2011 IEEE/RSJ International Conference on
Intelligent Robots and Systems
September 25-30, 2011. San Francisco, CA, USA

Kinodynamic Motion Planning with State Lattice Motion Primitives

Mihail Pivtoraiko and Alonzo Kelly

Abstract— **This paper presents a type of motion primitives that can be used for building efficient kinodynamic motion planners. The primitives are pre-computed to meet two objectives: to capture the mobility constraints of the robot as well as possible and to establish a state sampling policy that is conducive to efficient search. The first objective allows encoding mobility constraints into primitives, thereby enabling fast unconstrained search to produce feasible solutions. The second objective enables high quality (lattice) sampling of state space, further speeding up exploration during search. We further discuss several novel results enabled by using such motion primitives for kinodynamic planning, including incremental search, efficient bi-directional search and incremental sampling.**

I. INTRODUCTION

There has been significant interest recently in developing *motion primitives*, specially designed controls that facilitate motion planning, particularly under kinematics and dynamics constraints on robot motion. The controls are developed during construction of the planner (*pre-computed*) and represent *feasible* motions, i.e. those that satisfy the constraints of the system. Motion planners can utilize these primitives to enact efficient search in state space by ignoring system constraints, instead focusing on the environment and other constraints – thereby improving efficiency of the planning. The role of primitives in planning and the importance of their quality have been motivated both in deterministic [8], [4], [24] and randomized [7] planning domains. Their importance was also noted in the related area of reactive obstacle avoidance in the context of mobile robot navigation [2], [9], [19]. A number of popular approaches to kinematic and kinodynamic planning can readily incorporate primitives in their design. The requisit *local planner* in [12], [14], [22] can be implemented as a process that chooses an appropriate element from a set of primitives [7]. In deterministic approaches [1], [3], [8], [5], [13], [24], the vertex expansion (set of edges emanating from a vertex) can be a pre-determined set of primitives based on the state value that the vertex represents.

Motion primitives have been designed in the past through sampling control space in such a way as to result in good sampling in state space in terms of discrepancy, dispersion or path diversity [2], [8], [4], [9], [19], [27]. We refer to this line of work as *control-sampling* primitives. In general, designing such primitives is difficult due to the complexity of the relationship between the robot's control and state spaces under kinematics and dynamics constraints. Motivated by this, we propose *state lattice* primitives, designed via a reverse process. First, we establish an attractive sampling rule in state space, perhaps one that is convenient and efficient for the planning problem (e.g. commensurate with the robot's world model, such as an occupancy grid). Then, we compute the controls that steer the system between these samples using a boundary value problem (BVP) solver. The approach can be viewed as a way of extending the Lazy LRM [21] to handle kinematics and dynamics constraints by leveraging the related research in BVP. Such solvers are available for a variety of systems, such as car-like [30], chained form [26], [27], as well as in rough terrain [11] and dynamics [17], [33] settings. A simple example of a set of car-like primitives in a three-dimensional (2D position and heading) state space is shown in Figure 1. A functional planner would require such a set for each of the 8 discrete values of heading, the multiples of $\pi/4$.

The benefits of this type of primitives are four-fold. First, by providing the freedom to choose an arbitrary sampling of state space for primitive endpoints, quality state sampling policies (low discrepancy and dispersion) may be utilized, leading to efficient exploration of state space during search. As Figure 1 illustrates, the resulting primitives may feature good path diversity as well. Second, under certain assumptions, such as flat and uniform terrain for the example above, this freedom allows designing primitives to be position-invariant. Experience with fielded applications demonstrated that position-invariance assumptions are often satisfied in practice, as long as a trajectory following controller absorbs external perturbations [29]; if necessary, trajectory post-processing may also be performed [20]. Once computed, position-invariant primitives can be utilized anywhere in search space, thereby moving integration of the controls to planner design phase. This is more efficient than affine invariance [8], since primitive transformation during search is limited to translation. Third, special reachability tree pruning rules can be easily designed. In contrast to control-sampling

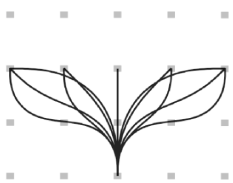

Fig. 1. The state lattice motion primitives are regularly arranged in state space. To design them, a convenient state sampling rule is chosen (e.g. a low discrepancy lattice), then a BVP solver is used to connect the samples via feasible motions.

This work was supported in part by the NASA Graduate Student Researchers Program Fellowship.

The authors are with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA {mihail, alonzo}@cs.cmu.edu

primitives, where primitive endpoints are dense in state space, state lattice ones yield a structure, where all paths leading to a region in state space also lead to a unique state value, as illustrated in Figure 2. This structure can be exploited to attain unprecedented search efficiency in the area of kinodynamic planning, including incremental search (a potential to speed up planning by orders of magnitude) [16], incremental sampling [21] and bi-directional search [18]. Finally, the freedom in state sampling may allow fitting the search space to the known structure of the environment. This strength of the approach was utilized recently to fit search spaces to such settings as parking lots [24], roads [32], mines [5] and indoor environments [31].

One drawback that may be experienced with the proposed primitives is the potentially significant computation that may be required to design this type of primitives (perhaps running the BVP solver repeatedly). However, this computation is off-line and does not affect the runtime of the planner. As another potential difficulty in certain applications, the constraint that the motions are arranged in a particular manner may conflict with other relevant objectives. For example, minimizing the length of primitives may be helpful for planning amidst dense obstacles, since shorter motions are less likely to be obstructed [4], [23]. Meeting such an objective may be more challenging if a constraint on endpoint arrangements is placed. Finally, even though the motions computed using lattice primitives are feasible and may be executed by the system *verbatim*, most physical systems suffer from inaccuracy in control leading to trajectory following error. Some applications may still require a trajectory-following controller, motivated above to satisfy position-invariance assumptions in rough terrain and similar scenarios. Significant disturbances in the environment, such as slopes or wind may be accounted for as additional state variables. The recommendation for a trajectory following controller does not offset the value of planning feasible motions, since non-feasible ones are more difficult or impossible to follow.

This paper provides a more general exposition of lattice primitives introduced in [28] and motivates them beyond field robotics [29]. It also proposes new applications of these primitives in incremental and randomized search (Section II), as well as *D* decomposition*, a novel algorithm to apply elements of D* search [16] to representation design, whereby near-minimal sets of lattice primitives are generated automatically (Section III). Experimental validation is discussed in Section IV.

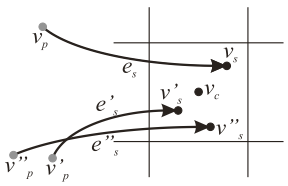

Fig. 2. State cell predecessors. Three control-sampling primitives, edges $\{e_s, e'_s, e''_s\}$, emanate from their corresponding predecessor vertices $\{v_p, v'_p, v''_p\}$ and arrive at successor vertices $\{v_s, v'_s, v''_s\}$.

II. Motion Planning with Lattice Primitives

In this section, we discuss the specifics of applying the state lattice primitives in planning using two prominent classes of search algorithms: deterministic (e.g. A*, D* [16] and their variants) and randomized (e.g. PRM [14], EST [12], RRT [22] and their variants). The planning problem is specified with a seven-tuple $(X, X_{free}, x_{init}, x_{goal}, U, f, c)$. The robot *state space*, $X \subset \mathbb{R}^n$, is an n-dimensional compact differentiable manifold, equipped with a metric ρ. $X_{free} \subseteq X$ is the set of states that satisfy global constraints (e.g. control bounds, obstacle avoidance, etc.). The boundary conditions for the planning problem are $x_{init} \in X_{free}$ and $x_{goal} \in X_{free}$. The set of robot controls U contains the inputs that the system accepts. The function f is the system model (equation of motion) and encodes kinematics and dynamics constraints: $\dot{x} = f(x,u)$, where $x \in X, u \in U$. The function $c : U \times X \to \mathbb{R}$ specifies the cost of executing a control $u \in U$ in X. The solution to the planning problem is a control $u_s : [t_0, t_f] \to U$, where t_0 is the starting time and t_f is the final time, such that $c(u_s, x_{init})$ is minimized. The corresponding path $\pi_s : [t_0, t_f] \to X_{free}$ (obtained by integrating $f(x_{init}, u_s)$) satisfies $\pi_s(t_0) = x_{init}$ and $\pi_s(t_f) = x_{goal}$.

Finding the exact solution involves optimization over the continuum of X and U, a difficult problem because of obstacles and local optima in X. Instead, it is common to establish pruning rules that reduce the system's reachability in X and U to discretized representations, often structured as graphs. We assume a directed graph $G = V \cup E$, where V is a set of vertices, representing samples in X, and E is a set of edges, representing samples in U. Each edge is one of the pre-computed, feasible primitives. The dimensionality of V is chosen so that a concatenation of edges is also a feasible motion. The least-cost path in the graph is the solution to the planning problem.

A. Deterministic Search

The strengths of deterministic, exhaustive search include attractive guarantees, such as optimality (under certain conditions, such as heuristic admissibility) and resolution-completeness. One drawback, however, is the so-called "curse of dimensionality", the exponential growth of complexity with dimension of the search space. Nevertheless, this search technique remains attractive for systems that can be modeled well in a few dimensions, including car-like [24], tracked [5], flying [8] and other systems of practical interest.

Such approaches to deterministic nonholonomic planning typically guarantee feasibility by elaborating the primitives $e_s \in E$ by choosing a control $u_s \in U$ and integrating $f(v_p, u_s)$ for a certain Δt. The successor vertex v_s is established at the endpoint of e_s. Since, in general, such edges and their endpoints will be dense in X, such planners attempt to prune the edges that are very similar, redundant or otherwise do not contribute to efficient exploration of X. For example, if the endpoint of a certain edge e_s is a vertex v_s, then a second edge e'_s terminating in v'_s is discouraged if distance $\rho(v_s, v'_s)$ is small. To this end, these approaches establish a discretization of X into cells, as shown in Figure 2. If

a cell contains v_s, then e'_s is ignored if v'_s would occupy the same cell. The proposed lattice primitives may be used identically, except that the need to detect similar edge endpoints is eliminated, since all motions are designed to arrive at specific state values, e.g. v_c in Figure 2. A similar pruning rule is in effect in this setting, except that it is developed offline during search space design.

Well-informed search heuristics have the potential to increase the efficiency of the search substantially. Developing good heuristics for planning with differential constraints is a challenging problem, and various approaches are being developed [5], [6]. By position invariance of lattice primitives, applications are enabled to pre-compute the free-space costs of motions, stored in a look-up table, leading to the perfect heuristic in terms of mobility constraints [15], [24].

1) Incremental Search: In many applications featuring physical robots, it is beneficial to perform incremental search: once a plan is computed, it is efficiently modified (by reusing previous computation) should new information about the environment invalidate it [16]. This enables the planner to react quickly to frequent changes of the world model, including those due to uncertainty and noise of the perception, localization and other systems. This type of search is a standard component in many fielded robotics systems, since plan repair can be vastly more efficient than replanning from scratch.

The reachability pruning schemes above, necessary for control-sampling primitives, are not fully compatible with incremental search. The operation of such search requires an ability to enumerate the edges that lead to a particular state, e.g. the edges $\{e_s, e'_s, e''_s\}$ leading to a cell containing v_s in Figure 2. A key component of incremental search algorithms is the *rhs*-value, the one-step lookahead cost value, that is defined as [16]:

$$rhs(v_s) = \begin{cases} 0 & \text{if } v_s = x_{init} \\ \min_{v_p \in Pred(v_s)} \left(g(v_p) + c(v_p, v_s) \right) & \text{otherwise} \end{cases} \quad (1)$$

where $g(v_p)$ denotes the cost of the vertex v_p, $Pred(v_s)$ is the set of the predecessors of v_s, e.g. $\{v_p, v'_p, v''_p\}$ in Figure 2. One of the ways of reusing previous computation is the capacity to pick a different predecessor of a state in the event that the current predecessor edge increases its cost. However, as depicted in Figure 2, the predecessor edges cannot be swapped in general because the distance between their endpoints $\rho(v_s, v'_s) \neq 0$ (by their density). This issue is resolved with lattice primitives: this distance is zero, since all successors converge to the identical v_c. Note that this is similar to traditional applications of incremental search in grids.

Moreover, in grids, it is typically easy to determine the set of cells (and, consequently, edges) that are affected when a region in workspace changes cost. With arbitrary motion primitives, this computation is more involved, since the edges may span several cells. As discussed in [29], this computation can be done *a priori* by virtue of the lattice structure. Once we compute the *swaths* of all primitives, we collect and store a list of edges that pass through the origin cell. By position invariance, this list may be reused anywhere in the search space.

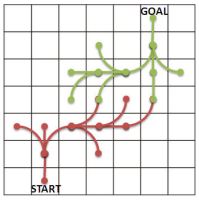

Fig. 3. Bidirectional search with lattice primitives. The BVP problem of connecting the leaves of the two trees is eliminated with lattice primitives that feature regularity of endpoints in state space.

2) Incremental Sampling: The advent of randomized planning in recent years has spurred inquiry into effective sampling methods that avoid the "curse of dimensionality" while offering better solution quality and completeness guarantees than standard randomized approaches. Deterministic incremental sampling techniques have been proposed as viable alternatives [21]. One of them is the Halton points, a d-dimensional generalization of the van der Corput sequences of d bases, one for each coordinate [10]. A basic version of such incremental sampling in square grids can be thought of as increasing discretization resolution by a factor of 2^{di}, $i \in \mathbb{N}$.

The planners that do not enforce structure in edge connectivity would be required to regenerate the plan from scratch every time the sampling resolution is incremented. However, lattice primitives enable the reuse of previous computation via the same mechanism that allows incremental search. Due to regular structure, the connections between primitives belonging to different resolution levels become trivial, thereby allowing the results of planning at different resolution to be reused. One application of this approach is *anytime* planning, where the quality of the computed plan is improved with more computation.

B. Randomized Search

Lattice primitives may be utilized in randomized search in a manner that is similar to other types of primitives [7]. As suggested in Section I, the local planner component in [14], [12], [22] and similar planners may be designed to choose an element of a set of primitives that is a good fit to extend the tree or the graph towards a random sample. However, by virtue of the regular structure of the state samples, lattice primitives would enable additional capacity to execute parallelized kinodynamic planning, e.g. bi-directional [18], as well as a series of independent searches [25]. Figure 3 illustrates that, unlike control-sampling primitives that would likely require multiple BVP solutions to connect partial planning results, the layout of lattice primitives makes this connection automatic.

III. Designing Lattice Primitives

Having motivated the proposed type of primitives, we suggest a principled approach to designing them given the

planning problem formulated in Section II. Assuming a system model and a corresponding BVP solver, we discuss this design in three stages: making a choice of the dimensions to include in the state space representation, developing the sampling rule in that state space, and designing a compatible, near-minimal set of lattice primitives that is a good representation of the system's reachability. The first two stages (Section III-A) determine the set of controls $U_l \subset U$ that can possibly be represented with such primitives. Generally, this set is infinite; Section III-B is dedicated to developing a near-minimal primitive set $E_a \subset U_l$ that, when used as the vertex expansion in search, will reconstruct a good approximation to U_l. More precisely, a planner, based on optimal (exhaustive) search and equipped with E_a, will be able to compute the motions in U_l (preserve completeness) and will guarantee bounds on suboptimality of these motions w.r.t. U_l. An algorithm that computes E_a automatically is presented.

A. State Space Sampling

In general, the problem of selecting the minimal number of dimensions that adequately represent the planning problem is quite challenging. In the case of designing lattice primitives, this issue is influenced by the choice of the BVP solver. In case the solver does not fix the dimensionality, an iterative dimensionality reduction process may be undertaken. Once a set of lattice primitives is designed at the highest dimensionality, it may be repeated with one of the dimensions removed. The process iterates until the loss of representation quality exceeds application tolerances.

Once state dimensionality is fixed, we develop a state sampling rule using two principles. First, it is beneficial for the sampling rule to minimize discrepancy or dispersion [21]. Grids and similar regular lattice structures typically minimize these measures, and they are frequently used in this setting. Second, among similarly performing search spaces, those with more coarse sampling are preferred. This is an Occam's razor statement: a simpler approach is likely to lead to a solution that is easier to develop and test. Since controls are induced by state sampling in this setting, it is beneficial to choose state samples that reduce the cost of controls, e.g. lead to a greater number of straight-line motions. The above principles are purposefully broad: each application imposes specific requirements on state sampling. For example, sampling of position and orientation variables of robots is often closely related to other design specifications, such as the fidelity of perception information and control accuracy of the vehicle.

B. Primitive Set Decomposition

Even though the representable set of controls U_l is infinite, the reachability of many systems of practical interest can be captured well by analyzing a finite, albeit very large, subset $\hat{U}_l$. For example, for car-like robots, we could define $\hat{U}_l$ as the set of motions that are contained in a region (centered around the robot) that is much larger in extent than the robot's minimum turning radius. This motion set will include many maneuvers that the robot is capable of executing, including multi-point turns in close quarters. Next we develop an explicit and exact representation of $\hat{U}_l$ as a graph $\hat{G}_l = \hat{V}_l \cup \hat{E}_l$, as justified in Section II.

By the given lattice state sampling rule, the set $\hat{V}_l$ is known. Theorem 1 develops a primitive set E_O that generates $\hat{E}_l$, a superset of $\hat{U}_l$, when used as the Dijkstra's vertex expansion; free space is assumed below, unless otherwise noted. More precisely, E_O is a set of primitive sets defined for all possible trajetory initial states in $\hat{V}_l$, up to the invariant dimensions (e.g. position). For example, different values of heading in Figure 1 would require different vertex expansions; E_O can be viewed as the union of the corresponding primitive sets.

Theorem 1: Suppose origin vertices $O \subset \hat{V}_l$ are chosen (up to invariant dimensions). For every vertex $v_i \in \hat{V}_l$, an edge from each element of O to v_i is computed using the BVP solver and added to E_O (initially empty). When used as the Dijkstra's vertex expansion, E_O will search (equivalently, generate) a $\hat{G}_l$ such that $\hat{E}_l \supseteq \hat{U}_l$.

Proof: First, by construction, we conclude that E_O contains $\hat{V}_l$ as endpoints of its primitives. Using E_O as the Dijkstra's vertex expansion amounts to replicating its edges at every $v_i \in \hat{V}_l$. If, per connectivity of $\hat{U}_l$, a certain v_i connects to a set of vertices $\{v_j\}$, then E_O, when replicated at v_i, will connect it to at least the same vertices, since $\{v_j\} \subseteq \hat{V}_l$. Thus, the process of replicating E_O at $v_i \in \hat{V}_l$ generates at least the edges present in $\hat{U}_l$, and therefore the induced set of edges $\hat{E}_l \supseteq \hat{U}_l$. ■

Using E_O as the vertex expansion represents an extreme of quality-complexity trade-off. The cost of the capacity to explore at least all of $\hat{U}_l$ during search is a very large branching factor, $|E_O|$. Next we discuss an approach to manage this trade-off by computing an approximation primitive set $E_a \subset E_O$ of much smaller cardinality, while guaranteeing bounded suboptimality of computable motions w.r.t. $\hat{U}_l$ in terms of arbitrary notion of cost.

This process attempts to decompose each motion in $\hat{E}_l$ into two or more other motions that are also in $\hat{E}_l$. Decomposition (Figure 4) is allowed only if the concatenation of the components is within a user-specified threshold on cost increase, defined as a cost ratio $c_t > 1$ of the concatenated motion vs. the original one:

$$g(v_p) + c(v_p, v_s) \leq c_t g(v_s) \tag{2}$$

The component motions that can be reused to generate other motions are collected into E_a. E_a is designed to be capable of generating every one of the motions in $\hat{E}_l$, within the specified cost increase threshold.

A brute-force approach to decomposing $\hat{E}_l$ by enumerating all possible motion decompositions would be exponential in $\left|\hat{E}_l\right|$ and therefore prohibitively expensive. We propose two greedy algorithms that produce near-minimal E_a, given $\hat{E}_l$ and c_t.

1) Leave-one-out Decomposition: This algorithm decomposes the motions in $\hat{E}_l$ in decreasing order of their cost. This implements a heuristic on managing the dependencies

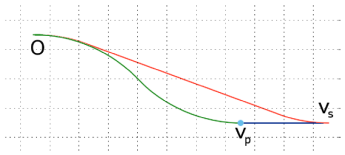

Fig. 4. Motion decomposition. A motion is approximated by a concatenation of two shorter motions. The ratio of their combined cost to the original cost is constrained not to exceed a user-specified threshold.

of component motions, namely that the higher-cost motions are likely to be decomposed by lower-cost motions. Each motion e_s to be decomposed is selected and removed from $\hat{E}_l$. Next, optimal A* search is used to compute a motion from O to the endpoint of e_s using vertex expansion $\hat{E}_l \setminus \{e_s\}$. If the least cost motion represents a cost increase greater than c_t, the e_s is reinserted into $\hat{E}_l$, since it cannot be decomposed satisfactorily. Otherwise, it is left out of $\hat{E}_l$, and the algorithm repeats by selecting the next motion to decompose. Very large values of c_t will lead to a degeneracy where almost all motions are decomposed. This condition can be detected by observing the resulting E_a.

2) D Decomposition:* The previous algorithm has a disadvantage in complexity: it requires running A* "from scratch" $\left|\hat{E}_l\right|$ times, in graphs with large (albeit reducing) branching factors. Inspired by the capacity of incremental search algorithms, e.g. D* Lite [16], to prevent repetitive "from scratch" search by virtue of reusing computation, we propose an alternative with much better runtime (Algorithm 1). It does a single Dijkstra's elaboration of E_O vertex expansion to explore the extent of $\hat{U}_l$ (line 2) and then performs the decomposition analysis from Section III-B.1 while reusing the collected vertex predecessors, similar to evaluating the *rhs*-value (1). In this manner, unlike the traditional application in planning, the principles of the D* algorithm are used here for search space design.

Once the predecessors are computed, only those that can possibly yield c_t-decompositions are retained (lines 3-7). If a state has only a single predecessor, it implies by construction that the one and only motion that connects it to the origin is the original edge in $\hat{E}_l$, and motion decomposition cannot succeed. This edge is entered into E_a. Since the motions that form decompositions may be further decomposed themselves, we keep track of per-vertex cost thresholds with a database *db*; thresholds for all vertices are initially set to c_t on line 5.

Line 8 follows the cost-sorting heuristic discussed in Section III-B.1. The next line selects edges in decreasing order of cost of their destination endpoints. Lines 10-12 re-evaluate the number of admissible predecessors. Line 13 enumerates the options for decomposing the edge e_s leading to v_s. The algorithm attempts to select the decomposition that is likely to minimize the final E_a. To this end, it uses two other heuristics. The first one is sorting potential decompositions in increasing order of cost (sort on line 13), in an attempt to choose a decomposition with cumulative cost that is as close as possible to the original motion. The

Input: finite control set $\hat{E}_l$, cost threshold c_t
Output: approximating control set E_a

1 $E_a = \emptyset$;
2 Run Dijkstra's search with $\hat{E}_l$, starting at O;
3 **foreach** $e_s \in \hat{E}_l$ **do**
4 $\quad Pred_{c_t}(v_s) = \{v_p, \text{ s.t. } g(v_p) + c(v_p, v_s) \leq c_t g(v_s)\}$;
5 $\quad db(v_s) = c_t$;
6 $\quad$ **if** $|Pred_{c_t}(v_s)| = 1$ **then**
7 $\quad\quad E_a = E_a \cup e_s$;
8 $\quad$ **end**
9 **end**
10 $E_{sort} = \text{sort}(\hat{E}_l)$;
11 **foreach** $e_s \in E_{sort}$ **do**
12 $\quad$ **if** $|Pred_{c_t}(v_s)| = 1$ **then**
13 $\quad\quad E_a = E_a \cup e_s$;
14 $\quad\quad$ continue;
15 $\quad$ **end**
16 $\quad$ **foreach** $v_p \in sort(Pred_{c_t}(v_s))$ **do**
17 $\quad\quad$ **if** $e_{v_p,v_s} \in E_a$ **then**
18 $\quad\quad\quad$ adjust_threshold(v_p, v_s);
19 $\quad\quad\quad$ goto 9;
20 $\quad\quad$ **end**
21 $\quad$ **end**
22 $\quad v_p^* = \underset{v_p \in Pred_{c_t}(v_s)}{\text{argmin}} \; g(v_p) + c(v_p, v_s)$;
23 $\quad$ adjust_threshold(v_p^*, v_s);
24 $\quad E_a = E_a \cup e_{v_p^*, v_s}$;
25 **end**

Algorithm 1: D* Decomposition.

Input: vertices v_p and v_s
Output: $db(v_p)$, $Pred_{c_t}(v_p)$, and E_{sort} modified

1 $\alpha = (g(v_p) + c(v_p, v_s))/g(v_s)$;
2 $c_t' = \frac{db(v_s)g(v_s) - c(v_p,v_s)}{\alpha g(v_s) - c(v_p,v_s)}$;
3 **if** $c_t' < db(v_p)$ **then**
4 $\quad db(v_p) = c_t'$;
5 $\quad Pred_{c_t}(v_p) = \{v_p', \text{ s.t. } g(v_p') + c(v_p', v_p) \leq db(v_p)g(v_p)\}$;
6 $\quad$ **if** $g(v_p) \geq g(v_s)$ **then**
7 $\quad\quad E_{sort} = E_{sort} \cup e_p$;
8 $\quad$ **end**
9 **end**

Algorithm 2: adjust_threshold function.

second heuristic is preferring a decomposition, in which one of the segments is already found in E_a (lines 14-16), in an attempt to avoid adding new elements to E_a.

Once an edge is decomposed into two segments connecting at vertex v_p, the segment leading to v_p may in turn be decomposed; however the cost threshold for its decomposition may have to be different from c_t. To see this reasonsing, suppose path length is taken as cost (Figure 4). Using the cost sorting heuristic, the longer, original motion (terminating in v_s) will be decomposed before the first segment of decomposition (terminating in v_p). When

the latter is in turn selected for decomposition, we rewrite (2) as:

$$c_t' g(v_p) + c(v_p, v_s) \leq c_t g(v_s) \tag{3}$$

Depending on the value of $c(v_p, v_s)$, the required cost increase threshold c_t' for the recursive decomposition of v_p may be $c_t' < c_t$. Thus, as soon as a decomposition relationship is established between the vertices, the necessary reduction of cost threshold of the predecessor vertex is computed with the function *adjust_threshold*. If the algorithm gets to line 17, no decomposition opportunities have been selected by the heuristics. This line selects a decomposition with the least-cost predecessor; it is added to E_a on line 19.

Algorithm 2 presents the *adjust_threshold* function. In line 2, it computes c_t', the new cost threshold for the predecessor vertex v_p. If this value is less than the previous one, it is recorded on line 4, and the list of admissible predecessors is reviewed on line 5. Lines 6-7 address the case where the cost sorting heuristic is incorrect, and the predecessor v_p has equal or greater cost than v_s. In this case, any changes done to v_p need to be propagated to its potential predecessors, so the next iteration of Algorithm 1 must select v_p's edge, e_p (line 3).

IV. Experimental Results

We present experimental validation results of kinodynamic planning using two examples: a double integrator system inspired by [3] and a car-like system with complex dynamics. We also describe a multi-query BVP approach (Section IV-B) that is helpful for implementing the second example, presented in Section IV-C. Both examples were developed by applying the design principles in Section III (Figure 6). The primitives, developed with Algorithm 1, are then used to perform incremental search by utilizing the unmodified D* Lite algorithm [16] (Figure 7).

A. Double Integrator

The *double integrator* is a simple dynamics system, where the acceleration is controlled directly. Formally, it is governed by a second-order differential equation $\ddot{x} = u$, where $x, u \in \mathbb{R}$. We look at a one-dimensional example here, although more dimensions, similarly defined, can be added easily due to their independence. The state transition equation $\dot{x} = f(x,u)$ can be written as $(\dot{x}_1, \dot{x}_2) = (x_2, u)$. Thus, the terminal states of a trajectory, x_{init} and x_{goal} are specified by the corresponding positions and velocities. A discrete-time model with controls given by $u = \{-1, 0, 1\}$ has reachability tree trapped on a lattice (Figure 5a).

To illustrate the application of Algorithm 1 to this system, consider an example in Figure 5b. The system starts at zero position and zero velocity and attempts to reach a specified position value with zero terminal velocity. The trajectory in this example is the sequence of black arrows in part b) of the figure. Part c) shows a few of the elements of the set $\hat{E}_l$ for this example; the whole set is prohibitively large to illustrate. Given this input, the algorithm does Dijkstra's (line 2), which generates the sets of admissible predecessors of the vertices.

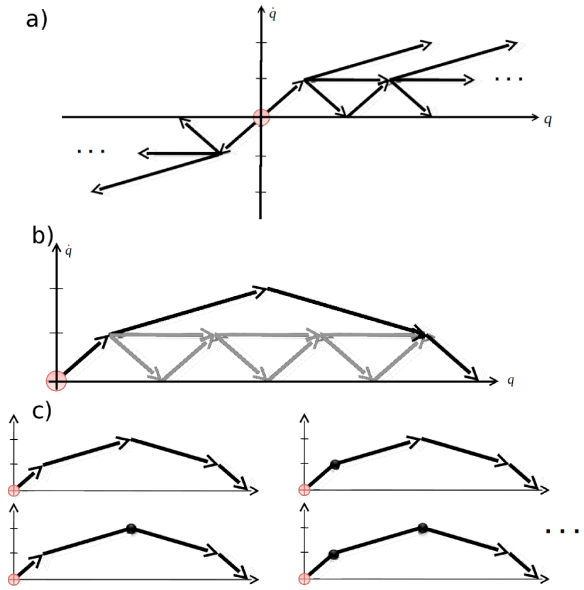

Fig. 5. Double integrator system example.

This set for the goal state, $Pred_{c_t}(x_{goal})$ is shown in gray in Figure 5b. Since the edges between all the states in this set come from the original selection of edges allowed in this problem setup, we observe that the solution E_a contains the same set of edges we started with (Figure 5a). In the case of this simple problem, the algorithm is able to find the optimal solution (that is, resulting c_t ratio is 1.0), however it is usually not the case for more realistic problems, as suggested by the following sections. The solution path (black line in part b) of the figure) can be constructed with the replanning search algorithms we consider here.

B. Multi-Query BVP Approach via Reachability Analysis

Since the BVP solver was not available for this system, we utilized an alternative approach; it is described here because it illuminates the experimental setup. First, a dense reachability tree of the system was generated via significant computation using high-resolution, regular sampling in two-dimensional control space, consisting of angular wheel velocity and the steering angle. A reachability pruning technique [1], [8], [13] was utilized via high-resolution, regular sampling in seven-dimensional state space, consisting of 2D position, heading, steering angle, longitudinal, lateral and angular velocities of robot body. The generated reachability was analyzed to engineer appropriate dimensionality of representation. It was observed that the lateral and angular velocity dimensions could be dropped without a significant loss of representation quality: the subset of reachability with these variables considered to be zero (per pruning resolution) was henceforth considered. This reasoning was motivated by dimensionality reduction considerations in Section III-A.

A sampling scheme of the remaining five dimensions was developed via an application-minded approach. First, the chosen state sampling resolution was lower than the reachability resolution by an integer multiple. Since minimum turn-

ing radius was small compared to vehicle size, the position was sampled as a grid with cell size equal to approximately half of min. turning radius. Heading was sampled in 16 non-uniform values: half were multiples of $\pi/4$, and the other half were related to $\arctan(1/2)$ to maximize straight paths toward nearby cells. Only extremal values of steering angle (including 0) and longitudinal velocity (three and two, respectively), were chosen in order to explore the envelope of the system dynamics. This sampling rule generated another, yet smaller, subset of system reachability, denoted U_l in Section III-B. Since the state resolution was a multiple of the reachability pruning resolution, the endpoints of the primitives were close to their respective state cell centers. Those that were not sufficiently close were improved via gradient descent optimization by treating the durations of control space samples, comprising the trajectory, as variables and the distance of the end-point to cell center as the objective. This was a significant computation, since gradient estimation involved repeated execution of the physics simulation.

C. Car-like Robot with Dynamics

The system in this example is a wheeled robot with three driven wheels, one of which steers, as shown in Figure 7b. The system is simulated using the Open Dynamics Engine^{TM}software; the system model is not available in closed form. The vehicle has significant mass, is capable of achieving high speeds and is placed on a very slippery flat surface to highlight the effects of dynamics, such as significant drift, sliding sideways, etc.

A comparative study of lattice and high-diversity primitives [2], [4], based on A* search, showed that the difference in performance in terms of runtime, solution quality and completeness of both types of primitives is less than statistically significant.

The benefit of the principled approach here is that the length of primitives is selected automatically, while it is fixed for path diversity primitives.

Next we describe experimental validation of incremental search in this context, although other planning approaches may benefit from such primitives, as discussed in Section II. Figure 7 illustrates an example where a new obstacle invalidated a segment of the previously computed trajectory. D* modified the plan efficiently by limiting vertex expansions to a small neighborhood. The initial plan was computed in 1.42 seconds (on commodity hardware), and was repaired in 0.35 seconds – a nearly 4-fold speedup with respect to re-planning from scratch.

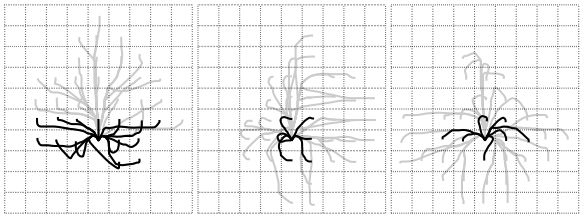

Fig. 6. Automatic pruning of primitive sets. Several subsets of primitives, generated for the given system, are shown; abundance of high curvature is due to extreme dynamics. Top and bottom rows include primitives at low and high velocity, resp. The columns show the primitives with final headings 0°, 90° and 180° (left to right). Gray motions have been pruned, as concatenations of black motions can replicate them within specified cost increase threshold.

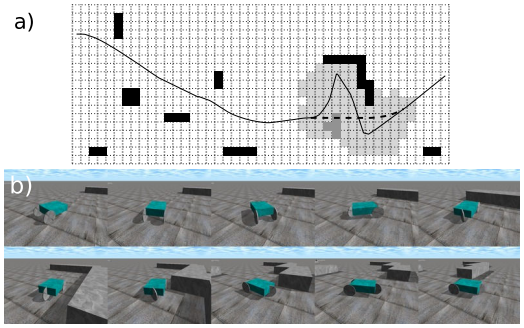

Fig. 7. Kinodynamic incremental planning. Robot is avoiding a number of obstacles (black cells), while traveling at high speed on slippery surface. A new obstacle (dark gray cells) is discovered and invalidates a segment of the previous trajectory (dotted line). D* repairs the path by expanding states (light gray) only in the affected region.

V. Conclusions and Future Work

We discussed a type of primitives that is designed via regular sampling in state spaces. These primitives are precomputed to meet two objectives: to capture the mobility constraints of the robot as well as possible and to establish a state sampling policy that is conducive to efficient search. The first objective allows encoding mobility constraints into primitives, thereby enabling fast unconstrained search to produce feasible solutions. The second objective enables high quality (lattice) sampling of state space, further speeding up exploration during search. We further discuss several novel results enabled by using such primitives for kinodynamic planning, including incremental, bi-directional search and incremental sampling. Future work includes identifying new state and control sampling techniques that further improve properties of planning in deterministic and randomized domains.

VI. Acknowledgments

The authors gratefully acknowledge the contribution of the NASA Graduate Student Researchers Program.

References

[1] Jerome Barraquand and Jean-Claude Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10(2-4):121–155, October 1993.

[2] M. S. Branicky, R. A. Knepper, and J. J. Kuffner. Path and trajectory diversity: Theory and algorithms. In *Proc. IEEE International Conference on Robotics and Automation ICRA 2008*, pages 1359–1364, 2008.

[3] Bruce Donald, Patrick Xavier, John Canny, and John Reif. Kinodynamic motion planning. *J. ACM*, 40(5):1048–1066, 1993.

[4] L. H. Erickson and S. M. LaValle. Survivability: Measuring and ensuring path diversity. In *Proc. IEEE International Conference on Robotics and Automation ICRA '09*, pages 2068–2073, 2009.

[5] Xiuyi Fan, Surya Singh, Florian Oppolzer, Eric Nettleton, Ross Hennessy, Alexander Lowe, and Hugh Durrant-Whyte. Integrated planning and control of large tracked vehicles in open terrain. In *Proceedings of the International Conference on Robotics and Automation*, 2010.

[6] T. Fraichard and A. Scheuer. From reeds and shepp's to continuous-curvature paths. *IEEE Transactions on Robotics*, 20(6):1025–1035, December 2004.

[7] Emilio Frazzoli, Munther A. Dahleh, and Eric Feron. Real-time motion planning for agile autonomous vehicles. *JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS*, 25(1), 2002.

[8] Jared Go, Thuc D. Vu, and James J. Kuffner. Autonomous behaviors for interactive vehicle animations. *Graph. Models*, 68(2):90–112, 2006.

[9] Colin Green and Alonzo Kelly. Toward optimal sampling in the space of paths. In *13th International Symposium of Robotics Research*, 2007.

[10] J.H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2(1):84–90, December 1960.

[11] T.M. Howard and A. Kelly. Optimal rough terrain trajectory generation for wheeled mobile robots. *International Journal of Robotics Research*, 26(2):141–166, 2007.

[12] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. Conf. IEEE Int Robotics and Automation*, volume 3, pages 2719–2726, 1997.

[13] Sören Kammel, Julius Ziegler, Benjamin Pitzer, Moritz Werling, Tobias Gindele, Daniel Jagzent, Joachim Schder, Michael Thuy, Matthias Goebl, Felix von Hundelshausen, Oliver Pink, Christian Frese, and Christoph Stiller. Team annieways autonomous system for the darpa urban challenge 2007. *Journal of Field Robotics*, 25(9):615 – 639, August 2008.

[14] 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*, 12(4):566–580, 1996.

[15] R. A. Knepper and A. Kelly. High performance state lattice planning using heuristic look-up tables. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3375–3380, 2006.

[16] Sven Koenig and Maxim Likhachev. D*-Lite. In *Eighteenth national conference on Artificial intelligence*, pages 476–483, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[17] J. Zico Kolter, Christian Plagemann, David T. Jackson, Andrew Y. Ng, and Sebastian Thrun. A probabilistic approach to mixed open-loop and closed-loop control, with application to extreme autonomous driving. In *Proceedings of the International Conference on Robotics and Automation*, 2010.

[18] Jr. Kuffner, J. J. and S. M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proc. IEEE Int. Conf. Robotics and Automation ICRA '00*, volume 2, pages 995–1001, 2000.

[19] Manfred Lau and James J. Kuffner. Behavior planning for character animation. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 271–280, New York, NY, USA, 2005. ACM.

[20] J.-P. Laumond, S. Sekhavat, and F. Lamiraux. Guidelines in nonholonomic motion planning. *Robot motion planning and control*, 1998.

[21] Steven M. LaValle, Michael S. Branicky, and Stephen R. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *International Journal of Robotics Research*, 23(7-8):673–692, 2004.

[22] Steven M. LaValle and Jr. James J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, 2001.

[23] Peter J. Leven. *A framework for real-time path planning in changing environments*. PhD thesis, University of Illinois, Champaign, IL, USA, 2001. Adviser-Hutchinson, Seth.

[24] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *International Journal of Robotics Research*, 28(8):933–945, August 2009.

[25] Maxim Likhachev and Anthony Stentz. R* search. In *AAAI*, 2008.

[26] R. M. Murray and S. S. Sastry. Nonholonomic motion planning: steering using sinusoids. *IEEE Transactions on Automatic Control*, 38(5):700–716, May 1993.

[27] S. Pancanti, L. Pallottino, D. Salvadorini, and A. Bicchi. Motion planning through symbols and lattices. In *Proc. IEEE Int. Conf. Robotics and Automation ICRA '04*, volume 4, pages 3914–3919, 2004.

[28] Mihail Pivtoraiko and Alonzo Kelly. Generating near-minimal spanning control sets for constrained motion planning in discrete state spaces. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3231–3237, August 2005.

[29] Mihail Pivtoraiko, Ross A. Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

[30] J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 145(2):367–393, 1990.

[31] Martin Rufli, Dave Ferguson, and Roland Siegwart. Smooth path planning in constrained environments. In *Proc. IEEE Int. Conf. Robotics and Automation ICRA '09*, pages 3780–3785, 2009.

[32] Martin Rufli and Roland Siegwart. On the design of deformable input-/ state-lattice graphs. In *Proceedings of the International Conference on Robotics and Automation*, 2010.

[33] Yuval Tassa, Tom Erez, and William D. Smart. Receding horizon differential dynamic programming. In *Proceedings of the Neural Information Processing Systems Conference*, 2007.