number of configurations per milestone for problems which PRM is better suited and fewer milestones and a larger number of configurations per milestone for problems which RRT or EST are the better choice.

The results of our experiments are for good parameter selections for each method. We ran each benchmark several times with an initial guess for the parameters. Then based on the results of the experiments, we modified our guesses until the particular motion planner was able to solve the benchmark consistently in as little time as possible. In a new example, selecting the optimal parameters can be difficult, particularly for SRT, because of the number of the parameters and the relation between K and m. In general, generating no more than several hundred milestones and using milestones with several dozens of configurations seemed to be the best setting. For the experiments of Table V, we used the following values.

1) SRT *parameters:* For all our experiments with SRT, we set $n_c = 15$, $n_r = 8$, and $n_p = 20$. The values of the other parameters varied. Our choices were guided by Table IV.

The "pentomino" benchmark is in the *cat1* category, since it has no obstacles; we set $K = 400$, $m = 20$, and $n_i = 30$.

The "fence1", "fence2", "fence4", "narrow4h2", "narrow6", and "narrow8" benchmarks belong to the *cat3* category because of the very small dimensions of the openings making it almost impossible for the robots to wiggle their way through. In addition, many of these benchmarks have several robots. For these benchmarks, we set $K = 2000$, $m = 100$, and $n_i = 100$.

The "tunnel1" benchmark fits into the *cat2* category since the openings in the tunnel are slightly larger than those in the "fence" and "narrow" benchmarks. We set $K = 1000$, $m = 50$, and $n_i = 70$.

The "tunnel2" benchmark is harder because it has two robots instead of one, and thus, considered between *cat2* and *cat3* categories. A similar classification holds for "rooms1", "rooms2", and "combo1" benchmarks since the openings in the fences and the tunnels are slightly larger than those in the "fence" and "narrow" benchmarks. For these benchmarks, we set $K = 1600$, $m = 50$, and $n_i = 70$.

2) PRM *parameters:* The performance of PRM is determined by two parameters, K and n_c. For the "fence1", "fence2", "narrow4h2" benchmarks, we set $K = 150000$ and $n_c = 125$. For the "tunnel1" benchmark with uniform sampling, we set $K = 50000$ and $n_c = 75$, and for the non-uniform sampling cases, we set $K = 35000$ and $n_c = 75$. For the benchmarks in Table V(c), we set $K = 60000$ and $n_c = 100$. For the other benchmarks of Table V, we could not find parameters that would solve the problem in less than 5 hours.

3) RRT *and* EST *parameters:* RRT and EST iterate over the connection strategy until the two trees are connected or a predefined number of iterations has been reached. For this reason, we set $n_i = 500000$ to allow RRT and EST to iterate as much as it was needed to find a solution.

F. Comparison of SRT with Other Planners

SRT can be made into PRM, RRT, or EST by setting its parameters as in Table II. We tested the performance of all these motion planners on the benchmarks of section IV-A. Table V contains a summary of our results. In each case, we report the running time in seconds, averaged over sixteen runs.

In Table V(a), we compare the performance of SRT to other planners when uniform sampling is used. Our experiments showed that PRM, RRT, or EST could not solve the "fence2", "narrow6", or "narrow8" problems even after 5 hours of computation, while SRT solved these problems in 867.6s, 2935.2s, and 7270.2s on average, respectively. For the benchmarks "fence1", "fence2", "narrow4h2", we observe a significant reduction in the running time of SRT versus PRM, RRT, and EST. The running time improvement is less significant in the case of the "pentomino" benchmark due to the short time needed to solve this puzzle. As Table V(a) indicates, PRM with basic uniform random sampling is generally slow since it requires a considerable amount of time to preprocess the configuration space. Importantly, for the cost of two or three bi-directional RRT or EST queries, SRT can preprocess the configuration space to obtain a structure that answers queries more robustly and more quickly than the corresponding sampling-based tree planners. The differences between the methods were more pronounced in the examples with more complex scenes and with more robots. We use fairly standard implementations of PRM, EST, and RRT. We think it is likely that improvements to either subroutine would be an improvement to SRT.

In Table V(b, c), we compare the performance of SRT to PRM, RRT, and EST using several sampling methods aimed at improving the performance of the building blocks of SRT in scenes with narrow passages. The purpose of these experiments is to show that improvements in the sampling employed by PRM, RRT, or EST also improve the performance of SRT.

TABLE V
COMPARING SRT TO PRM, RRT, AND EST.

benchmark	PRM	RRT	EST	SRT[RRT]	SRT[EST]
fence1	5638.80s	7209.00s	X	**114.60s**	351.00s
fence2	14007.00s	X	X	**868.18s**	872.78s
fence4	X	X	X	3307.40s	**3158.51s**
random4	X	6133.80s	X	2242.39s	**1577.97s**
narrow4h2	14809.20s	9045.00s	X	1666.95s	**1290.25s**
narrow6	X	X	X	3131.71s	**2935.10s**
narrow8	X	X	X	**7270.20s**	7525.80s
pentomino	X	168.48s	47.38s	58.29s	**19.89s**

(a) Summary of the results when uniform random sampling is used.

sampling	PRM	RRT	EST	SRT[RRT]	SRT[EST]
uniform	463.83s	X	X	89.35s	**85.31s**
gaussian	409.22s	X	X	**51.45s**	77.41s
bridge	377.86s	X	X	**38.54s**	84.27s
obstacle1	335.67s	X	X	**38.84s**	43.78s
obstacle2	334.75s	X	X	44.27s	**36.30s**

(b) Summary of the results for the "tunnel1" benchmark when several sampling methods are used.

benchmark	PRM		RRT	EST	SRT[RRT]		SRT[EST]	
tunnel2	2697.67s	obstacle2	X	X	**262.75s**	obstacle2	290.96s	obstacle2
rooms1	2391.54s	obstacle2	X	X	193.00s	obstacle2	**149.17s**	obstacle2
rooms2	4448.97s	obstacle1	X	X	542.96s	obstacle1	**406.77s**	obstacle1
combo1	1136.12s	obstacle1	X	X	**136.59s**	obstacle1	190.89s	obstacle1

(c) Summary of the results for many benchmarks when several sampling methods are used. Only the best average running time and the sampling method used to obtain it are indicated.

Entries for RRT and EST show average time per query. Entries for PRM, SRT[RRT] and SRT[EST] show average time to build the roadmap and then solve ten random queries; average time per query is not shown separately since it is less than 0.1s. Entries marked with X show that the problem could not be solved even after 5 hours of computation. Entries in bold show the best running time across the row. Each running time is obtained as an average of sixteen runs.

Table V(b) contains the results of our experiments for the "tunnel1" benchmark. RRT and EST were not able to solve the queries even after 5 hours of computation. We believe this is due to the small dimensions of the tunnel that barely allow the robot to move making it hard for RRT and EST to connect queries on the opposite sides of the tunnel. Such connections require RRT and EST to first gear the exploration from query configurations towards the openings of the tunnel and then progress inside the tunnel. This is difficult since in their opportunistic approaches RRT and EST spend most of the time trying to connect configurations on the opposite sides of the dividing wall, which does not have any openings. On the other hand, PRM is able to solve this problem even when uniform sampling is used due to the large number of samples it generates making it possible to connect some configurations inside the tunnel. The large number of samples needed, however, takes its toll on the running time of PRM. The performance of PRM can be improved by using different sampling methods, especially obstacle-based methods. As shown in Table V(b), the average running time of PRM is reduced from 463.83s to 334.75s when uniform and obstacle-based sampling are used, respectively. SRT, which retains the global sampling property of PRM, is able to offer to RRT and EST easier tree connections which could have some of their configurations inside the tunnel or near the openings of the tunnel. In this way, SRT is able to perform better than PRM, RRT, and EST. Furthermore, as our experiments indicate, improvements to the sampling done by PRM carry over to SRT. As an example, the average running time of SRT[RRT] is reduced from 89.35s to 38.84s and the running time of SRT[EST] is reduced from 85.31s to 36.30s when uniform and obstacle-based sampling are used, respectively. Similar results were obtained for other benchmarks, as summarized in Table V(c).

In Table V(c), we summarize the results of our experiments for the "tunnel2", "rooms1", "rooms2", and "combo1" benchmarks. In addition to the average running time, we also indicate the sampling method, i.e., uniform, gaussian, bridge, obstacle1, or obstacle2, that was most effective. Similar to the performance on the "tunnel1" benchmark, RRT and EST did not solve any of the benchmarks in less than 5 hours despite the sampling method used. The performance of PRM was improved by the non-uniform sampling methods, and as in the "tunnel1" benchmark, the improvements were more significant when the obstacle-based sampling methods were used. As indicated in Table V(c), these improvements carried over to SRT reducing its running time to a fraction of the running time of PRM.

G. Discussion and Insights on Sampling Methods

It is well known that PRM, RRT and EST are sensitive to the interplay between the distance metric and the incremental planner [4], [39]. We also made this observation in our experiments. In environments with thin features, in particular

the "fence", "rooms", and "combo" environments, RRT and EST tended to produce many configurations that were stuck near the obstacles. As both RRT and EST are opportunistic, it is difficult for these methods to abandon the current region of the configuration space and explore other regions that might lead to successful connections. As part of SRT, RRT and EST may still get stuck on particular regions when connecting two trees. However, these connections run only for few iterations and immediately after, RRT and EST explore new regions as they try to connect other trees. Thus, SRT allows RRT and EST to quickly explore many different regions significantly improving the likelihood for successful connections. In environments with a single narrow feature, RRT and EST are forced to do a similar amount of work to answer a single query to the preprocessing phases of PRM or SRT. This phenomena also accounts for the better performance of RRT on the random example compared to other examples where solving a query can often be done without considering the whole configuration space. Finally, we believe that the efficiency of SRT derives in part from offering the tree-based planners easier queries as they come from the nearest neighbor clustering and also from the fact that RRT and EST make use of locality and are capable of answering easier queries while avoiding difficult and irrelevant parts of the configuration space.

Our planner has several opportunities for early exit. In collision detection, bisection checking on a path allows for early exit for paths with many collisions. In the PRM layer, only checking edges between different components improved the running time. Also, the tree-based planners can make an early exit by quickly checking n_{p} close pairs for connection with a straight-line planner before running bi-directional RRT or EST. Since SRT uses all of these exit opportunities, the time improvements are most significant.

In some of our experiments, the running time of SRT was far superior to PRM. This occurs for several reasons. SRT checks fewer edges but works harder for each edge. Also, the nearest neighbor queries lead to super-linear growth in the running time. On more difficult examples, PRM needs many milestones to succeed and the kd-tree has many points in it. As the number of points in the tree grows, this cost begins to dominate the running time since it is the only super-linear cost in the implementation. The hierarchical representation of SRT yields much smaller trees and this problem does not manifest as seriously.

Our experiments confirmed an observation that has been made earlier about PRM. Uniform random sampling is very easy to implement and in many cases the simplest way to solve a path planning problem, but it is not always the most efficient. Our experiments also indicate that improvements in the sampling methods used by PRM, RRT, or EST improve the performance of SRT. By exploiting efficient sampling schemes, such as EST and RRT, a planner with better performance is obtained. Other combinations of sampling methods can be used, but a comprehensive testing of all combinations is not possible due to time and space limitations.

We also note that in the reported experiments q_{init} and q_{goal} were not used as milestone roots during the roadmap construction to make the problem even more difficult for SRT.

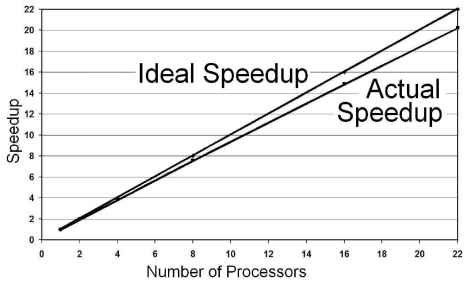

Fig. 3. Speedup of parallel SRT for the "fence2" benchmark with RRT as the local planner of SRT. Similar speedups are obtained for the other benchmarks.

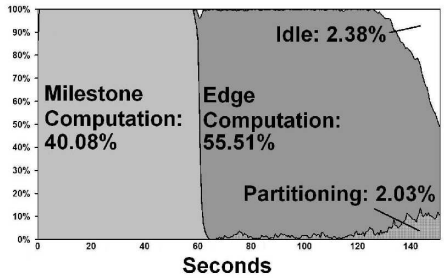

Fig. 4. Distribution of computation of parallel SRT for the "fence4" benchmark with RRT as the local planner of SRT running on 22 processors. Similar distributions of computation are obtained for the other benchmarks.

H. Measuring Parallel Efficiency

To measure the parallel efficiency of SRT, we ran on various benchmarks the parallel code with 1, 2, 4, 8, 16 and 22 processors - the maximum number of processors we had available. Run times are averaged over sixteen runs. In Table VI, we report results for SRT with RRT and EST as its local planners. In each case, we report time with 1 and 22 processors (time[1] and time[22]). Also, for the parallel runs, we report the percentage of time spent in milestone computation (mc), edge computation, (ec), communication (comm), waiting (idle), and parallel efficiency (eff), which is calculated by $t_s/(t_f \cdot N)$, where t_s is sequential time, t_p is parallel time, and N is the number of processors.

The plot in Fig. 3 is for "fence2" and indicates the speedup obtained for different numbers of processors. The plot in Fig. 4 is for "fence4" and presents logged data showing how processors spend their time. These plots are characteristic of the behavior of the algorithm on the other benchmarks as well.

The overall efficiency of the parallel SRT is reasonably high on average, 88.8%, and in all our experiments in the range $67-99\%$. We also had a benchmark where superlinear, 1.12%, speedup was obtained. The speedup graph in Fig. 3 is almost linear which suggests that the efficiency constant is not decaying with the number of processors. However, the parallel SRT places a load on the scheduler which is proportional to the number of processors. As the number of processors increases, this will eventually become a problem. A possible solution might be to increase the number of schedulers or to have a hierarchy of schedulers. This is left as future work [50].

Nevertheless, there are several advantages of the parallel SRT. It is fairly simple and makes little use of any blocking

TABLE VI
PARALLEL SRT VERSUS SEQUENTIAL SRT.

	time[1]		time[22]		mc		ec		comm		idle		eff	
benchmark	a	b	a	b	a	b	a	b	a	b	a	b	a	b
fence2	868.18	872.78	42.82	41.54	41.08	27.76	45.39	63.11	9.65	6.57	3.88	2.56	0.92	0.95
fence4	3307.40	3158.51	151.84	149.23	40.08	23.65	55.51	70.40	2.03	4.49	2.38	3.41	0.99	0.96
narrow4h2	1666.95	1290.25	93.21	79.51	39.02	27.15	50.28	58.13	6.18	9.36	4.52	5.36	0.81	0.74
narrow6	3131.71	2935.10	173.41	176.15	45.00	26.05	45.09	65.33	6.92	5.83	2.99	2.79	0.82	0.76
random4	2242.39	1577.97	125.56	107.83	30.36	13.17	64.12	78.97	3.91	4.88	1.61	2.98	0.81	0.67
random-chain	10050.48	10691.09	512.28	551.93	23.30	21.86	72.19	74.29	2.21	1.55	2.30	2.30	0.89	0.88
puma-bars	8097.04	10207.89	327.32	414.10	2.48	2.06	87.60	89.39	8.44	7.64	1.48	0.91	1.12	1.12

Columns (a) and (b) refer to data for SRT[RRT] and SRT[EST], respectively. Columns (time[1]) and (time[22]) show the running time in seconds of the sequential SRT and the parallel SRT with 22 processors, respectively. Columns (mc), (ec), (comm), and (idle) show the percentage of the running time of the parallel SRT spent in milestone computation, edge computation, communication, and idle, respectively. Column (eff) shows the efficiency of the parallel SRT.

communication. Milestone and edge computations are also nearly fully distributed and storage is also distributed evenly.

Virtually all of the communication overhead occurs during the edge computations. This stage would be the most reasonable place to attempt to make further improvements. The graph partition scheme we used in our implementation optimized the sum of the number of edges in the L_{P_i}'s. A better quantity to optimize would be to maximize the minimum number of edges over all L_{P_i}'s. This would favor better load balancing.

V. DISCUSSION

We observed in our experiments that SRT is a powerful planner which combines advantages of traditional sampling-based single query and multiple query planners. By varying SRT's parameters, a smooth spectrum between single query planners and PRM can be obtained from our implementation, as discussed in section IV-E. The sampling done in SRT has common attributes with earlier refinement and non-uniform sampling techniques used in PRM [32]. We believe that the efficiency of SRT derives (1) from offering the sampling-based tree planner easier queries as they come from the closest neighbor clustering and (2) the fact that the global sampling property of PRM is retained so that efficient sampling-based expansion heuristics, such as RRT and EST, do not get trapped.

Our planner was effective for high-dimensional problems, which were constructed by putting multiple non-convex rigid robots in various scenes. In many cases the advantages of SRT were striking. SRT is designed primarily as a multiple-query planner, but it should be noted that it is most effective for difficult planning problems. For such problems, distinction between single versus multiple query methods is not clear. If the initial and the goal configurations are known in advance, they should be used as roots of the milestones during the roadmap construction stage to make SRT even more efficient for single-query planning. Also, note that in difficult examples, we obtained a roadmap of the configuration space for a fraction of the cost of solving a single query by a sampling-based tree planner, such as RRT and EST.

The efficiency of SRT is not limited to the specific single query planners that we used. We observed that SRT exhibits similar behavior no matter whether RRT or EST is being used as its local planner. In fact, other sampling-based tree planners with good coverage properties can be substituted.

Furthermore, we suggested a parallel implementation of SRT and obtained an efficient division of labor allowing SRT to tackle problems of unprecedented complexity. We plan to scale our SRT implementation to a cluster with several hundred nodes. To do this, it is likely that some decentralization of the scheduling computations will become necessary [50]. Our goal is to apply our work to increasingly hard planning problems dealing with flexible robots [35], [37], [45], reconfigurable robots [56], manipulation planning [51], complex planning instances [52], and computational biology applications [6], [7].

ACKNOWLEDGMENT

The authors would like to thank all the reviewers and members of the Physical and Biological Computing group at Rice University for their helpful comments and discussions.

REFERENCES

[1] M. Akinc, K. E. Bekris, B. Y. Chen, A. M. Ladd, E. Plaku, and L. E. Kavraki, "Probabilistic roadmaps of trees for parallel computation of multiple query roadmaps," in *Interantional Symposium on Robotics Research*, ser. Springer Tracts in Advanced Robotics (STAR), D. Paolo and R. Chatila, Eds. Springer Verlag, 2003.

[2] N. M. Amato. Motion planning benchmarks. Texas A&M University. [Online]. Available: http://parasol-www.cs.tamu.edu/dsmft/benchmarks/

[3] N. M. Amato, B. Bayazit, L. Dale, C. Jones, and D. Vallejo, "OBPRM: An obstacle-based PRM for 3d workspaces," in *Robotics: The Algorithmic Perspective*, P. Agarwal, L. E. Kavraki, and M. Mason, Eds. AK Peters, 1998, pp. 156–168.

[4] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo, "Choosing good distance metrics and local planners for probabilistic roadmap methods," *IEEE Transactions on Robotics and Automation*, pp. 442–447, 2000.

[5] N. M. Amato and L. Dale, "Proabilistic roadmap methods are embarrasingly parallel," in *IEEE International Conference on Robotics and Automation*, 1999, pp. 688–694.

[6] N. M. Amato, K. Dill, and G. Song, "Using motion planning to map protein folding landscapes and analyze folding kinetics of known native structures," in *International Conference on Research in Computational Molecular Biology*, April 2002, pp. 2–11.

[7] M. S. Apaydin, D. L. Brutlag, C. Guestrin, D. Hsu, and J.-C. Latombe, "Stochastic roadmap simulation: An efficient representation and algorithm for analyzing molecular motion," in *International Conference on Research in Computational Molecular Biology*, April 2002, pp. 12–21.

[8] J. Barraquand and J.-C. Latombe, "Robot motion planning: A distributed representation approach," *International Journal of Robotics Research*, vol. 10, no. 6, pp. 628–649, December 1991.

[9] K. E. Bekris, B. Y. Chen, A. M. Ladd, E. Plaku, and L. E. Kavraki, "Multiple query motion planning using single query primitives," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003, pp. 656–661.

[10] P. Bessiere, E. Mazer, and J.-M. Ahuactzin, "The Ariadne's Clew algorithm," *Journal of Artificial Intelligence Research*, vol. 9, pp. 295–316, 1998.

[11] R. Bohlin and L. E. Kavraki, "A randomized algorithm for robot path planning based on lazy evaluation," in *Handbook on Randomized Computing*, P. Pardalos, S. Rajasekaran, and J. Rolim, Eds. Kluwer Academic Publishers, 2001, pp. 221–249.

[12] V. Boor, M. H. Overmars, and A. F. van der Stappen, "The gaussian sampling strategy for probabilistic roadmap planners," in *IEEE International Conference on Robotics and Automation*, 1999, pp. 1018–1023.

[13] M. S. Branicky, S. M. LaValle, K. Olson, and L. Yang, "Quasi-randomized path planning," in *IEEE International Conference on Robotics and Automation*, 2001, pp. 1481–1487.

[14] S. Carpin and E. Pagello, "On parallel RRTs for multi-robot systems," in *8th Conference of the Italian Association for Artificial Intelligence*, 2002, pp. 834–841.

[15] D. J. Challou, D. Boley, M. L. Gini, and V. Kumar, "A parallel formulation of informed randomized search for robot motion planning problems," in *IEEE International Conference on Robotics and Automation*, 1995, pp. 709–714.

[16] D. J. Challou, M. L. Gini, and V. Kumar, "Parallel search algorithms for robot motion planning," in *IEEE International Conference on Robotics and Automation*, 1993, pp. 46–51.

[17] M. de Berg, M. van Kreveld, and M. H. Overmars, *Computational Geometry: Algorithms and Applications*. Berlin: Springer, 1997.

[18] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM*, vol. 40, pp. 1048–1066, 1993.

[19] S. A. Ehmann and M. C. Lin, "Geometric algorithims: Accurate and fast proximity queries between polyhedra using convex surface decomposition," *Computer Graphics Forum - Proceedings of Eurographics*, vol. 20, pp. 500–510, 2001.

[20] R. Geraerts and M. Overmars, "A comparitive study of probabilistic roadmap planners," in *Algorithmic Foundations of Robotics V*, J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, Eds. Springer-Verlag, 2002, pp. 43–58.

[21] L. J. Guibas, C. Holleman, and L. E. Kavraki, "A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1999, pp. 254–260.

[22] D. Henrich, C. Wurll, and H. Wörn, "Multi-directional search with goal switching for robot path planning," in *International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, 1998, pp. 75–84.

[23] C. Holleman and L. E. Kavraki, "A framework for using the workspace medial axis in PRM planners," in *IEEE International Conference on Robotics and Automation*, 2000, pp. 1408–1413.

[24] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin, "On finding narrow passages with probabilistic roadmap planners," in *Robotics: The Algorithmic Perspective*, P. Agarwal, L. E. Kavraki, and M. Mason, Eds. Wellesley, MA: A.K. Peters, 1998, pp. 141–154.

[25] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, 2002.

[26] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *IEEE International Conference on Robotics and Automation*, 1997, pp. 2719–2726.

[27] D. Hsu, T. Jiang, J. Reif, and Z. Sun, "The bridge test for sampling narrow passages with probabilistic roadmap planners," in *IEEE International Conference on Robotics and Automation*, 2003, pp. 4420–4426.

[28] P. Isto, "Constructing probabilistic roadmaps with powerful local planning and path optimization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002, pp. 2323–2328.

[29] L. E. Kavraki, M. N. Kolountzakis, and J.-C. Latombe, "Analysis of probabilistic roadmaps for path planning," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 1, pp. 166–171, February 1998.

[30] L. E. Kavraki and J.-C. Latombe, "Probabilistic roadmaps for robot path planning," in *Practical Motion Planning in Robotics: Current Approaches and Future Challenges*, K. Gupta and A. P. del Pobil, Eds. West Sussex, England: John Wiley, 1998, pp. 33–53.

[31] L. E. Kavraki, J.-C. Latombe, R. Motwani, and P. Raghavan, "Randomized query processing in robot motion planning," in *Proceedings of ACM Symposium on Theory of Computing*, 1995, pp. 353–362.

[32] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, June 1996.

[33] W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technology Journal*, vol. 49, pp. 291–307, 1970.

[34] A. M. Ladd and L. E. Kavraki, "Measure theoretic analysis of probabilistic path planning," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 2, pp. 229–242, 2004.

[35] ——, "Using motion planning for knot untangling," *International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 797–808, 2004.

[36] ——, "Fast exploration for robots with dynamics," in *Workshop on Algorithmic Foundations of Robotics*, 2004.

[37] F. Lamiraux and L. E. Kavraki, "Planning paths for elastic objects under manipulation constraints," *International Journal of Robotics Research*, vol. 20, no. 3, pp. 188–208, 2001.

[38] S. M. LaValle and M. S. Branicky, "On the relationship between classical grid search and probabilistic roadmaps," in *Algorithmic Foundations of Robotics V*, J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, Eds. Springer-Verlag, 2002, pp. 59–76.

[39] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *New Directions in Algorithmic and Computational Robotics*, B. R. Donald, K. Lynch, and D. Rus, Eds. AK Peters, 2001, pp. 293–308.

[40] ——, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.

[41] T.-Y. Lie and Y.-C. Shie, "An incremental approach to motion planning with roadmap management," in *IEEE International Conference on Robotics and Automation*, vol. 4, 2002, pp. 3411–3416.

[42] S. R. Lindemann and S. M. LaValle, "Current issues in sampling-based motion planning," in *Interantional Symposium on Robotics Research*, ser. Springer Tracts in Advanced Robotics (STAR), D. Paolo and R. Chatila, Eds. Springer Verlag, 2003.

[43] T. Lozano-Pérez and P. O'Donnell, "Parallel robot motion planning," in *Proc. IEEE Internat. Conf. on Robotics and Automation*, Sacramento, USA, 1991, pp. 1000–1007.

[44] J. Michel, F. Pellegrini, and J. Roman, "Unstructured graph partitioning for sparse linear system solving," in *Solving Irregularly Structured Problems in Parallel*, ser. Lecture Notes in Computer Science, 1997, pp. 273–286.

[45] M. Moll and L. E. Kavraki, "Path planning for variable resolution minimal-energy curves of constant length," in *IEEE International Conference on Robotics and Automation*, Barcelona, Spain, 2005.

[46] M. Morales, S. Rodriguez, and N. Amato, "Improving the connectivity of PRM roadmaps," in *IEEE International Conference on Robotics and Automation*, 2003, pp. 4427–4432.

[47] M. Overmars and P. Švestka, "A probabilistic learning approach to motion planning," in *Algorithmic Foundations of Robotics*, K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, Eds. A.K. Peters, 1995, pp. 19–37.

[48] C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, 1982.

[49] J. Phillips, L. E. Kavraki, and N. Bedrossian, "Spacecraft rendezvous and docking with real-time, randomized optimization," in *AIAA Guidance, Navigation, and Control*, 2003.

[50] E. Plaku and L. E. Kavraki, "Distributed sampling-based roadmap of trees for large-scale motion planning," in *IEEE International Conference on Robotics and Automation*, Barcelona, Spain, 2005.

[51] A. Sahbani, J. Cortés, and T. Siméon, "A probabilistic algorithm for manipulation planning under continuous grasps and placements," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2, 2002, pp. 1560– 1565.

[52] G. Sánchez and J.-C. Latombe, "On delaying collision checking in PRM planning: Application to multi-robot coordination," *International Journal of Robotics Research*, vol. 21, no. 1, pp. 5–26, 2002.

[53] M. Strandberg, "Augmenting RRT-planners with local trees," in *IEEE International Conference on Robotics and Automation*, 2004, pp. 3258–3262.

[54] P. Švestka and M. Overmars, "Motion planning for car-like robots using a probabilistic learning approach," *International Journal of Robotics Research*, vol. 16, no. 2, pp. 119–143, 1997.

[55] S. A. Wilmarth, N. M. Amato, and P. F. Stiller, "MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space," in *IEEE International Conference on Robotics and Automation*, 1999, pp. 1024–1031.

[56] M. Yim, D. G. Duff, and K. D. Roufas, "PolyBot: a modular reconfigurable robot," *IEEE International Conference on Robotics and Automation*, pp. 514–520, 2000.

Erion Plaku is a Ph.D. candidate in computer science at Rice University. He is studying algorithmic robotics under Dr. Lydia E. Kavraki. His research interests include motion planning and large scale distribution of motion planning for high-dimensional systems with dynamics. He received the B.S. degree (*summa cum laude*) in computer science and mathematics from the State University of New York, Fredonia, NY, in 2000 and the M.S. degree in computer science from Clarkson University, Potsdam, NY, in 2002.

Kostas E. Bekris is a Ph.D. candidate in computer science at Rice University. He is studying algorithmic robotics under Dr. Lydia E. Kavraki. His research interests include mobile robot navigation with sensing constraints, motion planning and robotic sensor networks. He received the B.S. degree in computer science from the University of Crete, Greece, in 2001 and the M.S. degree in computer science from Rice University, Houston, TX, in 2004.

Brian Y. Chen is a Ph.D. candidate in computer science at Rice University. He is studying algorithms for geometric comparison of protein structures under Dr. Lydia E. Kavraki. His research interests include protein function prediction, geometric pattern matching, and large scale distribution of pattern matching algorithms for bioinformatics applications. He received the B.A. degree in mathematics and the B.A. degree in computer science from Rutgers State University of New Jersey, Piscataway NJ, in 2000, and the M.S. degree in computer science from Rice University, Houston, TX, in 2003.

Andrew M. Ladd is a Ph.D. candidate in computer science at Rice University. He is studying algorithmic robotics under Dr. Lydia E. Kavraki. His research interests include motion planning with dynamics, uncertainty models in robotics and real-time planning. He received the B.S. degree in joint honors in mathematics and computer science from McGill University, Montreal, Canada, in 2000 and the M.S. degree in computer science from Rice University, Houston, TX, in 2003.

Lydia E. Kavraki is the Noah Harding Professor of Computer Science and Bioengineering at Rice University. Kavraki received a B.A. degree in computer science from the University of Crete in Greece, and a Ph.D. degree in computer science from Stanford University. Kavraki's research is in physical algorithms and their applications to robotics and bioinformatics. Her recent work focuses on the development of methods for robot planning in high dimensions and with physical constraints, planning with sensor nets, assembly planning, micromanipulation using microelectromechanical systems and flexible object manipulation. Kavraki also applies robotics methods to the modeling of biomolecular interactions for drug design. Her awards include the Grace Murray Hopper Award from the Association for Computing Machinery, a Sloan Fellowship, the Early Academic Career Award from the IEEE Society on Robotics and Automation. Kavraki is a member of IEEE and ACM and a Fellow of the American Institute for Medical and Biological Engineering.