Learning Dimensional Descent for Optimal Motion Planning in High-dimensional Spaces

Paul Vernaza
GRASP Laboratory
University of Pennsylvania
Philadelphia, PA 19104

Daniel D. Lee
GRASP Laboratory
University of Pennsylvania
Philadelphia, PA 19104

Abstract

We present a novel learning-based method for generating optimal motion plans for high-dimensional motion planning problems. In order to cope with the curse of dimensionality, our method proceeds in a fashion similar to block coordinate descent in finite-dimensional optimization: at each iteration, the motion is optimized over a lower dimensional subspace while leaving the path fixed along the other dimensions. Naive implementations of such an idea can produce vastly suboptimal results. In this work, we show how a profitable set of directions in which to perform this dimensional descent procedure can be learned efficiently. We provide sufficient conditions for global optimality of dimensional descent in this learned basis, based upon the low-dimensional structure of the planning cost function. We also show how this dimensional descent procedure can easily be used for problems that do not exhibit such structure with monotonic convergence. We illustrate the application of our method to high dimensional shape planning and arm trajectory planning problems.

Keywords: motion and path planning, robotics, robot arm planning, holonomic motion planning, optimal control, subspace representations

1 Introduction

In this work, we consider the problem of optimal motion planning for high-dimensional holonomic systems. Example instances of such problems include numerous problems in robotics and control, including planning for mobile manipulation, teams of robots, modular robots, and various other highly articulated systems.

Recent years have witnessed great strides in our ability to solve such problems in complex spaces, largely due to the rise of sampling-based motion planning in such variants as RRT (Kuffner and LaValle 2000), PRM (Kavraki et al. 1996), SBL (Sánchez and Latombe 2001), and EST (Hsu, Latombe, and Motwani 1999). Meanwhile, progress has also been made in deterministic planning algorithms, most of which are descended from the classic A* algorithm. Variants such as ARA* (Likhachev, Gordon, and Thrun 2004) and R* (Likhachev and Stentz 2008) have also been successfully applied to complex motion planning problems.

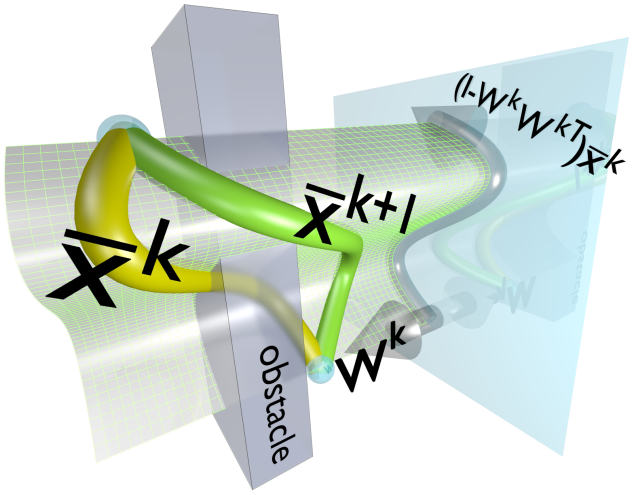


Figure 1: Illustration of one step of dimensional descent algorithm. Given initial path $\bar{x}^k$, the next path $\bar{x}^{i+1}$ is the optimal path constrained to lie on a manifold obtained by sweeping $\bar{x}^k$ along the learned direction W^k.

We present here a novel approach based upon a different way of thinking about such problems. Rather than devising more clever ways to efficiently sample high-dimensional spaces, as with the sampling-based planning methods; or exploring more efficient combinatorial algorithms, as with the deterministic planning methods; our approach focuses on automatically learning as much as possible about the structure of a specific planning problem, and subsequently exploiting that structure to solve the problem efficiently.

In particular, we show how to implement this idea to find optimal trajectories in high dimensional motion planning problems, via a method we refer to as Learning Dimensional Descent (LDD) At a high level, LDD is quite simple: we merely optimize the trajectory one dimension at a time, after transforming the problem into a set of coordinates that are learned automatically from the structure of the problem.

This procedure is illustrated in Fig. 1 for a block size $d = 1$; i.e., in each iteration, we optimize the path in only one of the learned directions W^k, while holding fixed the projection of the current path onto the other directions. Each iteration of LDD in this case therefore only requires the so-

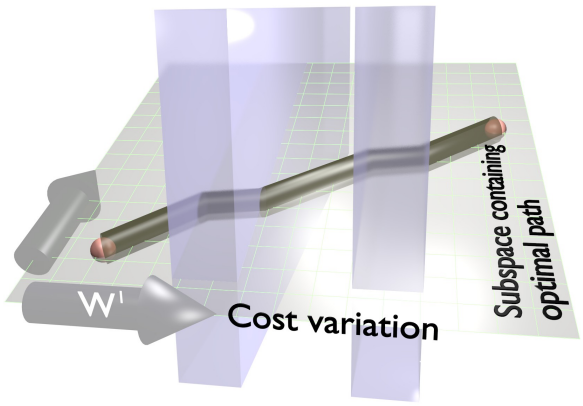


Figure 2: Illustration of Theorem 2.1. Shown is an optimal path in three dimensions of a cost function varying only in one dimension. Vertical blocks represent high-cost regions assumed to extend ad infinitum in directions orthogonal to the cost variation direction. The optimal path lies in a two-dimensional subspace: the smallest subspace that contains both the direction of cost variation and the endpoints.

lution of a simple two-dimensional dynamic programming problem. Crucially, we will show that it is possible to learn the basis directions W^k to ensure that LDD terminates with an optimal solution in one iteration, if the cost function varies only in a low-dimensional space. This is because the first iteration of LDD searches the subspace guaranteed to contain the optimal path for such problems, as illustrated in Fig. 2.

1.1 Related work

Our work can be motivated by the desire to break down a complex motion planning problem into simpler subproblems. In this sense, it is preceded by methods such as (Brock and Kavraki 2001), which generate a plan in high-dimensional configuration space given a plan in a low-dimensional workspace. Instead of performing a strict decomposition, many recent randomized planning methods have employed a hybrid approach, using low-dimensional auxiliary spaces to aid in creating a sort of sampling bias towards the most important degrees of freedom (Shkolnik and Tedrake 2009; Diankov et al. 2008; Berenson et al. 2009; Sucan and Kavraki 2008). Exemplary among these are BiSpace Planning (Diankov et al. 2008) and Task-space RRT (Shkolnik and Tedrake 2009), which both grow trees in a low-dimensional, user-defined workspace or *task space*.

To our knowledge, however, comparatively little work has been devoted to automatically identifying these interesting low-dimensional subspaces, which is the key idea of LDD. One notable exception is (Dalibard and Laumond 2009), which uses PCA to determine directions in which to expand an RRT. LDD, however, applies to the more general optimal motion planning problem.

LDD is comparable in this sense to RRT* (Karaman and Frazzoli 2010), which is asymptotically optimal. Unlike RRT*, LDD has a concrete computational complexity bound—i.e., exponential in the sampling resolution and the *dimension of the cost function* (a term defined loosely for now). If this dimension is low, (e.g., less than 5) and the sampling resolution reasonable, LDD can be used in practice to find globally optimal solutions, *independent of the dimension of the configuration space*. If the dimension of the cost function is only approximately low, LDD is still an approximation algorithm with provable suboptimality bound, as shown in Sec. 4.1.

We will see that LDD searches over a sequence of low-dimensional submanifolds of the configuration space; it is similar in this respect to *variational dynamic programming* (VDP (Barraquand and Ferbach 1994)). VDP, however, generates these submanifolds randomly, whereas LDD generates these using a learned basis, which leads to significant practical and theoretical benefits. VDP is also not guaranteed to produce monotonic improvement in the continuum limit, in contrast to LDD, because it does not account for important metric transformations between iterations, as described in Section 2.4.

An in-depth description of our method follows, in which we will describe some of the finer points of LDD. We then present results applying LDD to two challenging, high-dimensional, optimal motion planning problems and and give some theoretical justification as to why and when it might work well.

2 Method

We consider the problem of optimal holonomic motion planning in high-dimensional spaces. Let $x(t) : I \to \mathbb{R}^N$ denote a trajectory of some entity as a differentiable function from time (spanning the unit interval I) to configuration space of said entity, which is assumed to be $\mathbb{R}^N$ for the time being. Our goal will be to find a trajectory $x^*(t)$ such that $x^*(0) = x_a$ and $x^*(1) = x_b$, for some specified $x_a, x_b \in \mathbb{R}^N$, that minimizes a certain cost functional corresponding to the intuitive notion of a *least-cost path* under a given *cost function* $C(x) : \mathbb{R}^N \to \mathbb{R}$. Given these definitions, we formally define the problem of finding an optimal trajectory $x^*(t)$ as follows.

Definition 2.1 (Holonomic optimal motion planning)**.**

$$J\{x(t)\} = \int_0^1 \|\dot{x}(t)\| C(x(t)) dt \quad (1)$$

$$x^*(t) = \arg\min_{x(t)} J\{x(t)\} \quad (2)$$

$$\textit{subject to} \quad x(0) = x_a$$
$$x(1) = x_b$$

We solve this problem via dynamic programming (DP); specifically, the Fast Marching Method (FMM (Sethian 1996)), which is very similar to Dijkstra's algorithm, but approximately solves (2.1) on a sampled lattice, rather than solving a discrete approximation to (2.1), as would be required to apply Dijkstra.

The computational complexity of DP is nearly linear in the number of points at which we sample our domain. Unfortunately, in order to obtain a good sampling of most do-

mains, we must sample a number of points that grows exponentially with the dimension of the domain, making this a computationally intractable approach for high-dimensional problems.

2.1 The virtue of low-dimensional cost structure

Our solution to this problem of intractability is ultimately based on finding and exploiting the low-dimensional structure that lies latent in many motion planning problems. The following theorem is the core result that enables us to do so.

Theorem 2.1. *Consider a holonomic motion planning problem (2.1) with a cost function $C(x)$, and suppose that there exists an $N \times d$ matrix W such that $d \leq N$ and*

$$C(x) = C(WW^T x),\ \forall x.$$

Then there exist an optimal path $x^*(t)$ of this planning problem and a function $a(t) : I \to \mathbb{R}^d$ such that

$$x^*(t) = Wa(t) + x_a + (x_b - x_a)s(t) \tag{3}$$

This is illustrated in Fig. 2. Intuitively, this result means that cost functions that have a sort of low-dimensional structure, are associated with optimal paths that are also low-dimensional, in the sense that they are contained within low-dimensional spaces. A proof is given in the appendix.

2.2 Learning low-dimensional cost structure

A variation of this result, focusing more on the concomitant symmetry of the associated value function (not discussed here), was recently stated without proof in our previous work, which describes an algorithm known as SLASHDP (Vernaza and Lee 2011). SLASHDP, given an N-dimensional planning problem with cost function C, first learns an $N \times d$ matrix W such that $C(x) \approx C(WW^T x)$. Theorem 2.1 is then applied to find an optimal path of the approximate cost function $C(W(\cdot))$, by searching over the $d+1$-dimensional set of optimal paths described by (3). More precisely, SLASHDP (up to some minor details) finds an approximate solution $x^*(t)$ defined by

$$x^*(t) = \underset{a(t),s(t)}{\arg\min}\, J\{Wa(t) + x_a + (x_b - x_a)s(t)\}. \tag{4}$$

For most intents and purposes, LDD can be considered a generalization of SLASHDP. However, we will use the same subroutine to identify the basis in which to plan, and so we describe it here. This subroutine estimates the first basis vector as

$$w_1 = \underset{\|u\|=1}{\arg\max}\, \mathbb{E}_x u^T \nabla C(x) \nabla C(x)^T u \tag{5}$$

and subsequent basis vectors are chosen to optimize the same objective, but constrained to be orthogonal to previous basis vectors. In other words, w_1 is the direction that maximizes the expected squared directional derivative of $C(\cdot)$. We can therefore consider w_1 to be a direction on which the cost function depends strongly, and we can consider subsequent w_i to be ordered by the extent to which the cost function depends on them.

Computationally, solving the above optimization problem involves estimating the expectation $\mathbb{E}_x \nabla C(x) \nabla C(x)^T$. The tall matrix W is then formed from the eigenvectors of this matrix, sorted in order of decreasing eigenvalue. Simple random sampling in a finite search volume can be employed to estimate this expectation.

2.3 Learning dimensional descent

Our generalization of SLASHDP is fairly simple. Given the learned basis W, instead of searching once over the set of paths described by (3), we find a sequence of paths $\bar{x}^k$ defined by

$$\bar{x}^{k+1}(t) = \underset{a^k(t),s(t)}{\arg\min}\, J\{W^k a^k(t) + \bar{x}^k(s(t))\}, \tag{6}$$

where each W^k is a matrix comprised of a strict subset of the columns of W. Assuming that W^k is an $N \times d$ matrix, each step involves solving a $d+1$-dimensional DP problem. The algorithm begins by setting W^0 to the first d columns of W, and by setting $\bar{x}^0(s) = x_a + (x_b - x_a)s$. Subsequent steps set W^k equal to the next d columns of W, and so on. After all the columns of W are exhausted in this way (or we reach a predefined limit), we again set W^k equal to the first d columns of W. LDD proceeds in this way until convergence or some other termination condition is met.

Choosing the W^k in this way ensures that we choose directions to optimize in order of their relative importance. This intuitively minimizes the chance that LDD will get stuck in some poor local minimum early on. Furthermore, by Theorem 2.1, it ensures that LDD will terminate with a globally optimal solution in one step provided that the conditions of that theorem are met and we choose a large enough d (rigorous proof omitted for lack of space). We note also that whether or not a global optimum is achieved, LDD has the important property of monotonic convergence towards a local optimum. This is a simple consequence of the fact that the solution in one iteration is contained within the feasible set of the next iteration.

The geometry of the LDD optimization is illustrated in Fig. 1. The set of paths considered by LDD at each iteration is the (generally non-linear) submanifold of $\mathbb{R}^N$ obtained by sweeping $\bar{x}^k$ in the directions spanned by the column space of W^k. When W^k is a single column, this submanifold is simple *ruled surface*, as illustrated.

2.4 Technical details

The rest of this section is devoted to the subtle technical issue of solving the LDD optimization (6) via DP. A naive attempt to solve (6) might involve sampling a^k and s on a finite lattice, evaluating the cost function at these points as $C(a^k, s) = C(W^k a^k + \bar{x}^k(s))$, and applying the Euclidean FMM to the resulting problem. However, this will not yield the correct result, due to the aforementioned non-Euclidean geometry of the problem (Fig. 1). If we wish to optimize paths under the original objective, which specifies the Euclidean metric in $\mathbb{R}^N$, we must therefore take into account how this metric transforms under a change to the coordinates (a^k, s) of the submanifold that is the set of feasible paths.

To derive the correct procedure, we must therefore substitute the expression for the path in manifold coordinates

$x(t) = W^k a^k(t) + \bar{x}^k(s(t))$ into the Euclidean metric to derive its expression in terms of manifold coordinates. Doing so yields

$$\begin{aligned} \dot{x}^T \dot{x} &= \dot{a}^k(t)^T \dot{a}^k(t) + \dot{a}^k(t)^T (W^k)^T \frac{d}{dt}[\bar{x}^k(s(t))] \\ &\quad + \frac{d}{dt}[\bar{x}^k(s(t))]^T \frac{d}{dt}[\bar{x}^k(s(t))]. \end{aligned} \tag{7}$$

To render optimization with respect to this metric a problem amenable to optimization by standard methods, which generally assume the Euclidean metric, we note that we can make some helpful simplifying assumptions. First, note that we can equivalently parameterize the manifold by replacing $\bar{x}^k$ with $\bar{x}^k - W^k(W^k)^T\bar{x}^k$, yielding

$$x(t) = W^k a^k(t) + (I - W^k(W^k)^T)\bar{x}^k(s(t)). \tag{8}$$

(Note that this choice of manifold coordinates is also illustrated in Fig. 1.) Substitution of this expression into the metric causes the cross-term to cancel, resulting in the following diagonal metric, after simplification. Let $P = I - W^k(W^k)^T$. Then

$$\dot{x}^T \dot{x} = \dot{a}^k(t)^T \dot{a}^k(t) + \dot{s}^2 \frac{d\bar{x}^k}{ds}^T P^T P \frac{d\bar{x}^k}{ds}. \tag{9}$$

At this point, we wish to make the coefficient of $\dot{s}^2$ equal to one. Note that we can achieve this simply by assuming that $\bar{x}^k(s)$ has the arc-length parameterization after projection onto the subspace orthogonal to W^k. We must therefore take care to make this assumption in all expressions involving $\bar{x}_i$—specifically when applying (8) and when calculating the cost function in terms of manifold coordinates, a^k and s. Doing so ensures that we can safely apply an efficient Euclidean FMM solver to optimize the correct objective.

3 Experiments

We applied LDD to two high-dimensional motion planning problems—first, the problem of planning for a deformable robot; and second, the problem of planning for a planar arm in the presence of obstacles. For all of the experiments described below, we set $d = 1$.

3.1 Deformable robot planning

For this experiment, we simulated a robot that can translate freely in the plane as well as deform its shape. The deformation is controlled by a set of 16 parameters corresponding to Fourier series coefficients of the radius of the boundary of the robot, as a function of angle. To make the problem more challenging, we then applied a random rotation to these coordinates, randomly mixing together all the degrees of freedom, and applied LDD to the problem in these new coordinates.

Results are shown in Fig. 3. We compared dimensional descent in the original, randomized coordinates (denoted DD), to LDD in the learned coordinates; that is, we applied LDD to the problem in random coordinates, which then transformed the problem into learned coordinates. Dimensional descent in the random coordinates produced a very poor solution in which the robot did not correctly deform or translate to fit the maze, whereas LDD correctly translated the robot through the maze, while deforming the shape of the robot to fit the corridors of the maze.

We also compared the convergence properties of LDD versus DD in the random coordinates, shown as insets in Fig. 3. DD in random coordinates yielded a significant improvement in cost in the first iteration, but no noticeable improvement in subsequent iterations, indicating that DD quickly fell into a poor local minimum. LDD also made the bulk of its progress in its first iteration, but its solution after the first iteration was several orders of magnitude less costly than that of DD. After the first iteration, it continued to make steady improvement, especially up until the 10th iteration. After this iteration, LDD seemed to slowly approach a locally optimal solution.

3.2 Arm planning

We additionally applied LDD to the problem of planning a collision-free trajectory for an 11-DOF planar arm with no self-collisions. Our intent was to thoroughly study the ability of LDD to find high-quality solutions in very cluttered environments, especially in relation to other commonly used approaches.

Our main experiment is depicted in Fig. 4a, which also shows the LDD solution superimposed. We fixed the initial and final arm configurations and found collision-free paths between them, varying the number of obstacles in the environment. The label on each obstacle indicates the trial in which that obstacle was first introduced; each obstacle was then kept in the environment for subsequent trials, making later trials more challenging than earlier ones.

Quantitative results are given in Fig. 4b. We compared LDD to several other methods, including A* with different heuristics and RRT-based planners. Fig. 4b shows the cost of the solution obtained with each method for each trial. RRT is a standard bidirectional RRT, provided as a baseline for comparison, though it does not perform any optimization. All other methods attempt some optimization. S-RRT is a bidirectional RRT that has been post-processed with an elastic band planner to improve the quality of its solution, as is common practice. S-TSRRT is a bidirectional variant of Task-Space RRT (Shkolnik and Tedrake 2009), also post-processed with an elastic band. DD refers to dimensional descent in the original coordinates. A*Proj refers to A* using the heuristic of the distance of the end-effector to the goal, while A*Full is A* using the Euclidean distance heuristic in the configuration space.

Both A* variants and TS-RRT were only able to find solutions for the trials with less than seven obstacles in a reasonable amount of time—these methods ran for more than eight hours on our 3 GHz Intel Xeon test machine without finding a solution. By its nature, DD always found some solution, but it was not always feasible, as evidenced by the fact that the cost of its solutions far exceeded that of the baseline RRT in many trials. The standard bidirectional RRT (pre- and post-smoothing) and LDD were therefore the only methods consistently able to find solutions for the most difficult problems.

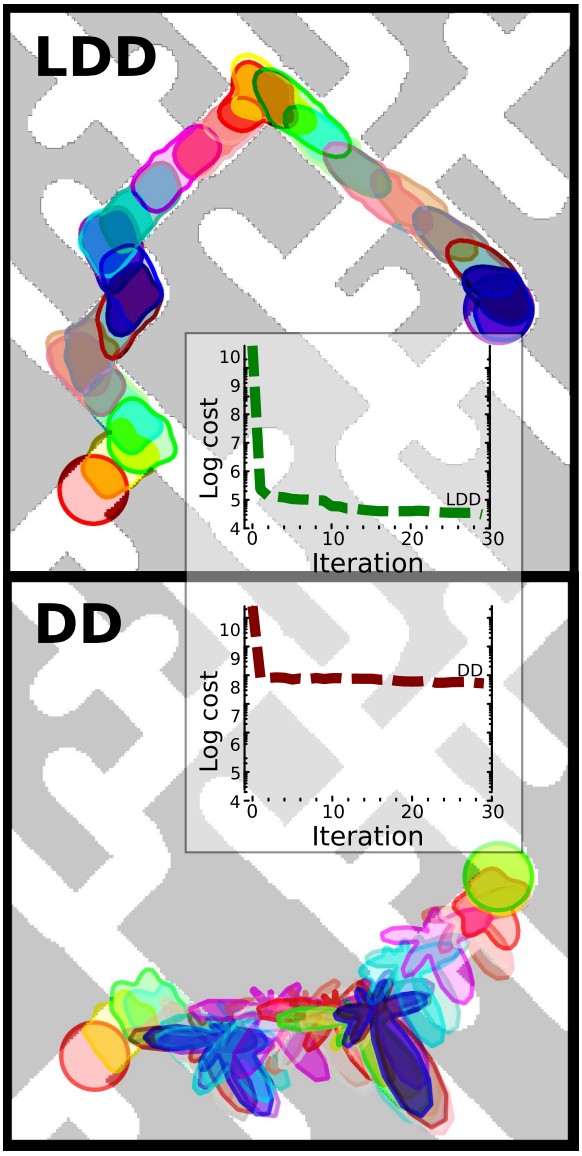


Figure 3: Results of experiment in 18-dimensional shape planning (see text for details). Consecutive shapes shown as overlapping, translucent objects. Insets show log cost as a function of iteration number.

In terms of solution quality, LDD consistently outperformed every other method tested. The performance gap between LDD and S-RRT was as great as a factor of five. For the most difficult trials, LDD still managed to find a solution of roughly half the cost of S-RRT. In order to find any solutions with A* in a reasonable amount of time, we had to apply a high heuristic weighting factor, which generally caused the solutions to be very suboptimal.

4 Analysis

In this section, we briefly examine two issues relating to the performance of LDD. First, we give a simple result illustrating how concrete approximation guarantees might be obtained for LDD. We then discuss the issue of local minima, demonstrating a simple case where a local minimum might lead to a poor solution.

4.1 Approximation guarantee

As previously mentioned, LDD is guaranteed to find the globally optimal path in one step provided that $C(x) = C(W^1 W^{1T} x),\ \forall x$. In practice, this condition will rarely be met with a basis W^1 containing only a few columns, which is the only case in which we can practically apply DP to find a solution. However, we can easily show that this result is robust in the sense that if the condition still holds approximately, we obtain a solution of bounded suboptimality.

To do so, we first consider a variant of LDD—denoted LDD'—that approximates the cost function C by a *compressed* cost function C' such that $C(x) \approx C'(x) = C(W^1 W^{1T})$ and $C'(x) \geq C(x)$, $\forall x$ (this variant is essentially SLASHDP (Vernaza and Lee 2011)). The following result then applies.

Theorem 4.1. *Suppose LDD' is applied to a problem with cost function C, and a compressed cost C' is obtained such that $C'(x) \geq C(x)$ and $C'(x) - C(x) \leq \delta$, $\forall x$. Let $J\{x;C\} = \int \|\dot{x}\| C(x) dt$ denote the cost functional evaluated on path x using cost function C, let $x^* = \arg\min_x J\{x;C\}$, and let $x'^* = \arg\min_x J\{x;C'\}$. Then, assuming $C(x) \geq 1,\ \forall x$,*

$$\frac{J\{x'^*;C\} - J\{x^*;C\}}{J\{x^*;C\}} \leq \delta. \tag{10}$$

Proof. We first have that for any path $x(t)$,

$$\begin{aligned} J\{x;C'\} - J\{x;C\} &= \int (C'(x) - C(x)) \|\dot{x}\| dt \\ &\leq \delta \int \|\dot{x}\| dt. \end{aligned} \tag{11}$$

We also observe that

$$J\{x^*;C\} \leq J\{x'^*;C\} \leq J\{x'^*;C'\} \leq J\{x^*;C'\}. \tag{12}$$

Therefore,

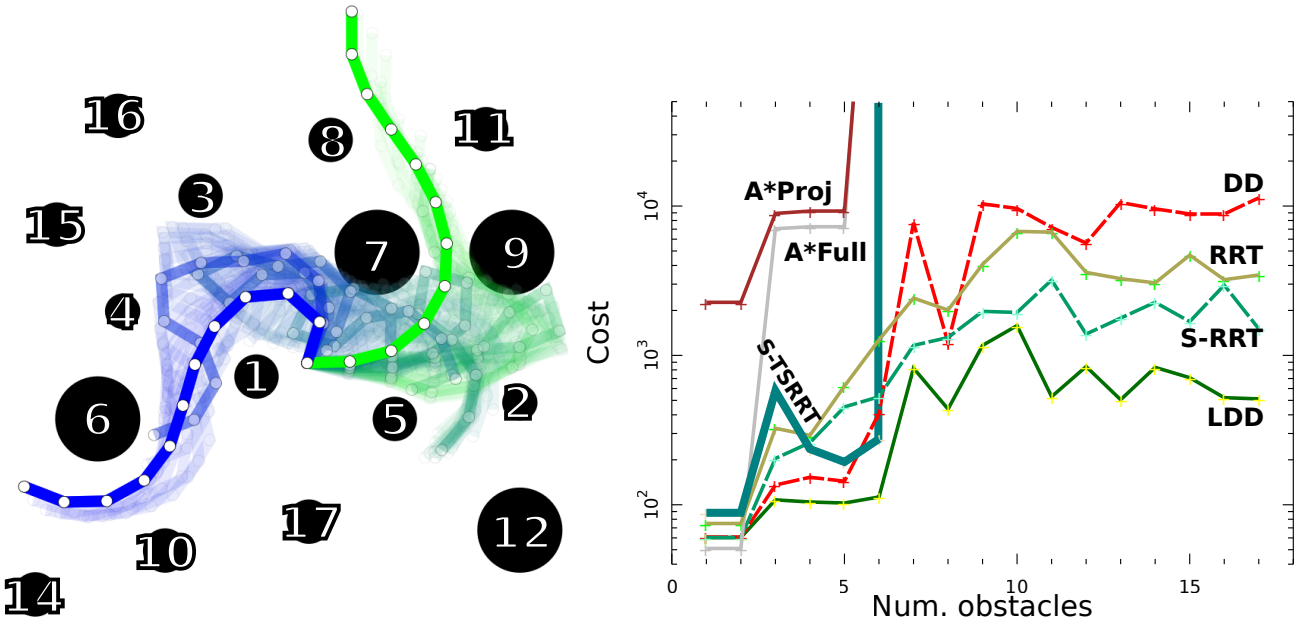
$$\begin{aligned} J\{x'^*;C\} - J\{x^*;C\} &\leq J\{x^*;C'\} - J\{x^*;C\} \\ &\leq \delta J\{x^*;C\}, \end{aligned} \tag{13}$$

which is equivalent to the desired result. □

The δ in the above theorem can be thought of a surrogate for how well the low-dimensional approximation holds. The result implies that the solution returned by LDD' is boundedly suboptimal in terms of δ, as expected. It is simple to show that this result also holds for LDD, since the LDD path is guaranteed to be no more costly than the LDD' path.

4.2 Local minima

LDD therefore begins with an approximately optimal solution and, as previously mentioned, improves upon it monotonically (neglecting discretization effects). However, LDD



(a) (b)

Figure 4: Fig. 4a: visualization of results applying LDD to an arm planning problem. End configurations represented as fully opaque lines, intermediate configurations by less opaque lines. Fig. 4b: comparison of several methods applied to the problem in 4a. Abscissa shows number of obstacles, and ordinate shows cost of found solutions (note log scale). Obstacles were added in order shown by 4a (obstacle 13 outside area shown). See Sec. 3.2 for details.

is theoretically vulnerable to local minima, a simple example of which is given here.

Figure 5a illustrates a local minimum that occurs in a three-dimensional problem if two-dimensional search submanifolds are employed. The figure illustrates a cage-like obstacle created by carving cubes out of a large hollow cube at the corners. The endpoints are illustrated as spheres, and the initial solution is illustrated as a tube drawn between them. In the first iteration, the plane illustrated by the grid is searched for an improved solution; however, no progress is possible, as no collision-free path exists between the endpoints lying in the plane. The same is true for all subsequent iterations, by symmetry. A local minimum is therefore immediately encountered. Figure 5b shows how this local minimum is fairly unstable with respect to changes in the cost function. In this case, a hole in the cage at the level of any search plane ensures that LDD is able to find a better, collision-free local minimum.

In practice, minima of the sort encountered in Fig. 5a could be escaped by introducing a small amount of random noise either into the path or the basis selection, which would likely result in a submanifold containing collision-free paths. Alternatively, raising the dimension of the searched submanifolds by one would cure the problem.

5 Conclusions

Motion planning in high-dimensions is a problem that is obviously computationally intractable in general. We have described LDD, an algorithm that automatically discovers simple structure in a motion planning problem in the hopes of bypassing this general complexity problem. Simple sufficient conditions were given for the global optimality of LDD based on the structure of the planning cost function. In real-world problems, LDD becomes an approximation algorithm that reliably surpasses the abilities of other commonly used methods to obtain high-quality, low-cost solutions.

We hope that this work will help stimulate discussion as to how low-dimensional structure might be automatically found and exploited in other ways, and in other variations of the motion planning problem.

A Proof of Theorem 2.1

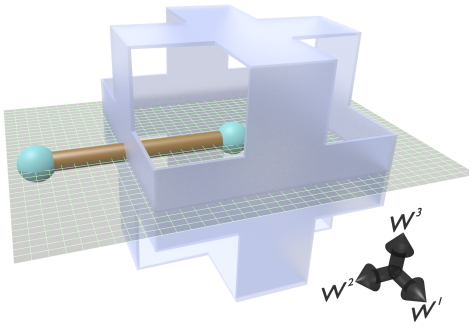
The constructive proof of Theorem 2.1 given here is a straightforward application of the calculus of variations (Arnold 1989). First, the assumption of the theorem implies that in some coordinates, the cost function C only depends on the first d coordinates. We therefore assume w.l.o.g. that the coordinates x_i are such that C depends only on the first d of these coordinates; i.e., $\partial C/\partial x_i = 0 \; \forall i > d, \forall x$. We will refer to coordinates on which the cost function does not depend as *cyclic* coordinates, to borrow a physics term.

We then define the Lagrangian

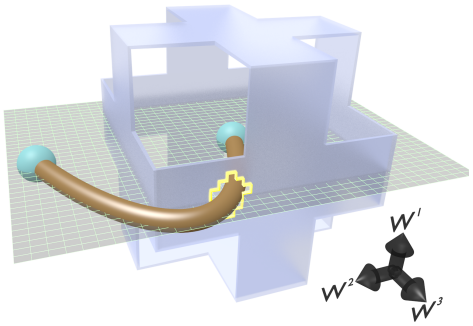
$$L(x, \dot{x}) = \|\dot{x}\| C(x)$$

and apply the Euler-Lagrange equations, yielding

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{x}_i} - \frac{\partial L}{\partial x_i} = \frac{d}{dt}\left[\frac{\dot{x}_i}{\|\dot{x}\|} C(x)\right] - \|\dot{x}\|\frac{\partial C}{\partial x_i} = 0.$$



(a) Poor local minimum



(b) Better local minimum

Figure 5: Illustration of LDD local minima with respect to two-dimensional search submanifolds. Fig. 5a shows a poor local minimum for a certain problem, while Fig. 5b shows a much better local minimum obtained in a slightly modified problem (note hole in wall). Dark arrows show learned basis.

For any cyclic coordinate, we can substitute $\partial C/\partial x_i = 0$, implying that $\forall t$, and for some yet-unknown constants k_i,

$$\frac{\dot{x}_i}{\|\dot{x}\|} C(x) = k_i. \tag{14}$$

We note without detailed proof that we can assume that $\|\dot{x}\| = \alpha$, where α is some irrelevant constant, since (1) can be shown to be invariant with respect to time-reparameterizations of $x(t)$ (i.e., moving faster or slower along some part of the path never yields any change in cost). (14) then yields $N-d$ independent, separable ODEs for each cyclic coordinate. Integration of these produces

$$\begin{aligned} x_i(t) &= \alpha k_i \int_0^t \frac{1}{C(x(t))} dt \\ &= \alpha k_i F(t), \end{aligned} \tag{15}$$

defining $F(t)$ in the last step, and where we have assumed w.l.o.g. that $x(0) = 0$.

We then write the path as a linear combination of the standard basis vectors, e_i:

$$x(t) = \sum_i x_i(t) e_i$$

and substitute (15) into this equation, which produces

$$x(t) = \sum_{i=1}^{d} x_i(t) e_i + \sum_{i=d+1}^{N} \alpha k_i F(t) e_i.$$

We now solve for the k_i in this expression to express the last basis vector in terms of the problem data. The k_i can be computed from the final conditions; i.e., $x_i(1) = \alpha k_i F(1)$. Substitution of this expression and simplification then yields

$$x(t) = \sum_{i=1}^{d} x_i(t) e_i + \frac{F(t)}{F(1)} \sum_{i=d+1}^{N} x_i(1) e_i.$$

By expressing the basis e_i in terms of the basis w_i, we can obtain the desired expression, with $(x_1(t), \ldots, x_d(t))$ becoming $a(t)$, and $F(t)/F(1)$ becoming $s(t)$.

References

Arnold, V. I. 1989. *Mathematical Methods of Classical Mechanics*. Springer-Verlag.

Barraquand, J., and Ferbach, P. 1994. Path planning through variational dynamic programming. In *IEEE International Conference on Robotics and Automation*, 1839–1846. IEEE.

Berenson, D.; Srinivasa, S.; Ferguson, D.; Collet, A.; and Kuffner, J. 2009. Manipulation planning with workspace goal regions. In *ICRA*, 618–624. IEEE.

Brock, O., and Kavraki, L. 2001. Decomposition-based motion planning: a framework for real-time motion planning in high-dimensional configuration spaces. In *ICRA*.

Dalibard, S., and Laumond, J. 2009. Control of probabilistic diffusion in motion planning. *Algorithmic Foundation of Robotics VIII* 467–481.

Diankov, R.; Ratliff, N.; Ferguson, D.; Srinivasa, S.; and Kuffner, J. 2008. Bispace planning: Concurrent multi-space exploration. *RSS*.

Hsu, D.; Latombe, J.-C.; and Motwani, R. 1999. Path planning in expansive configuration spaces. *International Journal Computational Geometry & Applications* 4:495–512.

Karaman, S., and Frazzoli, E. 2010. Incremental sampling-based algorithms for optimal motion planning. In *RSS*.

Kavraki, L.; Svestka, P.; Latombe, J.-C.; and Overmars, M. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*.

Kuffner, J., and LaValle, S. 2000. RRT-connect: An efficient approach to single-query path planning. In *ICRA*.

Likhachev, M., and Stentz, A. 2008. R* search. In *AAAI*.

Likhachev, M.; Gordon, G.; and Thrun, S. 2004. ARA*: Anytime A* with provable bounds on sub-optimality. In *NIPS*.

Sánchez, G., and Latombe, J.-C. 2001. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *Proceedings International Symposium on Robotics Research*.

Sethian, J. A. 1996. A fast marching level set method for monotonically advancing fronts. *PNAS* 93(4):1591–1595.

Shkolnik, A., and Tedrake, R. 2009. Path planning in 1000+ dimensions using a task-space Voronoi bias. In *ICRA*.

Sucan, I. A., and Kavraki, L. E. 2008. Kinodynamic motion planning by interior-exterior cell exploration. In *WAFR*.

Vernaza, P., and Lee, D. D. 2011. Efficient dynamic programming for high-dimensional, optimal motion planning by spectral learning of approximate value function symmetries. In *ICRA*.