Discrete Search Leading Continuous Exploration for Kinodynamic Motion Planning

Erion Plaku, Lydia E. Kavraki, and Moshe Y. Vardi
Department of Computer Science, Rice University, Houston, Texas 77005
Email: {plakue, kavraki, vardi}@cs.rice.edu

***Abstract*— This paper presents the Discrete Search Leading continuous eXploration (DSLX) planner, a multi-resolution approach to motion planning that is suitable for challenging problems involving robots with kinodynamic constraints. Initially the method decomposes the workspace to build a graph that encodes the physical adjacency of the decomposed regions. This graph is searched to obtain *leads*, that is, sequences of regions that can be explored with sampling-based tree methods to generate solution trajectories. Instead of treating the discrete search of the adjacency graph and the exploration of the continuous state space as separate components, DSLX passes information from one to the other in innovative ways. Each lead suggests what regions to explore and the exploration feeds back information to the discrete search to improve the quality of future leads. Information is encoded in edge weights, which indicate the importance of including the regions associated with an edge in the next exploration step. Computation of weights, leads, and the actual exploration make the core loop of the algorithm.**

Extensive experimentation shows that DSLX is very versatile. The discrete search can drastically change the lead to reflect new information allowing DSLX to find solutions even when sampling-based tree planners get stuck. Experimental results on a variety of challenging kinodynamic motion planning problems show computational speedups of two orders of magnitude over other widely used motion planning methods.

I. Introduction

Robot motion planning with complex kinodynamic constraints is a topic that has attracted a lot of recent attention [1]–[5]. It is greatly motivated by the availability of new robotic systems and the need to produce paths that respect the physical constraints in the motion of the robotic systems and hence can be translated into trajectories executed by the real platforms with minimum effort. Sampling-based tree planners, such as Rapidly-exploring Random Tree (RRT) [6], Expansive Space Tree (EST) [7], and others [1]–[5], [8]–[10] have in recent years been widely successful in kinodynamic motion planning. Such planners typically explore the state space using a single or a bidirectional tree [1]–[5], [7], [9], or multiple trees, as in the case of Sampling-based Roadmap of Trees (SRT) [10]. Recent books [1], [2] contain additional references and descriptions of many successful sampling-based tree planners.

Given the large amount of work on kinodynamic motion planning and the dominant place of sampling-based tree planners, a new planner for robots with kinodynamic constraints can be justified only if it offers significant advantages over previous work. This paper describes DSLX, a multi-resolution approach that as other existing and highly successful sampling-based tree planners (e.g., RRT, EST, SRT) uses tree exploration of the state space, but displays a superior performance when compared to them.

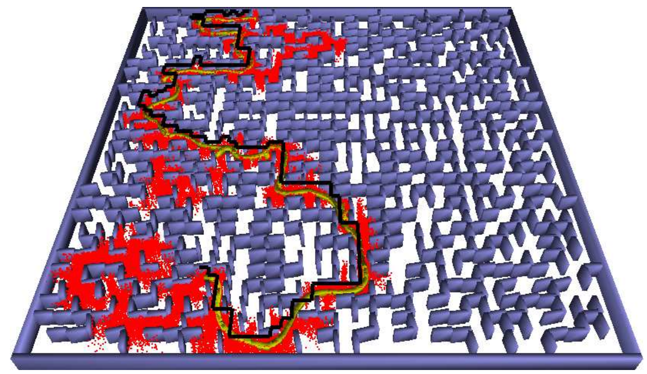


Fig. 1. Example of benchmark "RandomSlantedWalls," a kinodynamic motion planning problem solved by DSLX two orders of magnitude faster than other tree-based motion planning methods. The robot model is that of a smooth car (see Section III-A). Red dots indicate projections of the states of the exploration tree onto the workspace. The black line indicates the current lead that is used to guide the tree exploration toward the goal.

DSLX utilizes information provided by the problem specification and information gathered during previous exploration steps to guide future explorations closer to obtaining a solution to the given motion planning problem. This is a concept that has been studied before mainly in the context of sampling-based planners that construct roadmaps. For example, the original Probabilistic Roadmap Method (PRM) [11] uses the information of the connectivity of the samples to create more samples in parts of the space where connectivity is low. The work in [12] uses nearest-neighbors information in the context of PRM to define the utility of each sample in an information-theoretic sense and only add to the roadmap those samples that increase the overall entropy. The planners in [13] and [14] also utilize information in the context of PRM to find appropriate sampling strategies for different parts of the configuration space. In contrast to roadmap methods, traditional tree-based methods such as RRT [6], ADDRRT [4], EST [7] rely on limited information, such as distance metrics or simple heuristics to guide the exploration. Although the tree may advance quickly towards its goal, if it gets stuck it becomes more and more difficult to find promising directions for the exploration.

DSLX is a tree-based planner that systematically uses the information gathered during previous explorations steps to lead future explorations toward increasingly promising directions. Initially, DSLX constructs a coarse-resolution representation of the motion planning problem by obtaining a decomposition of the workspace. The decomposition is represented in terms of a graph whose vertices are regions in the decomposition and whose edges denote physical adjacency between different regions. DSLX exploits the simple observation that any solution trajectory that connects the initial state to the goal state corresponds to some coarse-grained sequence of neighboring regions that starts and ends at regions associated with the initial and goal states, respectively. Although the converse does not hold, a sequence of coarse-grained neighboring regions can however serve as a guide to the exploration.

The idea of using decompositions of the workspace, configuration space and state space appear early in the motion planning literature. Key theoretical results in motion planning were obtained using decomposition based methods [15]. Some of the first planners obtained decompositions of the workspace into regions that were linked in a graph which was subsequently used to find a path. An extensive discussion of early exact and approximate cell decomposition methods can be found in [15] (Chapters 5 and 6) and in [16]–[18]. Initially only geometric planning was considered. More recently approaches that deal with kinodynamic planning have appeared in the context of sampling-based methods [19]–[22].

The decomposition graph is weighted and the initial weights are all set to a fixed value. The core part of DSLX proceeds by repeating the following three steps until a solution is found or a maximum amount of time has elapsed:

1) Obtain a guide sequence, called a *lead*, by some discrete search method on the decomposition graph.
2) Explore the continuous state space for a short period of time. DSLX attempts to extend the branches of an exploring tree from one region to its neighbor, as specified by the lead.
3) Update the weights of the decomposition graph. The weight of an edge represents an estimation of how important the exploration of the regions it connects is for the construction of solution trajectories. The weight depends on the total time spent so far exploring, the progress of the exploration in these regions and other quantities. The weights are updated after each exploration of a region to reflect the new information gathered during the most recent exploration.

A critical difference and advantage of DSLX over earlier workspace decomposition methods is the close interaction of the discrete search and the continuous exploration and the flexibility this interaction provides. The coarse-grained representation can provide DSLX with many alternative leads. A central issue is which lead to choose from the possibly combinatorially large number of possibilities. Since the weight estimates that bias the computation of leads are based on partial information, it is important not to ignore leads associated with lower weights, especially during the early stages of the exploration. DSLX aims to strike a balance between greedy and methodical search by selecting more frequently sequences of coarse-grained regions that are associated with higher weights and less frequently sequences of coarse-grained regions that are associated with lower weights.

Through extensive experimentation on a variety of kinodynamic motion planning problems it has been observed that DSLX can focus the exploration on promising search directions while able to radically change these directions if the information gathered during exploration suggests other promising leads. This flexibility tends to prevent the method from getting stuck in the way that other sampling-based tree planners do. Extensive comparisons have been done with RRT [6], a more recent version of RRT called Adaptive Dynamic Domain RRT (ADDRRT) [4], EST [7], and SRT [10] showing that DSLX can be up to two orders of magnitude more efficient. Fig. 1 shows one such case for kinodynamic motion planning where DSLX finds solutions more than 170 times faster than the single-tree based methods and 90 times faster than SRT.

This paper is organized as follows. Section II describes DSLX in detail. Experiments and results are described in Section III. Section IV discusses the experimental results and provides insights into the reasons for the observed computational efficiency of DSLX. The paper concludes in Section V with a summary and directions for future work.

II. DSLX

Pseudocode for the overall approach is given in Algorithm 1. The construction of the coarse-grained decomposition into neighboring regions is given in line 2 and described in Section II-A. The lead computation occurs in line 6 and is described in Section II-B. The other lines refer to the state space exploration, which is described in Section II-C.

Algorithm 1 Pseudocode for DSLX

Input:
$\mathcal{W}$, geometric description of the workspace
$\mathcal{R}$, motion model and geometric description of the robot
s, g, initial and goal specifications
$t_{\max} \in \mathbb{R}^{>0}$, upper bound on computation time
$t_e \in \mathbb{R}^{>0}$, short time allocated to each exploration step
Output: A solution trajectory or NIL if no solution is found

1: STARTCLOCK1
2: $G = (V, E) \leftarrow$ COARSEGRAINEDDECOMPOSITION($\mathcal{W}$)
3: INITEXPLORATIONESTIMATES(G)
4: $\mathcal{T} \leftarrow$ exploration tree rooted at s
5: **while** ELAPSEDTIME1 $< t_{\max}$ **do**
6:    $[\mathrm{R}_{i_1}, \ldots, \mathrm{R}_{i_n}] \leftarrow$ COMPUTELEAD(G, s, g)
7:    STARTCLOCK2
8:    **while** ELAPSEDTIME2 $< t_e$ **do**
9:       $\mathrm{R}_{i_j} \leftarrow$ SELECTREGION($[\mathrm{R}_{i_1}, \ldots, \mathrm{R}_{i_n}]$)
10:       **for** several times **do**
11:          $x \leftarrow$ SELECTSTATEFROMREGION(R_{i_j})
12:          PROPAGATEFORWARD($\mathcal{T}, x, \mathrm{R}_{i_j}, \mathrm{R}_{i_{j+1}}$)
13:    **if** a solution is found **then**
14:       **return** solution trajectory
15:    UPDATEEXPLORATIONESTIMATES($G, [\mathrm{R}_{i_1}, \ldots, \mathrm{R}_{i_n}]$)
16: **return** NIL

A. Coarse-Grained Decomposition

This paper uses a simple grid-based decomposition of the workspace. Even with this simple decomposition, DSLX is computationally efficient in solving challenging kinodynamic motion planning problems, as indicated by the experimental results in Section III-D.

The coarse-grained decomposition of the workspace provides a simplified layer to the motion planning problem. As discussed in Section II-B, DSLX uses the coarse-grained decomposition to compute general leads from the initial to the goal region. It is important to note that DSLX allows for a great degree of flexibility on the decomposition of the workspace. In particular, since DSLX relies on information collected during exploration to determine which lead to select, it does not even require that the decomposition be collision-free. When regions that are occupied by obstacles are selected as part of the lead, the exploration estimates will indicate that no progress can be made. Consequently, such regions will be selected less and less frequently. The use of better workspace decompositions (see [1], [2] for details on different decomposition methods) may certainly further improve the computational efficiency of DSLX, since it will in general provide better search directions.

B. Coarse-Grained Leads

The coarse-grained decomposition is used to obtain sequences of neighboring regions that provide promising leads for the state space exploration. A transition graph $G = (V, E)$ is constructed based on the coarse-grained decomposition as described in Section II-A. Each vertex $v_i \in V$ is associated with a region R_i of the coarse-grained decomposition and an edge $(v_i, v_j) \in E$ indicates that R_i and R_j are neighboring regions in the workspace decomposition. Let $v(s) \in V$ and $v(g) \in V$ be the two vertices whose corresponding regions are associated with the initial and goal states, s and g, respectively.

A lead is computed by searching G for sequences of edges from $v(s)$ to $v(g)$. A weight w_{ij} associated with each edge $(v_i, v_j) \in E$ is an estimate of DSLX on the importance of including R_i and R_j as an edge in the lead. w_{ij} is updated after each exploration of R_i and R_j. Sequences of edges associated with higher weights are selected more frequently since, according to current estimates such edge sequences provide promising leads.

1) Computation of w_{ij}: The coverage $c(\mathcal{T}, R_k)$ of a region R_k by the states of the tree $\mathcal{T}$ is estimated by imposing an implicit uniform grid on R_k and measuring the fraction of cells that contain at least the projection of one state from $\mathcal{T}$. Let $c(\mathcal{T}, R_k)$ denote the coverage estimate of R_k by $\mathcal{T}$ at the beginning of the current exploration step and let $c'(\mathcal{T}, R_k)$ denote the new coverage estimate at the end of the exploration step. Thus $\alpha(\mathcal{T}, R_k) = c'(\mathcal{T}, R_k) - c(\mathcal{T}, R_k)$ measures the change in the coverage of R_k by $\mathcal{T}$ as a result of the current exploration step. Then, the weight w_{ij} is defined as

$$w_{ij} = 0.5(\alpha(\mathcal{T}, R_i) + \alpha(\mathcal{T}, R_j))/t + \epsilon/t_{\mathrm{acc}}(i, j),$$

where t is the computational time devoted to the exploration of R_i, R_j during the current exploration step; $t_{\mathrm{acc}}(i, j)$ is the accumulated time over all explorations of R_i, R_j; and ϵ is a small normalization constant.

Large values of w_{ij} are obtained when branches of $\mathcal{T}$ quickly reach previously unexplored parts of R_i and R_j and are thus indicative of promising leads. The accumulated time $t_{\mathrm{acc}}(i, j)$ is used to give a higher weight to those regions that have been explored less frequently. Initially, since there is no exploration information available, each weight w_{ij} is set to a fixed value.

2) Computation of leads: Many possible strategies can be used to compute search directions. The computation of a lead is essentially a graph searching algorithm and the literature on this subject is abundant (see [23] for extensive references).

The combination of search strategies in this work aims to provide leads that are more frequently biased toward promising directions. However, random leads are also used, although less frequently, as a way to correct for errors inherent with the estimates. The use of random leads is motivated by observations made in [10], [24], where random restarts and random neighbors have been suggested as effective ways to unblock the exploration when tree planners or PRM get stuck.

COMPUTELEAD (line 6 of Algorithm 1) frequently returns the most probable sequence of edges in G from $v(s)$ to $v(g)$. The probability p_{ij} associated with $(v_i, v_j) \in E$ is computed by normalizing the weight w_{ij}, i.e., $p_{ij} = w_{ij}/\sum_{(v_k, v_\ell) \in E} w_{k\ell}$. The probability of a sequence of edges is then defined as the product of the probabilities associated with its edges. The most probable edge sequence from $v(s)$ to $v(g)$ is the one with the highest probability. In order to compute the most probable edge sequence as defined above using a shortest path algorithm, such as Dijkstra's, the weight function used in the graph search is set to $-\log(p_{ij})$ for $(v_i, v_j) \in E$.

Almost as frequently, COMPUTELEAD returns the acyclic sequence of edges from $v(s)$ to $v(g)$ with the highest sum of edge weights. In order to compute such sequence using Dijkstra's shortest path algorithm, the weight function used in the graph search is set to $w_{\max} - w_{ij}$ for $(v_i, v_j) \in E$, where $w_{\max}$ denotes the maximum value for the weights.

Less frequently, COMPUTELEAD computes a random sequence of edges from $v(s)$ to $v(g)$. This computation is carried out by using depth-first search, where the unvisited children are visited in a random order.

C. Exploration

The exploration starts by rooting a tree $\mathcal{T}$ at the specified initial state s (line 3). The objective is to quickly grow $\mathcal{T}$ toward the goal by using the coarse-grained leads as guides for the exploration. The exploration is an iterative process (lines 8–12). At each iteration a region R_{i_j} is selected from the coarse-grained lead $[R_{i_1}, \ldots, R_{i_n}]$ and explored for a short period of time. The exploration aims to extend branches of $\mathcal{T}$ from R_{i_j} to $R_{i_{j+1}}$. For this reason, several states are selected from the states associated with R_{i_j} and are propagated forward toward $R_{i_{j+1}}$.

SELECTREGION*:* Note that some regions in $[R_{i_1}, \ldots, R_{i_n}]$ may not contain any states from $\mathcal{T}$, since the branches of $\mathcal{T}$ have yet to reach such regions. Such regions are not considered for selection, since they do not contain any states from which to propagate forward, as required by line 12 of Algorithm 1.

The objective is to select a nonempty region $R_{i_j} \in [R_{i_1}, \ldots, R_{i_n}]$ whose exploration causes $\mathcal{T}$ to grow closer to the goal. Since $[R_{i_1}, \ldots, R_{i_n}]$ specifies a sequence of neighboring regions that end at the region associated with the goal state, the order in which the regions appear in the lead provides an indication of how close $\mathcal{T}$ is to the goal. For this reason, DSLX prefers to select regions that appear toward the end of $[R_{i_1}, \ldots, R_{i_n}]$ more frequently than regions that appear at the beginning. Preference is also given to regions that have been selected less frequently for exploration. The exploration of such regions is vital in order to provide a balance between greedy and methodical search. This objective is achieved by selecting a region R_{i_j} from $[R_{i_1}, \ldots, R_{i_n}]$ with probability $\alpha j/n + (1-\alpha)/\mathrm{nsel}(R_{i_j})$, where $0 < \alpha < 1$ is selected uniformly at random and $\mathrm{nsel}(R_{i_j})$ is the number of times R_{i_j} has been selected for exploration.

SELECTSTATEFROMREGION*:* Each state x associated with region R_{i_j} is selected with probability $1/\mathrm{nsel}(x)$, where $\mathrm{nsel}(x)$ is the number of times x has been selected. Preference is thus given to the states associated with R_{i_j} that have been selected less frequently. A state $x \in \mathcal{T}$ is associated with region R_{i_j} iff $\mathrm{proj}(x) \in R_{i_j}$, where $\mathrm{proj}(x)$ denotes the projection of the state $x \in \mathcal{T}$ onto the workspace $\mathcal{W}$.

PROPAGATEFORWARD*:* A state x is propagated forward to a new state x_{new} by selecting a control u and applying u to x for several time steps or until a collision is found. If x_{new} is more than a minimum number of propagation steps away from x, then x_{new} and the edge connecting x to x_{new} is added to $\mathcal{T}$. The state x_{new} is also added to the appropriate region R_k. Since the objective is to guide the propagation from R_{i_j} toward the neighboring region $R_{i_{j+1}}$, the control u is selected as the control that brings $\mathrm{proj}(x_{\mathrm{new}})$ closer to $R_{i_{j+1}}$ out of several controls sampled uniformly at random. The propagation is computed by integrating the motion equations of the robot. This work uses a fourth-order Runge-Kutta integrator [1], [2].

III. EXPERIMENTS AND RESULTS

The design of DSLX was motivated by challenging kinodynamic motion planning problems with vehicles and the experiments in this paper were chosen to test the efficiency of DSLX in solving such problems. The performance of DSLX is compared against several existing state-of-the-art methods. Results presented in Section III-D show the competiveness of the proposed method, DSLX and highlight the benefits of incorporating discrete-search and coarse-grained decomposition into sampling-based approaches, as proposed in this work.

A. Robot Models

The motions of the robot are defined by a set of ordinary differential equations. The robot models used in this work consist of a kinematic car (KCar), a smooth (second-order) car (SCar), smooth unicycle (SUni), and a smooth differential drive (SDDrive). Detailed descriptions of these models can be found in [1], [2]. The range of controls and bounds on the state variables are empirically determined based on the workspaces used for the experiments.

1) Kinematic Car (KCar): The motion equations are $\dot{x} = u_0 \cos(\theta); \dot{y} = u_0 \sin(\theta); \dot{\theta} = u_0 \tan(u_1)/L$, where (x, y, θ) is the car configuration; u_0 and u_1 are the speed and steering wheel controls; L is the distance between the front and rear axles. The speed and steering control are restricted to $|u_0| \leq \mathrm{v}_{\max} = 1$ and $|u_1| \leq \psi_{\max} = \pi/4$.

2) Smooth Car (SCar): The kinematic car model can be extended to a second-order model by expressing the velocity v and steering angle ϕ as differential equations of the acceleration u_0 and the rotational velocity of the steering wheel u_1 controls, as follows: $\dot{x} = v\cos(\theta); \dot{y} = v\sin(\theta); \dot{\theta} = v\tan(\phi)/L; \dot{v} = u_0; \dot{\phi} = u_1$. The acceleration and rotational velocity of the steering wheel controls are restricted to $|u_0| \leq 0.0015\mathrm{v}_{\max}$ and $|u_1| \leq 0.0015\psi_{\max}$.

3) Smooth Unicycle (SUni): The motion equations are $\dot{x} = v\cos(\theta); \dot{y} = v\sin(\theta); \dot{\theta} = \omega; \dot{v} = u_0; \dot{\omega} = u_1$, where (x, y, θ) is the configuration; ω and v are the rotational and translational velocities, respectively. The translation u_0 and rotational u_1 accelerations are restricted to $|u_0| \leq 0.0015r$ and $|u_1| \leq 0.0015r$, where r is the radius of the unicycle.

4) Smooth Differential Drive (SDDrive): The motion equations are $\dot{x} = 0.5r(\omega_\ell + \omega_r)\cos(\theta); \dot{y} = 0.5r(\omega_\ell + \omega_r)\sin(\theta); \dot{\theta} = r(\omega_r - \omega_\ell)/L; \dot{\omega_\ell} = u_0; \dot{\omega_r} = u_1$, where (x, y, θ) is the configuration; ω_ℓ and ω_r are the rotational velocities of the left and right wheels, respectively; r is the wheel radius; and L is the length of the axis connecting the centers of the two wheels. In this work, the controls u_0 and u_1 are restricted to $|u_0| \leq 0.15$ and $|u_1| \leq 0.15$.

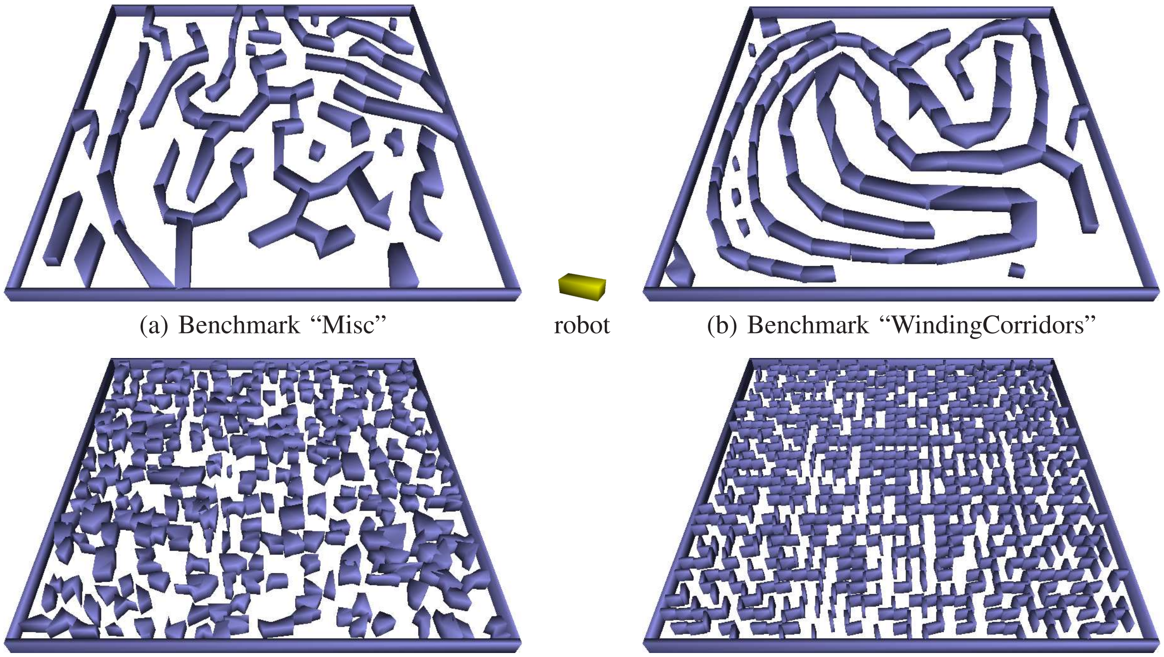
B. Benchmarks

The benchmarks used in the experiments are designed to vary in type and difficulty and to test different aspects of motion planning methods. Illustrations of benchmarks and robot geometries can be found in Fig. 1 and Fig. 2.

Benchmark "Misc" consists of several miscellaneous obstacles arranged as in Fig. 2(a). Random queries are created that place the robot in opposite corners of the workspace. In this way, the robot must wiggle its way through the various obstacles and the narrow passages in the workspace.

Benchmark "WindingCorridors" consists of long and winding corridors, as shown in Fig. 2(b). Random queries are created by placing the robot in two different corridors, either 4 and 5 or 5 and 4 (counting from left to right), respectively. This benchmark is chosen to illustrate the efficacy of motion planning methods in solving problems where even though the initial and goal specification place the robot in neighboring places in the workspace, the solution trajectory is rather long and the robot travels through a large portion of the workspace.

Benchmark "RandomObstacles" consists of a large number of obstacles (278 obstacles) of varying sizes placed at random throughout the workspace, as shown in Fig. 2(c). The random placement of the obstacles creates many narrow passages, posing a challenging problem for motion planning methods, since research [1], [2] has shown that many motion planners



(a) Benchmark “Misc” (b) Benchmark “WindingCorridors”

(c) Benchmark “RandomObstacles” (d) Benchmark “RandomSlantedWalls”

Fig. 2. Several benchmarks used for the experimental comparisons of DSLX. In each case, the robot geometry is a box and the workspace is a unit box. The body length and width of the robot in benchmarks “RandomObstacles” and “RandomSlantedWalls” are 1/40 and 1/60, respectively. In the case of benchmarks “Misc” and “WindingCorridors” the robot is twice the size of the one used in benchmarks “RandomObstacles” and “RandomSlantedWalls,” since “Misc” and “WindingCorridors” have in general more open areas and wider passages.

have a tendency of getting stuck in such random environments with narrow passages. Random queries place the robot in opposite sides of the workspace.

Benchmark “RandomSlantedWalls” consists of 890 obstacles resembling slanted walls, as illustrated in Fig. 1. Initially, a random maze is created using the disjoint set strategy and then only 97% of the maze walls are kept. Knocking down of the maze walls creates multiple passages in the workspace for connecting any two points. The dimensions of the remaining walls are set uniformly at random from the interval $[1/60, 1/90]$ in order to create obstacles of different sizes. Each of the remaining walls is rotated by some angle chosen at random from $[2^\circ, 15^\circ]$, so that the walls are aligned at different angles. This benchmark tests the efficiency of motion planning methods in finding solutions for problems with multiple passages. Random queries place the robot in opposite sides of the workspace.

C. Other Motion Planning Methods used in Comparisons

This work presents comparisons with RRT [2], [6], ADDRRT [4], EST [7], and SRT [10]. Standard implementations were followed as suggested in the respective research papers and motion planning books [2], [6]. These implementations utilize the OOPS-MP (Online Open-source Programming System for Motion Planning) framework [25] and are well-tested, robust, and efficient, as they have been widely used by our research group. Every effort was made to fine-tune the performance of these motion planners for the experimental comparisons presented in this paper.

In addition to single-tree versions, bidirectional versions of RRT, ADDRRT, and EST also exist. It has also been shown that SRT [10] takes the bidirectional search a step further and uses single-tree based methods such as RRT, EST, etc., to grow multiple trees in different regions of the state space and then connects the neighboring trees to obtain a solution trajectory. Note that for nonholonomic problems tree connections however may contain gaps [1], [2]. Trajectories obtained by DSLX, RRT, ADDRRT, and EST do not contain any gaps, while trajectories obtained by SRT contain gaps. Such gaps could be closed using steering or numerical methods [26] at the expense of incurring additional computational costs.

D. Results

For each benchmark, experiments are run using each of the robot models described in Section III-A. For each combination of benchmark and robot model, 30 random queries are generated as described in Section III-B. Each motion planning method is then used to solve all the input random queries. In each instance, the computational time required to solve the query is measured. Rice PBC and Cray XD1 ADA clusters were used for code development. Experiments were run on ADA, where each of the processors runs at 2.2GHz and has up to 8GB of RAM.

Table I contains a summary of the experimental results. For each benchmark and robot model combination, the table indicates the average computational time required by each motion planning method to solve 30 random queries. In addition, Table I indicates the computational speedup obtained by DSLX in comparison to the other motion planning methods used in the experiments of this work. The experimental comparisons of DSLX with the single-tree methods are summarized in Table I (columns 1–3), while the comparisons with the multi-tree method SRT are summarized in Table I (column 4).

1) Comparison with the single-tree methods: Table I shows that DSLX is consistently more efficient than RRT, ADDRRT, and EST. For each benchmark and robot model combination, the average time required to solve a query is considerably lower for DSLX.

When the simple kinematic car model (KCar) is used, DSLX is between 3–12 times faster on "Misc."; 9–13 on "WindingTunnels"; 3–10 on "RandomObstacles"; and 23–29 times faster on "RandomSlantedWalls."

When the other robot models are used, DSLX is between 7–32 times faster on "Misc.,"; 9–29 on "WindingTunnels"; 36–69 on "RandomObstacles"; and 102–255 times faster on "RandomSlantedWalls."

2) Comparison with the multi-tree method SRT: The best computational times for SRT are obtained when several trees are also grown in other parts of the state space in addition to the trees grown at the initial and goal states. The computational times obtained by SRT tend to be lower than the computational times required by the bidirectional versions of RRT, ADDRRT, and EST. Recall that, as discussed in Section III-C, the trajectories computed by SRT contain gaps, while trajectories computed by DSLX do not contain any gaps. Results indicated in Table I are obtained when SRT grows 75 trees using EST as its building block. Similar results were obtained when RRT is used as a building block of SRT.

As indicated in Table I, DSLX is in each case computationally faster than SRT. The computational speedup of DSLX varies from a factor of 3–10 on the easier problems to a factor of 48–90 times on the more challenging problems.

IV. A Closer Look at the State Space Exploration

Experimental results presented in Table I indicate that DSLX offers considerable computational advantages over the other motion planning methods used in this work across a variety of challenging benchmarks and robot models. The experimental results show that DSLX is capable of solving challenging motion planning methods in a matter of one to three minutes as opposed to several hours required by other methods. DSLX computationally outperforms powerful motion planning methods, such as RRT, ADDRRT, EST, and SRT, by an order of magnitude on easy problems and as much as two orders of magnitude on more challenging problems.

The understanding of the main reasons for the success of a motion planning method is in general a challenging issue and subject of much research. This section takes a closer look at the exploration done by RRT, ADDRRT, EST, and SRT and compares it to the exploration done by DSLX in order to provide some insights behind the computational efficiency of DSLX.

By using nearest neighbors to random states as exploration points, RRT [2], [6] is frequently led toward obstacles where it may remain stuck for some time [1], [2], [4], [10]. Adjusting the exploration step size of RRT, as ADDRRT does, has been shown to alleviate the problem to a certain extent but not in all situations [4]. The use of ADDRRT incurs additional computational costs, which in some cases, as those observed in this work, outweigh the benefits offered by ADDRRT. However, both in the case of RRT and ADDRRT, as the tree grows large, it becomes more frequent for the nearest neighbors to random states not to be at the frontier of the tree but instead at "inner" nodes of the tree. Consequently, especially in challenging problems where propagation is difficult, these methods end up exploring the same region many times, thus wasting computational time.

EST [7] on the other hand suffers from a different kind of problem. EST directs the search toward less explored regions of the state space. As the tree grows large, the growth of the tree slows down as there are many regions with similar low density distributions. Consequently, EST ends up slowly expanding the tree in all possible directions, which do not necessarily bring the exploration closer to the goal region.

SRT [10] approaches the above problems by using multiple trees in randomly selected regions of the state space and only growing each tree for a shorter period of time to avoid the slow down on the growth of the trees. SRT is however not particularly well suited for nonholonomic motion planning problems, since tree connections create gaps that may require considerable additional computational time to be closed [26].

Although these methods have been shown to work well in a variety of settings, the main drawback common to all these methods is that they only use a limited amount of information to guide the exploration. There is generally much more information available to motion planning methods that, if properly used, can significantly speed up the computations.

The main strength of DSLX is the systematic use of information gathered during previous explorations steps to guide future explorations. As detailed in Section II, DSLX takes into account all the available workspace information, the initial and goal specifications, and carefully and closely integrates the information gathered during exploration into a discrete search and exploration of the state space. The discrete search provides DSLX with leads that guide the exploration closer to the goal specification. The exploration of the state space provides valuable feedback information that is used by DSLX to refine the lead for the next exploration step. As the exploration progresses, the leads produced by DSLX become more accurate and thus cause the tree to reach the goal specification. Fig. 3 provides a snapshot of the exploration done by DSLX at different time intervals. The tree quickly grows and reaches the goal in a short amount of time.

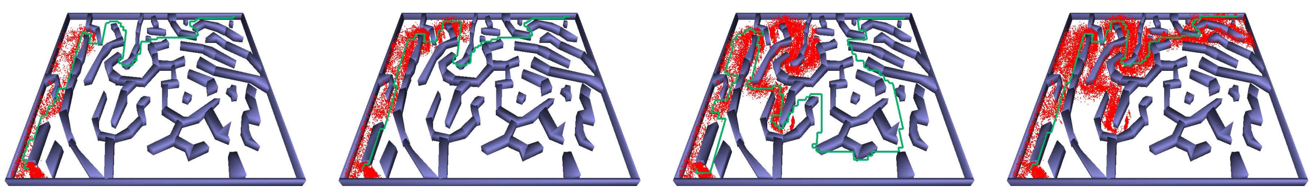
V. Discussion

We have presented DSLX, a tree-based motion planning method that relies on a decomposition of the workspace to obtain a coarse-grained representation of the motion planning problem and discrete search to find promising leads that bring the tree exploration closer to the goal. Information gathered

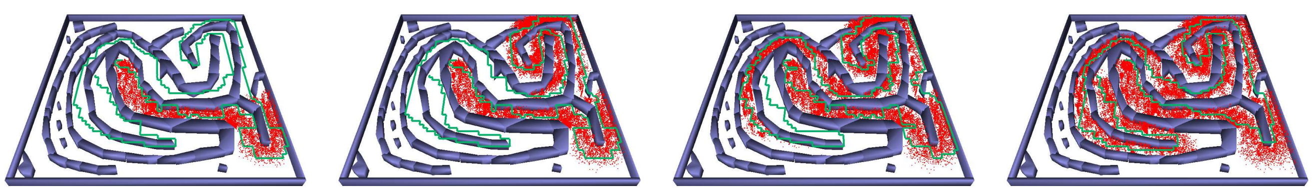
TABLE I

SUMMARY OF EXPERIMENTAL COMPARISONS WITH SINGLE- AND MULTIPLE-TREE METHODS

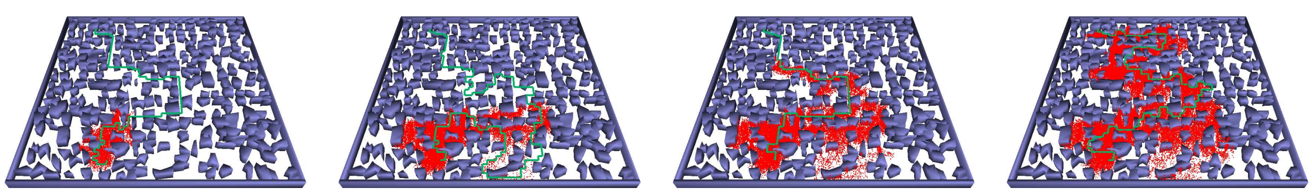
	Average time in seconds to solve one query					Speedup Obtained by DSLX			
	RRT	ADDRRT	EST	SRT	DSLX	RRT	ADDRRT	EST	SRT
"Misc"									
KCar	3.51	5.87	13.5	3.12	1.02	3.45	5.76	12.91	3.06
SCar	248.72	279.05	95.84	70.52	13.27	18.74	21.02	7.22	5.31
SUni	417.06	461.42	151.63	144.23	14.14	29.50	32.64	10.73	10.20
SDDrive	73.82	94.36	47.82	34.04	4.52	16.35	20.89	10.59	7.54
"WindingTunnels"									
KCar	15.33	19.29	22.94	12.21	1.70	9.03	11.37	13.52	7.18
SCar	282.49	231.11	90.76	92.32	9.46	29.85	24.42	9.59	9.75
SUni	161.16	175.92	83.14	106.73	8.57	18.21	20.53	9.70	12.45
SDDrive	178.10	213.35	142.59	108.75	7.60	23.43	28.06	18.76	14.30
"RandomObstacles"									
KCar	3.46	5.87	13.15	1.21	0.97	3.55	7.12	10.72	1.24
SCar	440.52	528.62	831.36	221.64	11.94	36.90	44.28	69.65	18.56
SUni	374.34	413.43	562.70	232.80	9.26	40.43	44.66	60.78	25.14
SDDrive	224.60	276.55	269.02	125.63	4.22	53.27	65.60	63.81	29.77
"RandomSlantedWalls"									
KCar	29.22	33.21	36.86	27.31	1.23	23.69	26.92	29.89	22.20
SCar	6330.95	6637.62	3716.27	1772.34	36.43	173.79	182.21	102.01	48.60
SUni	7207.27	6571.33	4819.99	2536.24	28.17	255.83	233.26	171.09	90.03
SDDrive	615.56	579.35	478.36	240.01	3.64	169.26	159.31	131.54	65.93



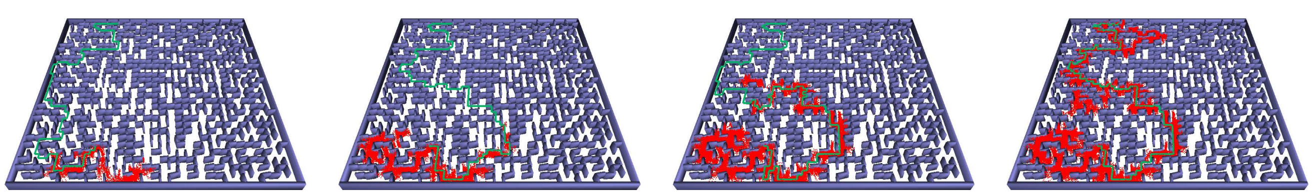
(a) Exploration of benchmark "Misc." after 2s, 4s, 6s, 8s of running time



(b) Exploration of benchmark "WindingTunnels" after 2s, 4s, 6s, 8s of running time



(c) Exploration of benchmark "RandomObstacles" after 2s, 4s, 6s, 8s of running time



(d) Exploration of benchmark "RandomSlantedWalls" after 6s, 12s, 18s, 24s of running time

Fig. 3. Snapshots of the tree exploration by DSLX of different benchmarks with the smooth car (SCar) as the robot model. Red dots indicate projections of the states of the exploration tree onto the workspace. The green line in each figure indicates the current lead.

during exploration is used to further refine the discrete search and improve the quality of future explorations. DSLX was shown to offer considerable computational advantages over other methods across a variety of kinodynamic motion planning problems. Experimental results indicated that DSLX is capable of solving challenging motion planning problems two orders of magnitude faster than other widely used motion planning methods.

The combination of coarse-grained representation, discrete search, and continuous exploration in the framework of DSLX results in an effective motion planner that allocates most of the available computational time to the exploration of the parts of the state space that lead to a solution for a given motion planning problem. This combination raises many interesting research issues, such as finding the best workspace decomposition for a motion planning problem, improving the discrete search, the continuous exploration, and the interaction between the different components, that we intend to investigate in future research. We are currently investigating extensions to the theoretical framework developed in [27] to analyze DSLX. Furthermore, we would like to apply and extend the framework of DSLX to increasingly challenging and high-dimensional problems in motion planning and other settings, such as hybrid-system testing [28]. As we address these challenging problems, it becomes important to extend the framework [29], [30] to obtain an effective distribution of DSLX that makes use of all the available computational resources.

Acknowledgment

This work has been supported in part by NSF CNS 0615328 (EP, LEK, MYV), NSF 0308237 (EP, LEK), a Sloan Fellowship (LEK), and NSF CCF 0613889 (MYV). Experiments were carried out on equipment supported by NSF CNS 0454333 and NSF CNS 0421109 in partnership with Rice University, AMD, and Cray.

References

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

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

[3] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in Robocup 2002: Robot Soccer World Cup VI*, Lecture Notes in Computer Science*, G. A. Kaminka, P. U. Lima, and R. Rojas, Eds. Berlin, Heiderlberg: Springer, 2003, vol. 2752, pp. 288–295.

[4] L. Jaillet, A. Yershova, S. M. LaValle, and T. Simeon, "Adaptive tuning of the sampling domain for dynamic-domain RRTs," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Edmonton, Canada, 2005, pp. 4086–4091.

[5] A. M. Ladd and L. E. Kavraki, "Motion planning in the presence of drift, underactuation and discrete system changes," in *Robotics: Science and Systems I*. Boston, MA: MIT Press, 2005, pp. 233–241.

[6] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," in *IEEE International Conference on Robotics and Automation*, 1999, pp. 473–479.

[7] 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.

[8] B. Burns and O. Brock, "Single-query motion planning with utility-guided random trees," in *IEEE Interantional Conference on Robotics and Automation*, Rome, Italy, Apr. 2007.

[9] 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.

[10] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki, "Sampling-based roadmap of trees for parallel motion planning," *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 597–608, 2005.

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

[12] B. Burns and O. Brock, "Toward optimal configuration space sampling," in *Robotics: Science and Systems*, Cambridge, MA, June 2005.

[13] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato, "A machine learning approach for feature-sensitive motion planning," in *International Workshop on Algorithmic Foundations of Robotics*, M. Erdmann, D. Hsu, M. Overmars, and A. F. van der Stappen, Eds. Utrecht/Zeist, Netherlands: Springer-Verlag, July 2004, pp. 361–376.

[14] D. Hsu, G. Sánchez-Ante, and Z. Sun, "Hybrid PRM sampling with a cost-sensitive adaptive strategy," in *IEEE Interantional Conference on Robotics and Automation*. Barcelona, Spain: IEEE Press, May 2005, pp. 3885–3891.

[15] J.-C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer, 1991.

[16] K. Kondo, "Motion planning with six degrees of freedom by multistrategic bidirectional heuristic free-space enumeration," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 267–277, 1991.

[17] J. Lengyel, M. Reichert, B. Donald, and P. Greenberg, "Real-time robot motion planning using rasterizing computer graphics hardware," *Computer Graphics (SIGGRAPH'90)*, pp. 327–335, 1990.

[18] R. Bohlin, "Path planning in practice: Lazy evaluation on a multi-resolution grid," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001, pp. 49–54.

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

[20] M. Foskey, M. Garber, M. C. Lin, and D. Manocha, "A voronoi-based hybrid motion planner," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, Maui, HI, 2001, pp. 55–60.

[21] Y. Yang and O. Brock, "Efficient motion planning based on disassembly," in *Robotics: Science and Systems*, S. Thrun, G. S. Sukhatme, and S. Schaal, Eds. Cambridge, USA: MIT Press, June 2005.

[22] H. Kurniawati and D. Hsu, "Workspace-based connectivity oracle: An adaptive sampling strategy for PRM planning," in *International Workshop on Algorithmic Foundations of Robotics*, S. Akella *et al.*, Eds. New York, NY: Springer-Verlag, 2006, in press.

[23] W. Zhang, *State-space Search: Algorithms, Complexity, Extensions, and Applications.* New York, NY: Springer Verlag, 2006.

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

[25] E. Plaku, K. E. Bekris, and L. E. Kavraki, "OOPS for Motion Planning: An Online Open-source Programming System," in *IEEE International Conference on Robotics and Automation (ICRA)*, Rome, Italy, 2007, pp. 3711–3716.

[26] 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 Interantional Conference on Robotics and Automation*, 2004, pp. 4362–4368.

[27] 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.

[28] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Hybrid systems: From verification to falsification," in *International Conference on Computer Aided Verification*, W. Damm and H. Hermanns, Eds. Berlin, Germany: Lecture Notes in Computer Science, Springer-Verlag, 2007, vol. 4590, pp. 468–481.

[29] E. Plaku and L. E. Kavraki, "Distributed sampling-based roadmap of trees for large-scale motion planning," in *IEEE Inter. Conf. on Robotics and Automation*, Barcelona, Spain, 2005, pp. 3879–3884.

[30] ——, "Distributed computation of the knn graph for large high-dimensional point sets," *Journal of Parallel and Distributed Computing*, vol. 67, no. 3, pp. 346–359, 2007.