Section Patterns: Efficiently Solving Narrow Passage Problems in Multilevel Motion Planning

Andreas Orthey and Marc Toussaint

***Abstract*—Sampling-based planning methods often become inefficient due to narrow passages. Narrow passages induce a higher runtime, because the chance to sample them becomes vanishingly small. In recent work, we showed that narrow passages can be approached by relaxing the problem using admissible lower dimensional projections of the state space. Those relaxations often increase the volume of narrow passages under projection. Solving the relaxed problem is often efficient and produces an admissible heuristic we can exploit. However, given a base path, i.e., a solution to a relaxed problem, there are currently no tailored methods to efficiently exploit the base path. To efficiently exploit the base path and thereby its admissible heuristic, we develop section patterns, which are solution strategies to efficiently exploit base paths in particular around narrow passages. To coordinate section patterns, we develop the pattern dance algorithm, which efficiently coordinates section patterns to reactively traverse narrow passages. We combine the pattern dance algorithm with previously developed multilevel planning algorithms and benchmark them on challenging planning problems like the Bugtrap, the double L-shape, an egress problem, and on four pregrasp scenarios for a 37 degrees-of-freedom shadow hand mounted on a KUKA LWR robot. Our results confirm that section patterns are useful to efficiently solve high-dimensional narrow passage motion planning problems.**

***Index Terms*—Intelligent robots, motion planning.**

I. Introduction

Sampling-based motion planning algorithms are a successful paradigm to automate robotic tasks [53]. However, sampling-based algorithms do not perform well when the state space of the robot contains narrow passages [39], [60], [81], [95], which are low-measure regions which have to be traversed to reach a goal. Narrow passages are often occurring in tasks which are particularly important in robotic applications, like grasping, peg-in-hole, egress/ingress, or long-horizon planning problems [25], [36].

In previous work, we and other research teams have shown that we can often efficiently solve high-dimensional planning problems by using admissible lower dimensional projections of the state space, a topic we refer to as multilevel motion planning [5], [23], [69], [78], [101]. When using a multilevel motion planning framework, we can often use solutions to simplified planning stages as admissible heuristics for the original problem [1], [72]. To efficiently exploit those admissible heuristics, we can use biased sampling methods [69], [77], which we can combine with classical planning algorithms like the rapidly exploring random tree algorithm [65], the probabilistic roadmap planner [67], its optimal star versions [69], or the fast marching trees planner [77]. However, while showing promising runtimes, those algorithms are prone to get trapped when run on problems involving narrow passages.

In this work, we address narrow passages in multilevel motion planning problems by developing section patterns. Section patterns are methods to explicitly address problematic situations that occur when we exploit solutions to relaxed problems.

We introduce four section patterns. First, we introduce the Manhattan pattern, which we use to compute solution paths which actuate the minimal amount of joints to reach a goal region, which is advantageous for high-dimensional systems [14], [69]. Second, we introduce the Wriggle pattern, which we use to make small random walk steps to traverse a narrow passage. Third, we introduce the Tunnel pattern, which we use to steer around small infeasible regions. Fourth, we introduce the Triple step pattern, which we use to backtrack in case the algorithm gets stuck. In Fig. 1, the Triple step pattern is showcased for a 37-degrees-of-freedom (DoF) robotic hand. We execute the pattern when a collision occurs. We first backstep, then sidestep, and finally we make a forward step to reach a goal position. The details of this and of the other pattern will be detailed later in this article.

To coordinate the execution of the four section patterns, we develop a novel algorithm we call *pattern dance*. The pattern dance algorithm applies the section patterns sequentially by trying first a pattern which is easy to compute (Manhattan pattern) and reverting to the more complex pattern like Wriggle or Tunnel only if needed. If all those patterns fail, we revert to the Triple step pattern, which is the most computationally demanding pattern. We embed this pattern dance algorithm into four multilevel planners [69]—namely, the quotient space RRT (QRRT) [65], the quotient space roadmap planner (QMP) [67], and its optimal versions QRRT* and QMP* [69].

Our contributions are as follows.

Manuscript received October 27, 2020; revised February 9, 2021; accepted March 29, 2021. This work was supported in part by MPI-IS for the Max-Planck Fellowship. This paper was recommended for publication by Associate Editor S. Huang and Editor W. Burgard upon evaluation of the reviewers' comments. *(Corresponding author: Andreas Orthey.)*

Andreas Orthey is with the Physical Reasoning and Manipulation Lab, Max Planck Institute for Intelligent Systems, 70569 Stuttgart, Germany (e-mail: aorthey@is.mpg.de).

Marc Toussaint is with the Physical Reasoning and Manipulation Lab, Max Planck Institute for Intelligent Systems, 70569 Stuttgart, Germany and also with the Intelligent Systems Lab, Technical University of Berlin, 10587 Berlin, Germany (e-mail: toussaint@tu-berlin.de).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TRO.2021.3070975.

Digital Object Identifier 10.1109/TRO.2021.3070975

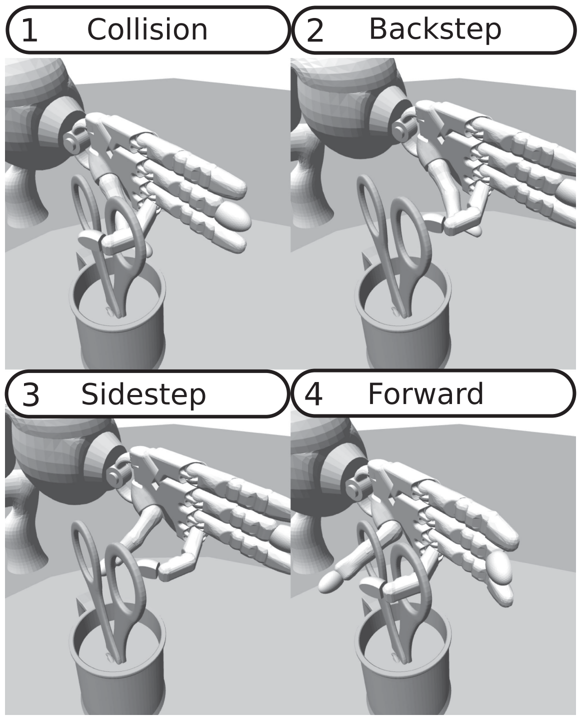

Fig. 1. Efficient exploitation of admissible heuristics (stemming from solution to relaxed problem) using the triple step pattern. The triple step pattern is one of four section patterns we advocate to efficiently exploit admissible heuristics near narrow passages.

1) We develop section patterns to efficiently exploit base space paths (solutions to relaxed problems).
2) To coordinate sections patterns, we develop the pattern dance algorithm.
3) We combine the pattern dance algorithm with four multilevel planners (QRRT, QRRT*, QMP, QMP*) and compare against 36 planners from the open motion planning library (OMPL) and a previous sidestepping algorithm [69] on 7 challenging scenarios.

II. Related Work

Let us review the literature by focusing on two topics. First, we focus on generating admissible heuristics [21] for motion planning problems involving continuous domains [53]. We discuss sources of admissible heuristics like constraint relaxations, lazy search, informed trees, and past experience. Second, given an admissible heuristic, we review methods to efficiently exploit the heuristic either using path section approaches, local minima avoidance or narrow passage handling.

A. Generating Admissible Heuristics

Motion planning [53] is a well-studied topic which has been successfully applied to a wide range of problem domains [64]. One of the most promising paradigms to solve motion planning problems are (asymptotically optimal) sampling-based planners [6], [26], [46], [84], [85]. However, these planners might become inefficient in state spaces which are too high-dimensional [69], contain intricate constraints [43], or narrow passages [55]. We can, however, often solve such problems efficiently, if we use admissible heuristics [1].

We believe there are three large sources of admissible heuristics. First, we can compute admissible heuristics as solutions to relaxed problems [72]. Early instances of this idea to motion planning can be found in the constraint relaxation frameworks by Bayazit *et al.* [5], Ferbach and Barraquand [23], Sekhavat *et al.* [88]. Newer instances of this idea are putting the focus on different aspects like the specific type of projection [31], [93], or the type of lower dimensional space [9], [67]. We refer to all those frameworks under the collective term multilevel motion planning [69]. We can apply multilevel frameworks both to holonomic [77], [78] and nonholonomic planning problems [69], [101]. To create multilevel abstraction, we can often remove links from a robot [5], [108], shrink links [3], [81], or approximate a robot by simpler geometries, either exact [32], [67], or approximate [11], [79], [96]. While most methods use prespecified levels of abstraction, we can also use workspace information to compute abstractions on the fly [58], [105], adaptively switch between abstractions [92], or learn useful abstractions for specific instances [9]. Our approach is similar, in that we also use a multilevel motion planning framework [69]. However, our work is complementary, in that we focus specifically on computing path sections in the presence of narrow passages in the state space.

A second source of admissible heuristics are lazy search [8], [34] and informed sets [27], [45]. Instead of using relaxations, we can compute lazy paths (paths not checked for collisions), either forward from the start [37] or backward from the goal [91], to create an efficient heuristic which we can exploit using dedicated algorithms [28]. Once a solution exists, we can also exploit informed sets, sets which exclude all states with provable higher cost-to-go [27], [28]. Those methods are particularly important, since edge evaluations is one of the bottlenecks in motion planning [48]. It makes, therefore, sense to develop heuristics which evaluate edges as late as possible [38], [62].

Third, inspired by pattern database approaches in discrete search [15], [22], [40], we can also construct admissible heuristics by using past experience. We can achieve this by either precomputing motion primitives, like steering functions or controllers like linear quadratic regulators [82], [83]. Or, we can store previous solution paths directly and use them as heuristics in new environments [18], [76]. Our work is complementary, in that we assume a given heuristic and we focus on exploiting this heuristic as efficiently as possible.

B. Exploiting Admissible Heuristic

Given an admissible heuristic, we can optimally exploit it by discretizing the state space [24] and by using the A* algorithm [1], [35], [72]. However, discretizing the state space usually does not scale well to higher dimensional state spaces [12], [29], [73], and performance would be sensitive to the resolution used [20]. To avoid discretization, we found three categories of work which use continuous methods to exploit admissible heuristics.

First, we can use biased sampling methods. A straightforward way would be to represent the heuristic value of a state by the radius of a hypersphere around the state [56]. We could then exploit this hypersphere using dynamic domain sampling [104]. Using such a scheme, we would expand states with higher heuristic values more often. Depending on the exact type of heuristic function used, we would obtain sampling distributions which would increase the probability to sample states which are near to restricted workspace geometries [98], [103], to state space obstacles [2], or to narrow passage [39]. Those sampling distributions could also be learned over time to improve sampling [42], [59]. Our approach is similar as we also use sampling-based methods. We differ, however, as we concentrate on designing efficient patterns complementary to biased sampling methods.

Given a solution to a relaxed problem, we can often use this solution as a guide path heuristic [96], [108] to quickly find a solution in the original state space. Using the parlance of fiber bundles, we call this the find section problem [69] . This problem requires a relaxed solution (a base path), which we can find by computing workspace regions [74], by using workspace graphs [17], [97], or by using a simpler robot geometry [96]. In more complex environments, it is often advantageous to use multiple base paths [17], [102], which decompose the original problem into smaller subproblems [7], [66], [75]. To exploit a base path, we can often use restriction sampling [67], [71] , which is highly efficient in high-dimensional state spaces, where uniform sampling would most likely fail to find solutions in a reasonable time [32]. Apart from biasing sampling, we can also explicitly search over the set of states which project onto the base path [108], which we call the path restriction. To find paths over path restrictions, we previously developed a sidestepping approach [69], where we propagate states along the path restriction and execute sidesteps when collision occur. However, as we show in Section V, sidesteps are often not beneficial for narrow passages. While we also search over path restrictions, we differ by developing dedicated patterns to more efficiently traverse narrow passages.

Path section approaches and other heuristic search methods often fail because they reach local minima. We define a local minimum as a region in state space where the heuristic is not or only weakly correlated with the true cost-to-go [100]. To address local minima, we can choose one of two approaches. First, we could preemptively avoid local minima. If the environment is static, we can learn minima regions and use this information to update the heuristic function [100]. Second, we could try to escape local minima. There exist several methods to escape local minima like deflating the heuristic value of states close to obstacles [19] or increasing the search resolution to prevent evaluation of closeby states [20]. A related idea is to utilize Tabu search [30] to prevent sampling in previously visited regions.

It is important to make the distinction between local minima which trap the planner and regions which might look like local minima but which a planner can actually traverse. We call such regions narrow passages [86]. To verify the existence of narrow passages in low-dimensional state spaces, we can use exact infeasibility proofs [4], [87], for example using geometrical shapes like alpha complices [63], or cell decomposition methods [107]. Because many state spaces have a local product structure, we can often use configuration space slices [57], [89] to efficiently test for infeasibility [99]. If the problem is feasible, we could then use the geometrical shapes to enumerate narrow passages [61]. To exploit narrow passages, we could bias sampling to the most constricted areas [95], [103]. We differ to those approaches by not explicitly modeling narrow passages or local minima, but we instead, develop reactive measures to escape minima and to traverse narrow passages. We thereby avoid spending time on irrelevant narrow passages.

III. Background

Let us describe the necessary background to follow the exposition of our algorithm in Sections IV and V. We start by explaining multilevel motion planning, i.e., planning with sequences of relaxed subproblems. While several formulations exist, we believe the framework of fiber bundles [69] to be a good way to concisely model multilevel abstractions and describe our algorithms. We then describe the concepts of lift, path restriction, and path section which are particularly important. Finally, we describe the notion of admissible heuristics, which is one of the fundamental concepts to exploit solutions to relaxed problems [72].

A. Optimal Motion Planning

Let X be the state space of the robot. To each state space, we associate a constraint function $\phi : X \to \{0, 1\}$ which evaluates to 0 if a state is constraint-free and to 1 otherwise. We use the constraint function to define the free state space $X_{\text{free}} = \{x \in X \mid \phi(x) = 0\}$. Together with an initial configuration $x_I \in X_{\text{free}}$ and a goal configuration $x_G \in X_{\text{free}}$, we define an optimal motion planning problem [6], [46], [84], as the tuple $(X_{\text{free}}, x_I, x_G, c)$, whereby our task is to develop an algorithm which computes a path from x_I to x_G while staying in X_{free} and minimizing the cost functional c. In this work, we use a minimal-length cost functional, but other costs are also possible like minimal energy or maximum clearance.

B. Multilevel Motion Planning

Since high-dimensional motion planning problems are often too computationally expensive to solve, we use a sequence of relaxed problems which we refer to as multilevel abstractions [69]. Given a state space X, let us denote a multilevel abstraction as the tuple $\{X_1, \ldots, X_K\}$ with $X_K = X$. To each state space X_k, we associate a constraint function ϕ_k and a projection π_k from X_k to X_{k-1}. We say that the projection π_k is admissible (w.r.t. the constraint functions), if $\phi_{k-1}(\pi_k(x)) \leq \phi_k(x)$ for any x in X_k. With admissibility, we basically guarantee that solutions are preserved under projections [65]. If we would allow inadmissible projections, we would potentially sacrifice solutions and thereby sacrifice (probabilistic) completeness.

C. Fiber Bundle Formulation

When working with multilevel abstraction, we quickly stumble upon situations where we lack the appropriate vocabulary

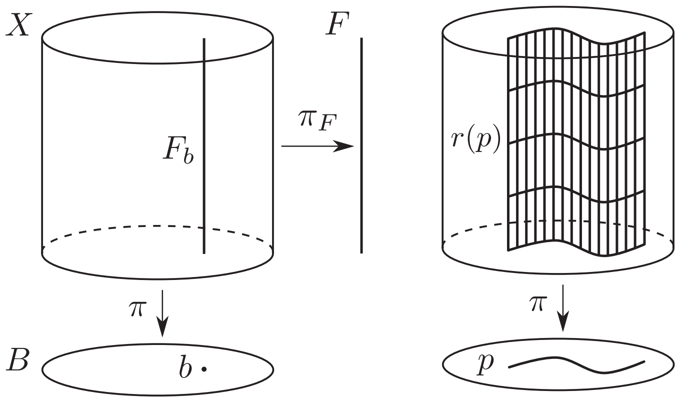

Fig. 2. Left: Fiber bundle $\mathbb{R}^3 \to \mathbb{R}^2$ with base space B, total space X, fiber space F, mappings π, π_F and fiber F_b over base element b. Right: Path restriction $r(p)$ over base path p. Adapted from [69].

to describe solution strategies. As a remedy, we describe multilevel abstractions using the framework of fiber bundles [41], [54], [90]. A fiber bundle is a tuple $(X_k, X_{k-1}, F_k, \pi_k, \pi_{F_k})$ consisting of a total space X_k, a base space X_{k-1}, a fiber space F_k, a projection mapping π_k from total to base space, and a fiber projection mapping π_{F_k} from total to fiber space. We assume the projection mapping π_k to be admissible. With a fiber bundle, we model product spaces which locally decompose as $X_k = X_{k-1} \times F_k$. The total space X_k is a union of fiber spaces which are parameterized by the base space X_{k-1}. If the level k is unimportant for the task as hand, we often refer to a fiber bundle as the tuple (X, B, F, π, π_F) with X being the total, B the base, F the fiber space, and π, π_F the base and fiber projection, respectively. We visualize a prototypical fiber bundle in Fig. 2 (left). For more details and motivation, we refer to our prior work [69]. For the purpose of this article, we focus on the three concepts of lift, path restriction, and path section, which we explain next.

D. Lift

Let (X, B, F, π, π_F) be a fiber bundle and let $b \in B$ be a base space element. We often like to project the element b back to the total space X. We call this operation a lift [69], [80]. We define a lift as a mapping $\text{LIFT} : B \to X$. To uniquely select an element in X, we will overload this function as a mapping $\text{LIFT} : B \times F \to X$ by providing a fiber space element f in F. If X is a product space, we define the lift as $\text{LIFT}(b, f) = (b, f)$ [69].

E. Path Restrictions

Let $p : I \to B$ with $I = [0, 1]$ be a path on the base space (a base path). Given a base path, one of the most central sets which we use in this work are path restrictions. A path restriction is the set $r(p) = \{x \in X \mid \pi(x) \in p[I]\}$, whereby $p[I] = \{p(t) : t \in I\}$ is the image of the base path in B and π is the projection from X to B. We visualize this situation in Fig. 2 (right), where we show the image of a base path on the disk-shaped base space and its associated path restriction on the total space.

F. Path Sections

Given a path restriction, we are often interested in finding paths which are lying inside the path restriction. We call them path sections [90]. A (smooth) path section w.r.t. a base path p is a continuous mapping s from base space B to total space X such that $\pi(s(u)) = u$ for any u in the image of p [54]. This means, for each base path element, we select a unique state from the path restriction in a continuous manner.

G. Admissible Heuristics

Our motivation to introduce path restrictions and path sections comes from the role they play in exploiting admissible heuristics. Given a goal state x_G, an admissible heuristic $h(x)$ for a state x in X is a lower bound on the true cost-to-go (or value) function $h^*(x)$, which we define as the cost of the optimal path from x to x_G through X_{free}. Formally, we write this condition as $h(x) \leq h^*(x)$ [1], [65], [72].

Given an admissible heuristic, we can try to reach the goal x_G by using locally optimal decisions [35]. If we are at a state x, we can make an optimal decision by doing a two-step approach. First, we compute the f-value of all its neighbors, which is the sum of its heuristic value and its cost-to-come from the start state. We then expand the state (node) with the lowest f-value, because, under the admissible heuristic, it is our best guess to efficiently reach the goal [72].

However, in a continuous domain, we cannot straightforwardly compute all neighboring states. Instead, we imagine computing a small ϵ-neighborhood around the state. To compute heuristic values, we project the complete neighborhood down onto the base space. To reach the goal, our best guess is to make a step into the direction of the current minimal-cost base path. The states which we would expand in that way are exactly the states on the path restriction. By searching a path section over this path restriction, we efficiently exploit the admissible heuristic given by the base path.

IV. Find Sections Using Pattern Dance

Our goal is to develop an algorithm which solves the find section problem, the problem of finding a path section over a given path restriction. After we state the problem, we discuss how the problem fits into the more general framework of motion planning using multilevel abstractions [69]. Finally, we discuss the pattern dance algorithm, which coordinates four section patterns to efficiently find feasible path sections.

A. Find Section Problem

Let (X, B, F, π, π_F) be a fiber bundle on X (possibly in a sequence of fiber bundles) and let $p : I \to B$ be a base path on B starting at $\pi(x_I)$ and ending at $\pi(x_G)$. Given the base path p and its path restriction $r(p) \subseteq X$, our goal is to develop an algorithm to find a feasible path section, i.e., a path lying in the intersection of the path restriction $r(p)$ and the free state space X_{free} connecting x_I to x_G. We call this problem the *find section problem*.

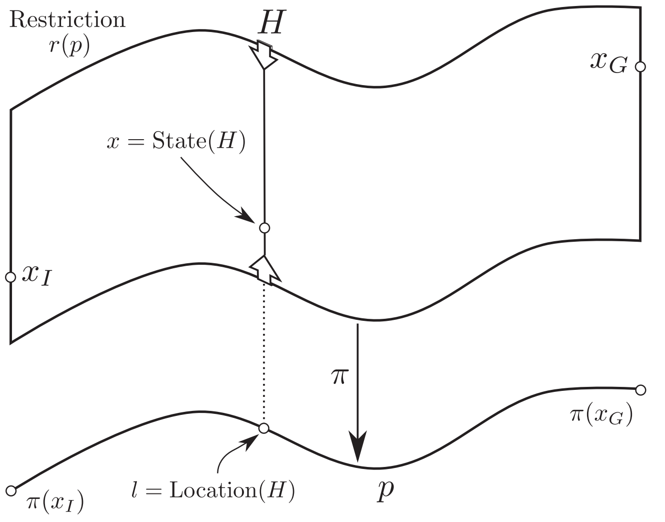

Fig. 3. Path restriction $r(p)$ on a total space X over a base path p from base space B, together with initial state x_I, goal state x_G, projection π and head pointer with head pointer H, consisting of state x and location l.

```
Algorithm 1: MultilevelPlanner($x_I, x_G, X_1, \dots, X_K$).
  Input: Initial state $x_I$, goal state $x_G$, state spaces $X_1, \dots, X_K$
  Output: Graphs $G_1, \dots, G_K$
   1: Let $\mathbf{X}$ be a priority queue()
   2: for $k = 1$ to $K$ do
   3:   $FindSection(X_k)$
   4:   $\mathbf{X}.push(X_k, \text{IMPORTANCE}(X_k))$
   5:   while $\neg ptc(X_k)$ do
   6:     $X_{\text{select}} = \mathbf{X}.pop()$
   7:     $Grow(X_{\text{select}})$
   8:     $\mathbf{X}.push(X_{\text{select}}, \text{IMPORTANCE}(X_{\text{select}}))$
   9:   end while
  10: end for
  11: return $Graphs(X_1, \dots, X_K)$
```

To illustrate the find section problem, we visualize it in Fig. 3. The figure shows a base path p on B (bottom) and its restriction $r(p)$ on X (top). Our goal is to connect x_I to x_G while staying inside $r(p)$. To efficiently solve the find section problem, we often need to track information along the path restriction. To track this information, we introduce the notion of a head pointer H as the tuple $H = (x, l, r)$ consisting of a path restriction $r(p) \subseteq X$ over a base path p in B, a current state x in $r(p)$ and a location $l \in [0, 1]$ defining the position along the base path. We think of the head pointer as a ruler which we move forward along the path restriction toward the goal state. In pseudocode, we refer to the current state as STATE(H) and its location as LOCATION(H).

B. Find Sections in Multilevel Planning

The find section problem is a subproblem of the more general multilevel motion planning problem (see Section III-B). In previous works, we proposed to solve multilevel planning

```
Algorithm 2: FindSection($X_k$).
  1: if Exists($X_{k-1}$) then
  2:   $p \leftarrow BasePath(\mathbf{G}_{k-1})$
  3:   r $\leftarrow Restriction(p)$
  4:   $H \leftarrow HeadPointer(x_I, \text{location} = 0, \text{r})$
  5:   $PatternDance(H)$
  6: end if
```

problems using a dedicated multilevel planner [69] . To clarify the role of finding sections, we describe this multilevel planner in Algorithm 1. We initialize this algorithm with an initial state x_I, a goal state x_G, and a sequence of bundle spaces $X_1, \dots, X_K$. To search for a feasible path, we first initialize a priority queue (Line 1), then we iteratively explore the bundle spaces (Line 2) by first trying to solve the find section problem (Line 3), then pushing the kth bundle space into the priority queue (Line 4). We compute the importance of a bundle space by the sampling density of its associated graph [69] as

$$\text{Importance}(X_k) = \frac{1}{|V_k|^{1/n_k} + 1} \tag{1}$$

with $|V_k|$ being the number of nodes in the graph G_k on X_k and n_k is the dimensionality of X_k. We then start a while loop a planner terminate condition (PTC) of the kth space is not fulfilled (Line 5). A PTC can be a timelimit, an iteration limit or a desired cost. We then pop the space with the lowest importance from the queue (Line 6), execute one grow iteration for the selected bundle space (Line 7), and push the space back to the queue thereby updating its importance (Line 8). The planner terminates if the PTC of all bundle spaces is false and returns the graphs of all computed levels (Line 11). From those graphs, we can then compute the (optimal) solution path using a discrete A* search [35] (if one exists). All multilevel planner share this high-level structure. Multilevel planner differ by how the Grow function is implemented.

We previously developed four multilevel planners. First, the quotient-space roadmap planner (QMP), in which we implement GROW as a probabilistic roadmap (PRM) step [47] . Second, the quotient-space rapidly exploring random tree (QRRT), in which we implement GROW as an RRT step [49] . Finally, we use the two asymptotically optimal versions QRRT* and QMP*, in which we implement a step of RRT* and PRM* [46], respectively. The algorithms also differ in how we compute the distance metric and how we implement sampling inside the grow function, as we detail in our previous publication [69].

The main contribution of our article, the pattern dance algorithm, is an efficient method to solve the find section problem. The integration into the multilevel planner is shown in the FINDSECTION method in Algorithm 2. First, we check if there exists a base space (Line 1). We then compute a base path p from the underlying graph or tree on the base space (Line 2). We then build a path restriction r from p (Line 3) and create a head on the path restriction (Line 4). We then call the pattern dance algorithm with the head as input.

Algorithm 3: PatternDance(H, depth $= 0$).

```
Parameters: Maximum branching factor $B_{\max}$, maximum
  depth factor $D_{\max}$, base space step size $\delta_{X_{k-1}}$
1:  if $ManhattanPattern(H)$ then
2:    returntrue
3:  end if
4:  if depth $\geq D_{\max}$ then
5:    returnfalse
6:  end if
7:  if WrigglePattern($H$) or TunnelPattern($H$) then
8:    returnPatternDance($H$, depth+1)
9:  end if
10: $l \leftarrow Location(H) + \delta_{X_{k-1}}$
11: $x_B \leftarrow BasePathAt(p, l)$
12: for $j \in [1, B_{\max}]$ do
13:   $x_F \leftarrow SampleFiber(x_B)$
14:   $x \leftarrow Lift(x_B, x_F)$
15:   $x_H \leftarrow State(H)$
16:   if $IsValid(x)$and$\neg CheckMotion(x_H, x)$ then
17:     if $TripleStepPattern(H, x)$ then
18:       return PatternDance($H$, depth+1)
19:     end if
20:   end if
21: end for
```

C. Pattern Dance Algorithm

We depict the pseudocode of the pattern dance algorithm in Algorithm 3. The input is a head over the path restriction and a recursion depth (initially set to zero). Inside the pattern dance algorithm, we coordinate the execution of four section patterns. The rational behind the coordination is to try less complex patterns first while we can successfully move the head forward along the path restriction. Only if no progress is made, we revert to more and more complex patterns to resolve the situation. We found this to be an efficient strategy to quickly find sections.

Those four section patterns are detailed in Section V and either move the head forward by controlling the lowest amount of joints possible (MANHATTANPATTERN), execute random walk steps with forward bias (WRIGGLEPATTERN), try to overcome small barriers using steps outside the path restriction (TUNNELPATTERN), or use a dedicated backtracking procedure (TRIPLESTEPPATTERN) to efficiently find feasible path sections.

Before going into detail, we provide a brief summary and motivation. The algorithm iterates through all four patterns, starting with the computationally most inexpensive one MANHATTANPATTERN (Line 1). If the pattern succeeds, we successfully return (Line 2). Otherwise, we check if we reached the maximum recursion depth (Line 4) and return with failure (Line 5).

If the depth is below the maximum depth, we continue by executing first the WRIGGLEPATTERN and the TUNNELPATTERN (Line 7). If one pattern successfully terminates, we recursively call the pattern dance algorithm and we increase the recursion depth (Line 8). If no pattern successfully terminates, we backtrack using the TRIPLESTEPPATTERN . To execute the triple step pattern, we first interpolate a single step forward along the base

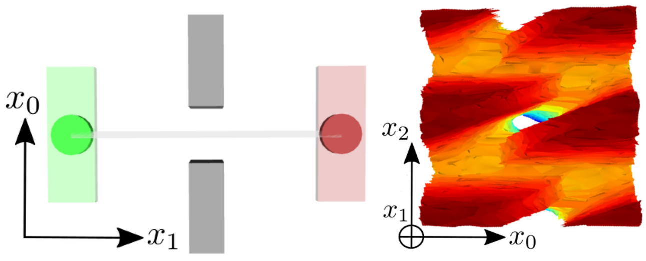

Fig. 4. Left: Rectangular rigid robot which has to traverse a narrow passage from a green start to a red goal state. Right: The geometry of its state spaces (darker colors are closer to start state).

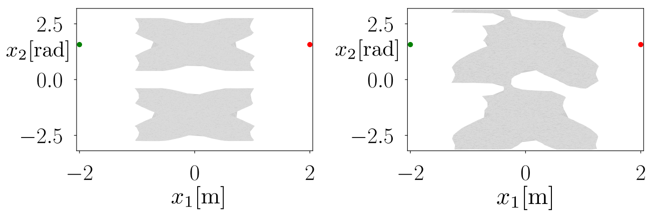

Fig. 5. Path restrictions near narrow passages.

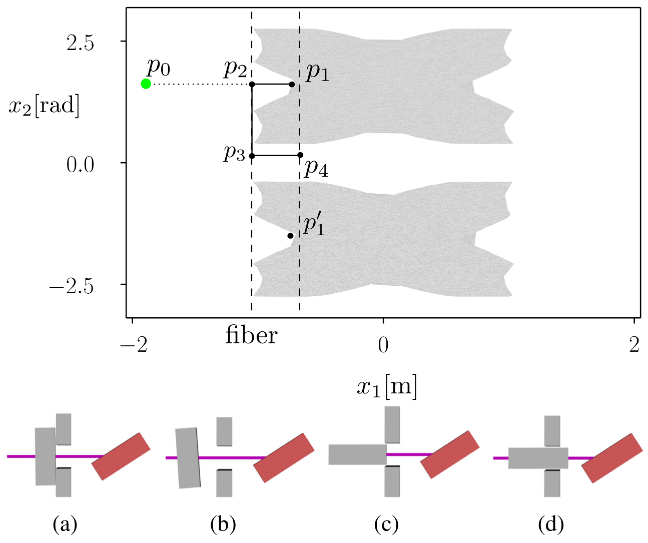

Fig. 6. Triple step pattern to traverse a narrow passage. We start at a state p_1 (a), backstep to a state p_2 (b), sidestep along the fiber to p_3 (c), and then step forward to reach a state p_4 (d). (a) At p_1 (after collision). (b) At p_2 (after backstep). (c) At p_3 (after sidestep). (d) At p_4 (after forward step).

path (Lines 10, 11). We then attempt to find a valid fiber space element for a maximum of $B_{\max}$ attempts (Line 12). This is done by first sampling a fiber state over the given base state (Line 13). We then lift the state to the path restriction (Line 14) to obtain a state x. If this state is valid and we *cannot* reach it from the head state (Line 16), we execute the triple step pattern with target x (Line 17). If we successfully executed the pattern, we call the pattern dance algorithm again recursively. Note that the small forward step of $\delta_{X_{k-1}}$ (Line 10) is an essential component of our algorithm. If we would sample directly over the head base state, we often would sample symmetrical local minima (as an example, see state p_1' in Fig. 6). We found this to be particularly

TABLE I
PARAMETERS USED IN ALGORITHM. THE VARIABLE μ_X REFERS TO THE MEASURE (VOLUME) OF THE STATE SPACE X

Parameter	Description	Values used
$D_{\max}$	Maximum depth of pattern dance	3
$B_{\max}$	Maximum branching of pattern dance	500
$S_{\max}$	Maximum sampling attempts	100
$\delta_{X_{k-1}}$	Step size on base space	$0.01\mu_{X_{k-1}}$
δ_{F_k}	Step size on fiber space	$0.01\mu_{F_k}$

important for higher dimensional state spaces, where we often encounter infinitely many symmetrical local minima (consider the set of horizontal rotations of the cylinder before entering the opening in the Bugtrap scenario in Section V).

To implement the section patterns and the pattern dance algorithm, we use the OMPL [94] . The algorithms are freely available and part of our multilevel motion planning extension of OMPL [69]. All code can be downloaded over github[1]. All parameters used in the algorithms are shown in Table I, including the values we use for the evaluations. The values for $B_{\max}, S_{\max}, D_{\max}$ are chosen as large as possible to still give good performance on our hardware.

V. SECTION PATTERNS

The pattern dance algorithm relies on four section patterns, to which we like to provide more details and motivation. Each of those section patterns is a particular approach to efficiently traverse narrow passages and escape local minima, whereby a local minimum is defined as a region where the heuristic cost is only weakly correlated with the true cost-to-go [100] . Each section pattern takes as input a head pointer and tries to move this head pointer forward along the path restriction. Please also consult Fig. 3 for visualization of the terminology used.

A. Manhattan Pattern

Our first section pattern to propagate the head pointer H is the Manhattan (MH) pattern. With the MH pattern, we interpolate a path between the head state and the goal state along the path restriction. To interpolate, we first interpolate along the base path while keeping the fiber element fixed. Once we reach the end of the base path, we interpolate along the fiber space to the goal state. This method is motivated by our desire to actuate the smallest number of joints at the same time, which is advantageous for high-dimensional systems [14].

We detail the MH pattern in Algorithm 4. We take as input a head pointer H over a path restriction r with base path p. We first project the head state onto the fiber (Lines 1 and 2) by using the fiber projection π_F. We then take the location of the head pointer along the base path (Line 3) and step along the base path in increments of $\delta_{X_{k-1}}$ (Line 5–10) and add the states to the path s (Line 4). This is done by computing the next base state (Line 6), lifting the base state into the total space (Line 7) and adding

[1]https://github.com/aorthey/MotionExplorer/ and https://github.com/aorthey/ompl/.

Algorithm 4: ManhattanPattern(H).

```
Parameters: Base space step size $\delta_{X_{k-1}}$
1:  $x_H \leftarrow State(H)$
2:  $x_F \leftarrow ProjectFiber(x_H)$        ▷ $\pi_F(x_H)$
3:  $l \leftarrow Location(H)$
4:  $s \leftarrow \emptyset$
5:  while $l < Length(p)$ do
6:      $x_B \leftarrow BasePathAt(p, l)$   ▷State $p(l)$ on base path
7:      $x \leftarrow Lift(x_B, x_F)$
8:      $s \leftarrow s \cup \{x\}$
9:      $l \leftarrow l + \delta_{X_{k-1}}$
10: end while
11: $s \leftarrow s \cup \{x_G\}$
12: $H \leftarrow CheckMotion(s)$             ▷Return Last Valid
13: return $HasReachedGoal(H)$
```

it to the path (Line 8). Once we reached the end of the base path, we add the goal state to the section (Line 11). The resulting path s is schematically shown in Fig. 3. Finally, we evaluate the path by moving along until a constraint violation occurs or we reached the goal state (Line 12). The function CHECKMOTION returns the last valid state which we use to update the head H. We then return true if the head has reached the goal and false otherwise.

B. Interlude: The Geometry Near Narrow Passages

The next three section patterns are tailor-made solutions to either traverse a narrow passage or to escape a local minimum. To motivate those patterns, we first study the geometry of state spaces near narrow passages. We use a simple toy example of a rigid rectangular body moving in the 2-D plane. The state space of this rigid body is the special euclidean group $SE(2)$, consisting of position and orientation. We assume that the body is located near to a narrow passages as shown in Fig. 4 (left). We will further assume that our task is to move the rigid body through the narrow passage, from a start state (green) to a goal state (red). We will represent a state as $(x_0, x_1, x_2) \in SE(2)$, with x_0, x_1 being vertical and horizontal displacement and x_2 the orientation. We visualize a subset of the state space in Fig. 4 (right), whereby points in collision are colored from dark red (low x_1 value, close to start) to bright blue (high x_1 value, close to goal).

To generate path restrictions, we first use a relaxation of the problem onto a circular disk as shown in Fig. 4 (Left). We model this relaxation using the fiber bundle $SE(2) \rightarrow \mathbb{R}^2$ with base space $\mathbb{R}^2$ and total space $SE(2)$ [67]. Let us assume a base path $p : I \rightarrow \mathbb{R}^2$ for the disk to be given. This path induces a 2-D path restriction in $SE(2)$, two of which we visualize in Fig. 5. The left figure shows a path restriction for a base path going straight through the passage, as shown in Fig. 4. The right figure shows a path restriction for a base path which goes slanted through the passage. Both are also slices through the state space geometry shown in Fig. 4 (right). From Fig. 5, we observe that there are at least three failure cases. Either, we reach a local minimum, we collide with constraints near a narrow passage or we get stuck in front of a small but infeasible region. For each case, we develop a dedicated section pattern to either advance or backtrack.

C. Triple Step Pattern

To escape a local minimum, we develop the triple step pattern. With the triple step pattern, we connect two states on the path restriction using a triple backtracking step.

The idea of the triple step pattern is to connect two states on (or near) the same fiber. Before explaining the pattern in detail, we first visualize the pattern in Fig. 6. You can see a rectangular rigid body in the plane, which is currently at state p_1 [see Fig. 6(a)] and which we like to move to state p_4 [see Fig. 6(d)]. To connect those states, we first move backward along the path restriction from p_1 to another state p_2 [see Fig. 6 (b)] while moving from p_4 to another state p_3 [see Fig. 6 (c)], respectively. We move backward until we can connect p_2 and p_3 by a straight line segment. In that case we execute a backstep from p_1 to p_2, a sidestep (along the fiber marked) from p_2 to p_3 and a forward step from p_3 to p_4. Note that p_4 is slightly moved forward such that we avoid situations where we backtrack to a symmetric local minimum like p_1' which would not improve our location along the path restriction.

We show the pseudocode for the triple step pattern in Algorithm 5. Our goal is to connect the head state to the given state x. We first compute a midpoint on the fiber space (Line 5) (to minimize the number of CHECKMOTION calls [62]). We then move backward along the base path while we are greater than the parameter $\delta_{X_{k-1}}$ (Lines 6 and 7). For each location, we interpolate a base state (Line 8), lift the state using the fiber midpoint (Line 9) and check if this state is valid. If it is valid, we compute intermediate states x_1 and x_2 (Lines 11 and 12) and check if the motion between them is feasible (Line 13). If that is true, we additionally check if the backward and forward steps are feasible (Lines 14, 15). If that is true, we add those edges to the graph (Lines 16–18), and update the head to our new state x (Line 19). In that case we return true (Line 20). If we fail to find such a triple step, we terminate once we reach the beginning of the base path location and return false (Line 27).

D. Wriggle Pattern

If we reach a local minimum, the triple step pattern is a way to backtrack to a narrow passage. However, we often might execute the triple step pattern prematurely, because we bumped into constraints near or in a narrow passage. To circumvent those situations, we use the wriggle pattern. With the wriggle pattern, we make coordinate random steps along the fibers of the path restriction and accept a step if it is valid, which is similar to retraction-based sampling [106]. We visualize this pattern in Fig. 7.

We show the pseudocode in Algorithm 6. We start by making one $\delta_{X_{k-1}}$ step forward from the head (Line 1). Until we have not reached the end (Line 3), we get the base state at location l (Line 4), and get the fiber element of the head state (Line 6). We

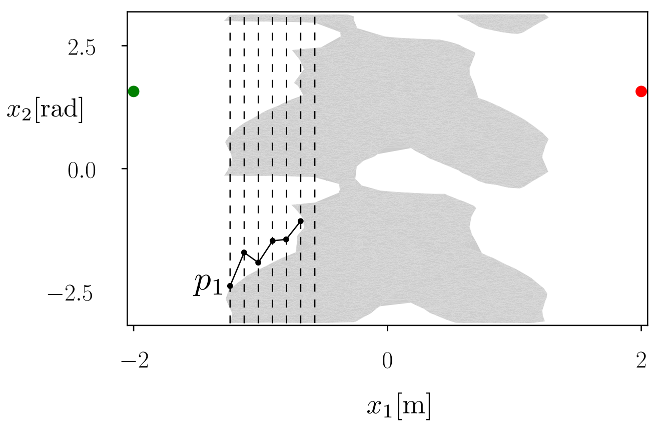

Fig. 7. Wriggle pattern to traverse a narrow passage: Given a feasible state p_1, we make coordinated random walk steps along the fibers of the path restriction. The distance between fibers is determined by the base space step size parameter $\delta_{X_{k-1}}$.

Algorithm 5: WrigglePattern(H).

```
Parameters: Base space step size $\delta_{X_{k-1}}$, fiber space step
  size $\delta_{F_k}$, maximum samples $S_{\max}$
1:  $l \leftarrow Location(H) + \delta_{X_{k-1}}$
2:  steps $\leftarrow 0$
3:  while $l < Length(p)$ do
4:    $x_B \leftarrow BasePathAt(p, l)$
5:    $x_H \leftarrow State(H)$
6:    $x_{F_H} \leftarrow ProjectFiber(x_H)$
7:    ctr $\leftarrow 0$
8:    while ctr $< S_{\max}$ do
9:      $x_F \leftarrow SampleUniformNear(x_{F_H}, \delta_{F_k})$
10:     $x \leftarrow Lift(x_B, x_F)$
11:     if $IsValid(x)$ then
12:       if $CheckMotion(x_H, x)$ then
13:         $\mathbf{G}_k \leftarrow \mathbf{G}_k \cup \{x_H, x\}$
14:         $UpdateHead(H, x)$
15:         steps $\leftarrow$ steps $+ 1$
16:         break
17:       end if
18:     end if
19:     ctr $\leftarrow$ ctr $+ 1$
20:   end while
21:   if ctr $\geq S_{\max}$ then
22:     break
23:   end if
24: end while
25: return steps $> 0$
```

then sample for $S_{\max}$ rounds (Line 8) by sampling a fiber state in the δ_{F_k} proximity of the head fiber state (Line 9). We then lift the base and fiber state (Line 10) and check if the state is valid (Line 11). If the state is valid, we check if the motion from the head to the new state is feasible (Lines 12–17). We terminate if we could not expand the state (Lines 21–23) or reach the end. We then return true if we made at least one step (Line 25).

Algorithm 6: TunnelPattern(H)

```
Parameters: Base space step size $\delta_{X_{k-1}}$, fiber space step size $\delta_{F_k}$, maximum samples $S_{\max}$
 1: $(x_{\text{End}}, l_{\text{End}}) \leftarrow TunnelEnd(H)$
 2: $x_H \leftarrow State(H)$
 3: $x_{F_H} \leftarrow ProjectFiber(x_H)$
 4: $d_{\text{best}} \leftarrow Distance(x_H, x_{\text{End}})$
 5: $l \leftarrow Location(H)$
 6: while $l \leq l_{\text{End}}$ do
 7:   if $CheckMotion(x_H, x_{\text{End}})$ then
 8:     $\mathbf{G}_k \leftarrow \mathbf{G}_k \cup \{x_H, x_{\text{End}}\}$
 9:     $UpdateHead(H, x_{\text{End}})$
10:     returntrue
11:   end if
12:   $l \leftarrow l + \delta_{X_{k-1}}$
13:   $x_B \leftarrow BasePathAt(p, l)$
14:   $\epsilon \leftarrow SmoothParameter(0, 10\delta_{X_{k-1}}, S_{\max})$
15:   ctr $\leftarrow 0$
16:   while ctr $< S_{\max}$ do
17:     $x_B \leftarrow SampleUniformNear(x_B, \epsilon(\text{ctr}))$
18:     $x_F \leftarrow SampleUniformNear(x_{F_H}, \delta_{F_k})$
19:     $x \leftarrow Lift(x_B, x_F)$
20:     if $IsValid(x)$ then
21:       $d \leftarrow Distance(x, x_{\text{End}})$
22:       if $d < d_{\text{best}}$ and $CheckMotion(x_H, x)$ then
23:         $\mathbf{G}_k \leftarrow \mathbf{G}_k \cup \{x_H, x\}$
24:         $x_H \leftarrow x$
25:         break
26:       end if
27:     end if
28:     ctr $\leftarrow$ ctr $+ 1$
29:   end while
30:   if ctr $\geq S_{\max}$ then
31:     returnfalse
32:   end if
33: end while
34: return false
```

E. Tunnel Pattern

While the wriggle pattern locally explores the neighborhood *inside* the path restriction, we often encounter situations where we find it advantageous to momentarily step *outside* the path restriction to overcome an infeasible region. From the perspective of the path restriction, we "tunnel" through the infeasible region, which we therefore refer to as the tunnel pattern. With the tunnel pattern, we assume to be located at a local minimum p_1 as shown in Fig. 8. To resolve this situation, we try to find the next valid state p_2 while keeping the fiber element constant. We then try to connect p_1 to p_2 by sampling valid states in a smoothly increasing neighborhood of the base space and a constant neighborhood in fiber space. While p_2 is not reached, we accept new states if they decrease the distance to p_2.

We show the pseudocode in Algorithm 7. We first search for a tunnel ending state x_{End} at base path location l_{End} (Line 1). To find the tunnel ending, we step forward along the base path without changing the fiber until we find a valid state. We then try to connect the head state x_H to the tunnel ending state x_{End}. We use a while loop to move along the relevant base path segment from the head location l to the tunnel end location l_{End} (Line 6). We first check if we can connect the head state to the tunnel end state (Line 7). If true, we add a new edge into the graph (Line 8), set the head to the tunnel ending state (Line 9), and return true (Line 10). Otherwise, we step forward along the base path with step size $\delta_{X_{k-1}}$ (Line 12) and query the base state at l (Line 13). Instead of using the base state exactly, we use a smoothly increasing neighborhood parameter ϵ. The value of ϵ depends on the counter CTR and smoothly interpolates between 0 and $10\delta_{X_{k-1}}$ using an Hermite polynomial [16] (Line 14). We then attempt to make a step toward the tunnel ending for a maximum of $S_{\max}$ attempts (Line 16). We do this by sampling a base space element (Line 17) and a fiber element (Line 18). We then lift the state (Line 19) and check for validity (Line 20). If the new state is valid, its distance is closer to the tunnel ending and we can connect it to the head state (Line 22), we add a new edge to the graph (Line 23), set the head state to the new state (Line 24), and continue forward (Line 25). If we fail to find a better sample for $S_{\max}$ attempts, we return false (Line 30-32). We also return false if we reach the base path location l_{End} without having a valid connection (Line 34).

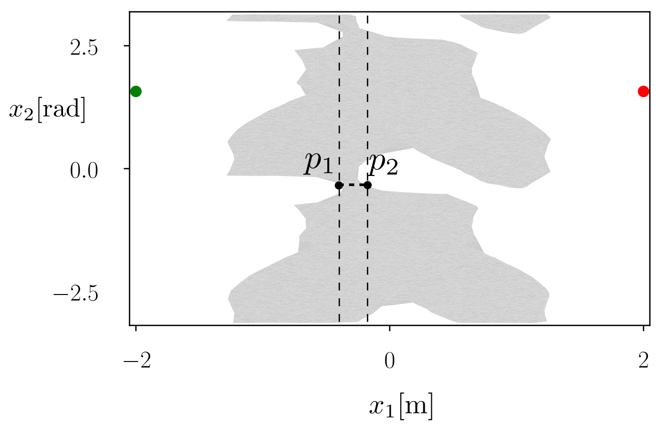

Fig. 8. Tunnel pattern to traverse a narrow passage: Given two feasible states p_1 and p_2, we connect them by momentarily leaving the path restriction to circumnavigate the infeasible region between them.

V. Evaluations

To evaluate our pattern dance algorithm, we integrate it into the multilevel planner QRRT, QRRT*, QMP, and QMP*, as we discussed in Section IV-B. We then conduct two comparisons. First, we compare our planner to 36 available planning algorithms in the OMPL [64] on 7 challenging environments as shown in Fig. 9. For each algorithm, we use the abbreviated name. For a full list of algorithms with full names and associated publication, see [69] and the OMPL documentation [94]. Second, we compare the multilevel planner with the pattern dance algorithm to an older version of the same multilevel planner, where we use a recursive sidestepping algorithm to quickly find sections [69].

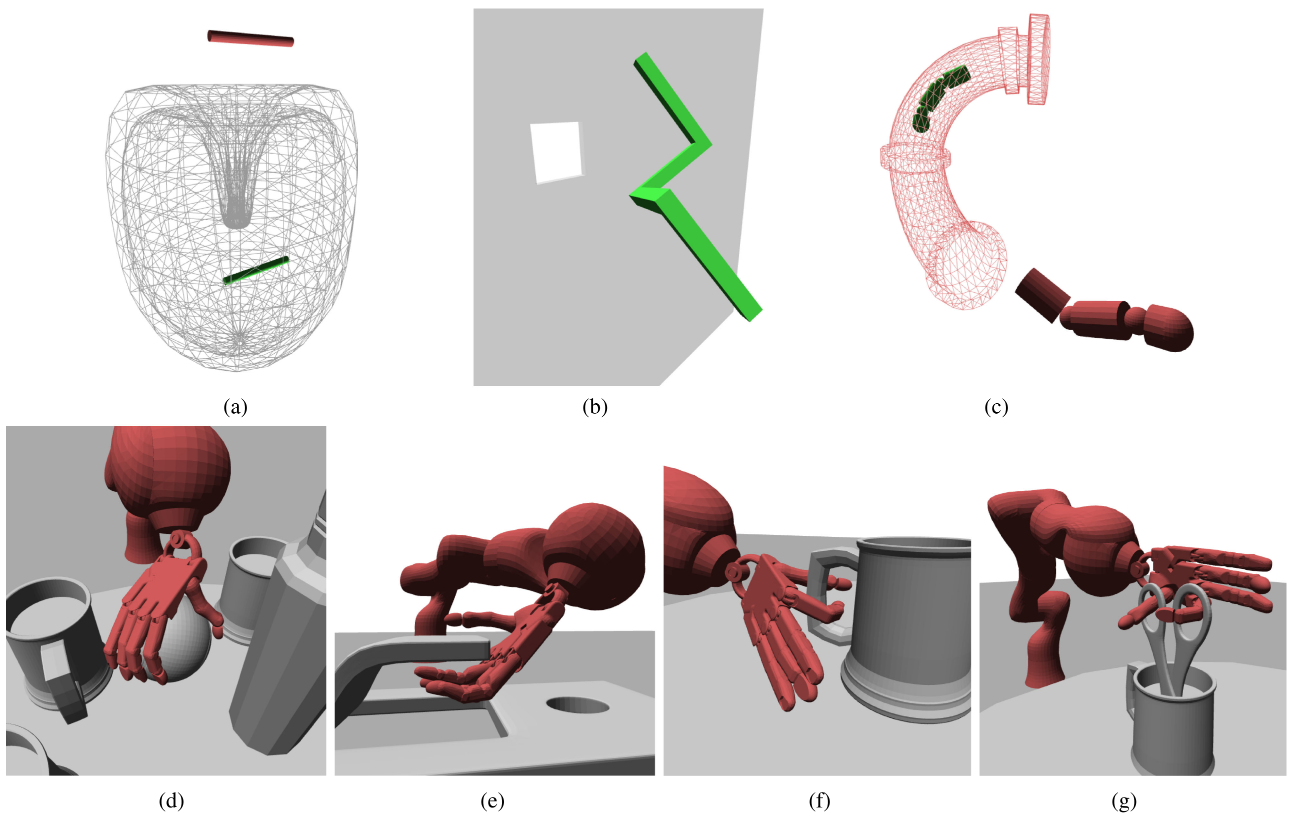

Fig. 9. Scenarios for evaluations. The task is to move the robot from the start state (green) to the goal state (red). Top row (left to right): Bugtrap (6-dof), double L shape (6-DoF) (goal configuration not shown) and chain egress (10-DoF). Bottom row: Overhand, underhand, single-finger, and double-finger pregrasp (each 37-DoF) (start configurations not shown). (a) 06D Bugtrap. (b) 06D Double Lshape. (c) 10D Chain Egress. (d) 37D ShadowHand Ball. (e) 37D ShadowHand Metal. (f) 37D ShadowHand Mug. (g) 37D ShadowHand Scissor.

A. Evaluation Metric

To evaluate, we use a 8 GB RAM 4-core 2.5 GHz laptop running Ubuntu 16.04. For each experiment, we use a minimum length cost (for planner which support cost functions) and we let each planner run 10 times with a cutoff time limit of 60 seconds. We then report on the average runtime over those 10 runs. We show the results in Table II and Table III.

Concerning the results, there are two notes of caution. First, we let each OMPL planner run out-of-the-box without any parameter tuning. Further tuning of parameters could potentially improve results significantly. Second, due to the high number of planner and scenarios, we let each planner run only ten times and take the average. However, averaging over ten runs might exhibit more variance and thereby create more outliers.

B. Six-DoF Bugtrap

For the first evaluation, we use the Bugtrap scenario [55] [see Fig. 9(a)]. The lowest runtime we found in the literature is 22.17 s for a version of the selective-retraction-RRT [55], [106] . However, this runtime is not directly comparable due to different hardware, implementation, parameters, and operating systems. To relax the problem, we use an inscribed sphere at the center of the cylindrical bug as shown in Fig. 10(d) and (g).

We show the results of our evaluation in Table II. The best performing planner is QMP (3rd planner in table) with 0.51 s followed by QMP* (4) with 0.90 s and QRRT (1) with 4.45 s. We also see good performance of the BiTRRT (13) planner [44] with 11.54 s. We note that the QRRT* (2) algorithm requires 24.87 s, which we believe to be caused by the additional burden of rewiring the tree [69], [85].

C. Six-DoF Double L Shape

In the next evaluation, we like to show that the section patterns are not specific to the cylindrical geometry, but are more widely applicable to other rigid bodies. As demonstration, we use the double L-shape scenario [98], where two L-shape bodies are connected to each other as shown in Fig. 9(b). The task is to move through a vertical wall with a small quadratic hole. We use a two-level relaxation by using an inscribed sphere as shown in Fig. 10(h) and (e). To make our method more robust against base paths too close to obstacles, we increase the size of the sphere slightly to increase clearance from obstacles.

Our evaluation shows that QMP performs best with 1.27 s followed by QMP* (1.63 s), QRRT (1.86 s), and QRRT* (2.00 s). The next best planner from OMPL is LBKPIECE1 (38) with 49.79 s.

D. Ten-DoF Chain Egress

In the third evaluation, we like to increase the complexity by considering an articulated chain (10-DoF) as shown in Fig. 9(c).

TABLE II
RUNTIME (S) OF MOTION PLANNER ON THE SCENARIOS FROM FIG. 9, EACH AVERAGED OVER 10 RUNS WITH CUTOFF TIME LIMIT OF 60 S. AN ENTRY – MEANS THAT PLANNER DOES NOT SUPPORT THE PARTICULAR STATE SPACE

	Runtime in seconds (10 run average)	**06D Bugtrap**	**06D Double Lshape**	**10D Chain Egress**	**37D ShadowHand Ball**	**37D ShadowHand Metal**	**37D ShadowHand Mug**	**37D ShadowHand Scissor**
1	QRRT (**ours**)	4.45	1.86	**0.55**	2.01	35.63	19.80	60.00
2	QRRT* (**ours**)	24.87	2.00	0.56	25.35	43.95	60.00	60.00
3	QMP (**ours**)	**0.51**	**1.27**	1.91	0.86	18.98	**1.20**	**14.52**
4	QMP* (**ours**)	0.90	1.63	7.29	**0.86**	**1.94**	1.63	37.27
5	RRT	60.00	60.00	49.77	60.00	60.00	60.00	60.00
6	RRTConnect	60.00	60.00	60.00	*1.70*	*8.16*	57.38	60.00
7	RRT#	60.00	60.00	45.43	60.00	60.00	60.00	60.00
8	RRT*	60.00	60.00	51.74	60.00	60.00	60.00	60.00
9	RRTXstatic	60.00	60.00	50.49	60.00	60.00	60.00	60.00
10	LazyRRT	60.00	60.00	55.56	60.00	60.00	60.00	60.00
11	TRRT	60.00	60.00	*0.81*	42.08	60.00	60.00	60.00
12	BiTRRT	11.54	54.30	*4.57*	60.00	60.00	60.00	60.00
13	LBTRRT	60.00	60.00	60.00	60.00	60.00	60.00	60.00
14	RLRT	60.00	60.00	51.39	*3.68*	28.47	60.00	60.00
15	BiRLRT	60.00	57.40	60.00	*1.52*	25.60	60.00	60.00
16	pRRT	60.00	60.00	49.41	60.00	60.00	60.00	60.00
17	FMT	60.00	60.00	60.00	60.00	60.00	60.00	60.00
18	BFMT	60.00	50.34	60.00	60.00	60.00	60.00	60.00
19	PRM	60.00	56.47	60.00	37.25	52.72	60.00	60.00
20	PRM*	60.00	57.80	60.00	34.24	50.04	60.00	60.00
21	LazyPRM	60.00	60.00	60.00	60.00	60.00	60.00	60.00
22	LazyPRM*	60.00	60.00	60.00	54.06	60.00	60.00	60.00
23	SPARS	60.00	59.73	60.00	60.00	60.00	60.00	60.00
24	SPARStwo	60.00	54.69	60.00	60.00	60.00	60.00	60.00
25	SST	60.00	60.00	60.00	60.00	60.00	60.00	60.00
26	EST	60.00	60.00	50.46	24.96	45.64	60.00	60.00
27	BiEST	60.00	60.00	59.85	29.79	33.36	60.00	60.00
28	InformedRRT*	60.00	60.00	-	60.00	60.00	60.00	60.00
29	SORRT*	60.00	60.00	-	60.00	60.00	60.00	60.00
30	kBIT*	60.00	60.00	-	34.17	46.44	60.00	60.00
31	kABIT*	60.00	60.00	-	50.28	44.56	60.00	60.00
32	AIT*	60.00	60.00	-	55.35	60.00	60.00	60.00
33	STRIDE	60.00	60.00	-	29.58	48.98	60.00	60.00
34	ProjEST	60.00	60.00	-	47.77	60.00	60.00	60.00
35	PDST	60.00	60.00	-	*3.25*	54.42	60.00	60.00
36	KPIECE1	60.00	60.00	-	*6.27*	32.48	60.00	60.00
37	BKPIECE1	60.00	60.00	-	52.35	60.00	60.00	60.00
38	LBKPIECE1	60.00	49.79	-	60.00	60.00	60.00	60.00
39	SBL	60.00	50.30	-	60.00	60.00	60.00	60.00
40	CForest	60.00	60.00	-	60.00	60.00	60.00	60.00

A bold entry signifies the lowest runtime.

The task is to remove the chain from a pipe, a typical egress scenario. Note that for such systems, we can find analytical feasible path sections if we assume the base path of the head to be curvature constrained [68]. However, we will not make such assumption in this article.

TABLE III
COMPARISON OF MULTILEVEL PLANNERS WITH SIDESTEPPING [69] VERSUS MULTILEVEL PLANNER WITH OUR PATTERN DANCE ALGORITHM

	Runtime in seconds (10 run average)	**06D Bugtrap**	**06D Double Lshape**	**10D Chain Egress**	**37D ShadowHand Ball**	**37D ShadowHand Metal**	**37D ShadowHand Mug**	**37D ShadowHand Scissor**
1	QMP (**ours**)	**0.51**	**1.27**	**1.91**	**0.86**	**18.98**	**1.20**	**14.52**
2	QMP (SideStepping)	60.00	26.08	60.00	1.07	55.37	6[a]	60.00
3	QMP* (**ours**)	**0.90**	**1.63**	**7.29**	**0.86**	**1.94**	**1.63**	**37.27**
4	QMP* (SideStepping)	60.00	30.11	60.00	1.76	60.00	12[a]	60.00
5	QRRT (**ours**)	**4.45**	**1.86**	**0.55**	**2.01**	**35.63**	**19.80**	60.00
6	QRRT (SideStepping)	60.00	27.72	9.14	18.65	60.00	44[a]	60.00
7	QRRT* (**ours**)	**24.87**	**2.00**	**0.56**	**25.35**	**43.95**	60.00	60.00
8	QRRT* (SideStepping)	60.00	60.00	16.42	42.33	54.05	**48**[a]	60.00

[a]Taken from [69].

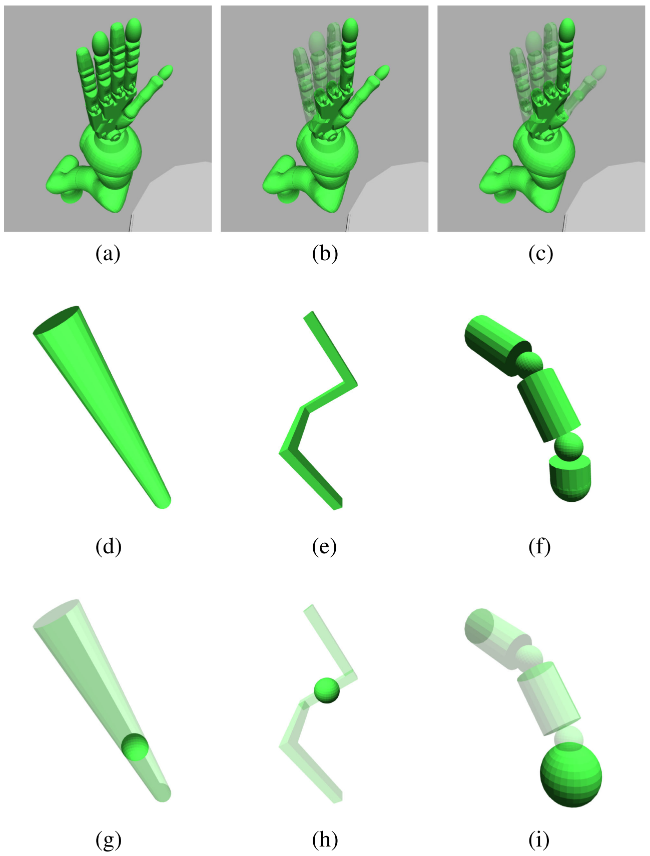

Fig. 10. Multilevel abstraction using simplified models. (a) Shadow hand level 3 $\mathbb{R}^{37}$. (b) Shadow hand level 2 $\mathbb{R}^{18}$. (c) Shadow hand level 1 $\mathbb{R}^{13}$. (d) Bugtrap level 2 $SE(3)$. (e) Double Lshape level 2 $SE(3)$. (f) Articulated chain level 2 $SE(3) \times \mathbb{R}^6$. (g) Bugtrap level 1 $\mathbb{R}^3$. (h) Double Lshape level 1 $\mathbb{R}^3$. (i) Articulated chain level 1 $\mathbb{R}^3$.

Algorithm 7: TripleStepPattern(H, x).

```
Parameters: Base space step size $\delta_{X_{k-1}}$
1:  $x_H \leftarrow State(H)$
2:  $l \leftarrow Location(H)$
3:  $x_{F_1} \leftarrow ProjectFiber(H)$
4:  $x_{F_2} \leftarrow ProjectFiber(x)$
5:  $x_{F_m} \leftarrow Steer(x_{F_1}, x_{F_2}, 0.5)$                ▷Midpoint
6:  while $l > \delta_{X_{k-1}}$ do
7:    $l \leftarrow l - \delta_{X_{k-1}}$
8:    $x_B \leftarrow BasePathAt(p, l)$
9:    $x_{\text{mid}} \leftarrow Lift(x_B, x_{F_m})$
10:   if $IsValid(x_{\text{mid}})$ then
11:     $x_1 \leftarrow Lift(x_B, x_{F_1})$
12:     $x_2 \leftarrow Lift(x_B, x_{F_2})$
13:     if $CheckMotion(x_1, x_2)$ then
14:       if $CheckMotion(x_H, x_1)$ then
15:         if $CheckMotion(x_2, x)$ then
16: $\mathbf{G}_k \leftarrow \mathbf{G}_k \cup \{x_H, x_1\}$
17: $\mathbf{G}_k \leftarrow \mathbf{G}_k \cup \{x_1, x_2\}$
18: $\mathbf{G}_k \leftarrow \mathbf{G}_k \cup \{x_2, x\}$
19: $UpdateHead(h, x)$
20: return true
21:         end if
22:       end if
23:       break                                  ▷End While Loop
24:     end if
25:   end if
26: end while
27: return false
```

To relax the problem, we use an inscribed sphere in the head of the chain as shown in Fig. 10(i) and (f). As in the case of the double L-shape, we slightly increase the size of the sphere to make our method more robust against base paths too close to obstacles.

In our evaluations, we show that QRRT performs best with 0.55 s followed by QRRT* (0.56 s). The next best planners are TRRT (11) (0.81 s), QMP (1.91), BiTRRT (12) (4.57 s), and QMP* with 7.29 s. Note that there are 12 OMPL planner which cannot address this problem, because they do not support compound state spaces or do not have dedicated projection functions for such spaces.

E. *Thirty-Seven-DoF Pregrasp*

For the next evaluations, we compute (pre)grasping paths for a ShadowHand mounted on a KUKA LWR robot. The tasks are to compute an overhand grasp on a ball [see Fig. 9(d)], an underhand grasp on a metal piece [see Fig. 9(e)], a single-finger precision grasp on a mug [see Fig. 9(f)] and a double-finger precision grasp on a scissor [see Fig. 9(g)]. The starting state for all scenarios is an upright position of the arm with hand being open, as shown in Fig. 10(a). To relax the problem, we use a three-level abstraction by first removing three fingers [see Fig. 10(b)] and subsequently removing the thumb [see Fig. 10(c)] of the hand.

Our evaluations show the following results. First, for the Ball scenario, we see that QMP and QMP* perform best with 0.86 s. The next best planner is the OMPL planner BiRLRT (15) [58] with 1.52 s, QRRT with 2.01 s and RRTConnect (6) with 1.70 s. We note that also the planner PDST (35) [52], RLRT (14) [58], and KPIECE1 (36) [93] perform competitively with 3.25, 3.68, and 6.27 s, respectively. The planner QRRT* does not perform well on this problem instance with 25.35 s, due to similar problems as on the Bugtrap scenario. Second, for the underhand grasp on the metal piece, we see that QMP* performs best with 1.94 s followed by RRTConnect (6) with 8.16 s and QMP with 18.98 s. We will address the discrepancy between QMP and QMP* further in Section VI. Third, for the single-finger precision grasp on the mug, we observe that QMP performs best with 1.20 s followed by QMP* with 1.63 s. While QRRT performs significantly worse (19.80 s), QRRT* was not able to solve this problem (60.00 s). Fourth, for the double-finger precision grasp on the scissor, we observe that QMP performs best with 14.52 s followed by QMP* with 37.27 s. No other planner is able to solve this problem. We will further discuss the high runtime of both QMP and QMP* in detail in Section VI.

VI. Limitations and Discussion

While our evaluations support the usage of section patterns for narrow passage planning problems, we also like to point out two limitations of our approach. To each limitation, we will discuss possible ways to eventually address and resolve the limitation.

A. *Increased Runtime on Metal and Scissor Scenario*

The first limitation is the increased runtime of our planner on the 37D ShadowHand Scissor and the Metal scenario. We distinguish between two subproblems. First, we observe that QRRT and QRRT* have a runtime of 60 s on the Scissor scenario. Both scenarios, however, are ingress scenarios, where the planner needs to find a narrow passage on the base space to enter the goal region, which is challenging for RRT-like algorithms [49] and could be addressed using a bidirectional version of QRRT.

Second, we observe that QMP and QMP* require 14.52 and 37.27 s to solve the Scissor scenario and that QMP requires 18.98 s to solve the Metal scenario. To explain this rather large increase in runtime, we have a closer look at the individual runtimes, which we show in Table IV. We can observe that both planner exhibit one of two outputs. Either, they quickly return a solution (usually less than 3 s, always less than 10 s) or they fail and time out at 60 s (three/two times for QMP, zero/six times for QMP*). To us, this indicates that both algorithms might be sensitive to the base space path. If the base path is not smooth enough, has kinks in it or is too close to obstacles, then we might not be able to solve it with the pattern dance algorithm. We could address this problem in the future by either additional smoothing of the base space path [101], by introducing conservative heuristics [13], or by switching to a different relaxed model [92].

TABLE IV
RUNTIME (S) FOR QMP AND QMP* ON EACH RUN. AVERAGE RUNTIMES ARE 18.98 S/1.94 S (QMP/QMP*) FOR THE METAL SCENARIOS AND 14.52 S/37.27 S FOR THE SCISSOR SCENARIO

Run	1	2	3	4	5	6	7	8	9	10
					37D ShadowHand Metal Scenario					
QMP	1.53	1.11	1.20	0.99	1.06	60.00	60.00	2.93	1.02	60.00
QMP*	0.98	1.15	0.93	1.23	2.73	1.13	1.03	7.61	0.98	1.65
					37D ShadowHand Scissor Scenario					
QMP	1.45	1.50	2.14	2.17	60.00	60.00	2.44	7.49	1.51	6.51
QMP*	60.00	60.00	2.22	60.00	6.27	60.00	60.00	60.00	1.92	2.30

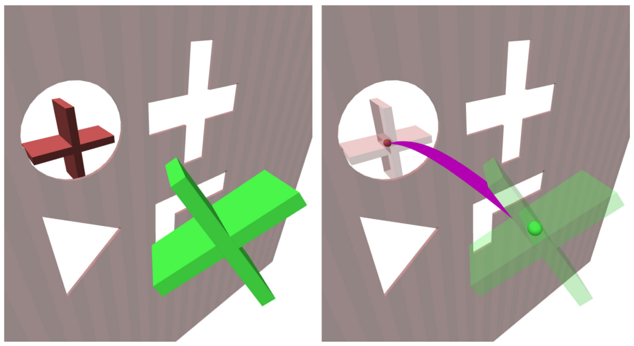
Fig. 11. Limitations of section pattern approach. Base path does not admit a feasible path section. See text for clarification.

B. Base Path Does Not Admit a Feasible Section

While all multilevel planner are probabilistically complete, we often need the pattern dance algorithm to efficiently solve a problem. However, we might encounter scenarios, where the base path does not admit a feasible path section. Such a situation is shown in Fig. 11. The scenario depicts an X-shaped robot, which has to traverse a shape-sorter box with different openings, which we relax by inscribing a sphere (right). Planning for the spherical robot might produce a base path going through the wrong hole. Such a base path does not admit a feasible path section, meaning there are no paths along the path restriction of the base path to traverse toward the goal. While multilevel planner are probabilistically complete and would eventually resolve the situation, we would not be able to solve this situation using our pattern dance algorithm. To address such situations, we could either compute several base paths [7], [33], [66], [70], [75], [102], and consider them as a multiarm bandit problem over path restrictions [50] or we could automatically choose an alternative relaxation using either a meta-heuristic [9] or a brute-force search [65].

VII. CONCLUSION

We developed the pattern dance algorithm, which took a base space path as input and efficiently searched for a feasible section in its path restriction, four dedicated section patterns, which we named Manhattan, Wriggle, Tunnel, and Triple step. We showed in evaluations, that our pattern dance algorithm successfully coordinated section patterns and outperformed a similar sidestepping algorithm [69]. We then showed that multilevel motion planning algorithms using our pattern dance algorithm outperformed classical planner from the OMPL library on challenging narrow passage scenarios including the Bugtrap, chain egress, and precision grasping. With some exceptions, we often observed runtime improvements by one to two orders of magnitudes.

While we demonstrated to efficiently solve narrow passage problems, we also pointed out two limitations. First, we observed an increased runtime in some planning instances. We could address this problem by either optimizing the base path [108], by improved neighborhood modeling [51] or by learning the section patterns themselves [42]. Second, we cannot handle cases where the base path does not admit a path section. We could addressed this problem by computing multiple base paths [66], [70], [102] or using more informed graph restriction sampling methods [65].

Despite limitations, we believe to have contributed a novel solution method which we can use to efficiently find sections over base path restrictions. We believe our method to be a promising tool to further probe, understand, and efficiently exploit high-dimensional state spaces.

REFERENCES

[1] S. Aine, S. Swaminathan, V. Narayanan, V. Hwang, and M. Likhachev, "Multi-heuristic A*," *Int. J. Robot. Res.*, vol. 35, no. 1–3, pp. 224–243, 2016.
[2] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo, "OBPRM: An obstacle-based PRM for 3D workspaces," in *Proc. Workshop Algorithmic Found. Robot.*, 1998, pp. 155–168.
[3] B. Baginski, "Local motion planning for manipulators based on shrinking and growing geometry models," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1996, pp. 3303–3308.
[4] J. Basch, L. J. Guibas, D. Hsu, and A. T. Nguyen, "Disconnection proofs for motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2001, pp. 1765–1772.
[5] O. B. Bayazit, D. Xie, and N. M. Amato, "Iterative relaxation of constraints: a framework for improving automated motion planning," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2005, pp. 3433–3440.
[6] K. E. Bekris and R. Shome, "Asymptotically optimal sampling-based planners," 2020, *arXiv:1911.04044*.
[7] S. Bhattacharya and R. Ghrist, "Path homotopy invariants and their application to optimal trajectory planning," *Ann. Math. Artif. Intell.*, vol. 84, no. 3/4, pp. 139–160, 2018.
[8] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2000, vol. 1, pp. 521–528.
[9] M. Brandao and I. Havoutis, "Learning sequences of approximations for hierarchical motion planning," in *Proc. Int. Conf. Autom. Plan. Scheduling*, 2020, vol. 30, pp. 508–516.
[10] T. Bretl, "Motion planning of multi-limbed robots subject to equilibrium constraints: The free-climbing robot problem," *Int. J. Robot. Res.*, vol. 25, no. 4, pp. 317–342, 2006.
[11] O. Brock and L. E. Kavraki, "Decomposition-based motion planning: A. framework for real-time motion planning in high-dimensional configuration spaces," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2001, vol. 2, pp. 1469–1474.
[12] H.-J. Bungartz and M. Griebel, "Sparse grids," *Acta Numerica*, vol. 13, pp. 147–269, 2004.
[13] I. Chatterjee, M. Likhachev, A. Khadke, and M. Veloso, "Speeding up search-based motion planning via conservative heuristics," in *Proc. Int. Conf. Autom. Plan. Scheduling*, 2019, pp. 674–679.
[14] J. Cortés, L. Jaillet, and T. Siméon, "Disassembly path planning for complex articulated objects," *IEEE Trans. Robot.*, vol. 24, no. 2, pp. 475–481, Apr. 2008.
[15] J. C. Culberson and J. Schaeffer, "Pattern databases," *Comput. Intell.*, vol. 14, no. 3, pp. 318–334, 1998.

[16] C. De Boor, K. Hollig, and M. Sabin, "High accuracy geometric hermite interpolation," *Comput. Aided Geometric Des.*, Elsevier, vol. 4, no. 4, pp. 269–278, 1987.
[17] J. Denny, R. Sandström, A. Bregger, and N. M. Amato, "Dynamic region-biased rapidly-exploring random trees," *in Algorithmic Foundations of Robotics XII*. Berlin, Germany: Springer, 2020, pp. 640–655.
[18] D. Driess, J.-S. Ha, and M. Toussaint, "Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image," in *Proc. Robot.: Sci. Syst.*, 2020.
[19] W. Du, S.-K. Kim, O. Salzman, and M. Likhachev, "Escaping local minima in search-based planning using soft duplicate detection," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2019, pp. 2365–2371.
[20] W. Du, F. Islam, and M. Likhachev, "Multi-resolution A*," 2020, *arXiv:2004.06684*.
[21] S. Edelkamp and S. Schroedl, *Heuristic Search: Theory and Applications*. New York, NY, USA: Elsevier, 2011.
[22] S. Edelkamp, P. Kissmann, and Á. Torralba, "Symbolic A* search with pattern databases and the merge-and-shrink abstraction," in *Proc. Eur. Conf. Artif. Intell.*, 2012, pp. 306–311.
[23] P. Ferbach and J. Barraquand, "A method of progressive constraints for manipulation planning," *IEEE Trans. Robot.*, vol. 13, no. 4, pp. 473–485, Aug. 1997.
[24] D. Ferguson, M. Likhachev, and A. Stentz, "A guide to heuristic-based path planning," in *Proc. Int. Conf. Autom. Plan. Scheduling*, 2005, pp. 9–18.
[25] M. Fu, A. Kuntz, O. Salzman, and R. Alterovitz, "Toward asymptotically-optimal inspection planning via efficient near-optimal graph search," in *Proc. Robot.: Sci. Syst.*, Jun. 2019.
[26] J. D. Gammell and M. P. Strub, "A survey of asymptotically optimal sampling-based motion planning methods," 2020, *arXiv:2009.10484*.
[27] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2014, pp. 2997–3004.
[28] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, "Batch informed trees (BIT*): Informed asymptotically optimal anytime search," *Int. J. Robot. Res.*, vol. 39, no. 5, pp. 543–567, 2020.
[29] M. B. Giles, "Multilevel Monte Carlo methods," *Acta Numerica*, vol. 24, pp. 259–328, 2015.
[30] F. Glover and M. Laguna, "Tabu search," in *Handbook of Combinatorial Optimization*. Berlin, Germany: Springer, 1998, pp. 2093–2229.
[31] K. Gochev, A. Safonova, and M. Likhachev, "Planning with adaptive dimensionality for mobile manipulation," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2012, pp. 2944–2951.
[32] M. X. Grey, A. D. Ames, and C. K. Liu, "Footstep and motion planning in semi-unstructured environments using randomized possibility graphs," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 4747–4753.
[33] J.-S. Ha, S.-S. Park, and H.-L. Choi, "Topology-guided path integral approach for stochastic optimal control in cluttered environment," *IEEE Robot. Auton. Syst.*, vol. 113, pp. 81–93, Mar. 2019.
[34] N. Haghtalab, S. Mackenzie, A. D. Procaccia, O. Salzman, and S. S. Srinivasa, "The provable virtue of laziness in motion planning," *Proc. Twenty-Eighth Int. Joint Conf. Artif. Intell., IJCAI-19*, 2019, pp. 6161–6165, doi: 10.24963/ijcai.2019/855.
[35] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, Jul. 1968.
[36] V. N. Hartmann, O. S. Oguz, D. Driess, M. Toussaint, and A. Menges, "Robust task and motion planning for long-horizon architectural construction planning," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2020.
[37] K. Hauser, "Fast interpolation and time-optimization with contact," *Int. J. Robot. Res.*, vol. 33, no. 9, pp. 1231–1250, 2014.
[38] B. Hou, S. Choudhury, G. Lee, A. Mandalika, and S. S. Srinivasa, "Posterior sampling for anytime motion planning on graphs with expensive-to-evaluate edges," 2020, *arXiv:2002.11 853*.
[39] D. Hsu, T. Jiang, J. Reif, and Z. Sun, "The bridge test for sampling narrow passages with probabilistic roadmap planners," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2003, vol. 3, pp. 4420–4426.
[40] S. Hu and N. R. Sturtevant, "Direction-optimizing breadth-first search with external memory storage," in *Proc. Int. Joint Conf. Artif. Intell.*, 2019, pp. 1258–1264.
[41] D. Husemoller, *Fibre Bundles*, vol. 5. Berlin, Germany: Springer, 1966.
[42] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 7087–7094.
[43] L. Jaillet and J. M. Porta, "Path planning under kinematic constraints by rapidly exploring manifolds," *IEEE Trans. Robot.*, vol. 29, no. 1, pp. 105–117, Feb. 2013.
[44] L. Jaillet, J. Cortés, and T. Siméon, "Sampling-based path planning on configuration-space costmaps," *IEEE Trans. Robot.*, vol. 26, no. 4, pp. 635–646, Aug. 2010.
[45] S. S. Joshi, S. Hutchinson, and P. Tsiotras, "Time-informed exploration for robot motion planning," 2020, *arXiv:2004.05 241*.
[46] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, 2011.
[47] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.
[48] M. Kleinbort, O. Salzman, and D. Halperin, "Collision detection or nearest-neighbor search? On the computational bottleneck in sampling-based motion planning," *in Algorithmic Foundations of Robotics XII*. Berlin, Germany: Springer, 2020, pp. 624–639.
[49] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2000, vol. 2, pp. 995–1001.
[50] H. Kurniawati and D. Hsu, "Workspace-based connectivity oracle: An adaptive sampling strategy for PRM planning," *in Algorithmic Foundation of Robotics VII*. Berlin, Germany: Springer, 2008, pp. 35–51.
[51] B. Lacevic and D. Osmankovic, "Improved C-space exploration and path planning for robotic manipulators using distance information," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2020, pp. 1176–1182.
[52] A. M. Ladd and L. E. Kavraki, "Fast tree-based exploration of state space for robots with dynamics," *in Algorithmic Foundations of Robotics VI*. Berlin, Germany: Springer, 2004.
[53] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.
[54] J. M. Lee, *Introduction to Smooth Manifolds*. New York, NY, USA: Springer, 2003.
[55] J. Lee, O. Kwon, L. Zhang, and S.-E. Yoon, "SR-RRT: Selective retraction-based RRT planner," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2012, pp. 2543–2550.
[56] Z. Littlefield and K. E. Bekris, "Efficient and asymptotically optimal kinodynamic motion planning via dominance-informed regions," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2018, pp. 1–9.
[57] T. Lozano-Perez, "A simple motion-planning algorithm for general robot manipulators," *IEEE J. Robot. Autom.*, vol. 3, no. 3, pp. 224–238, Jun. 1987.
[58] R. Luna, M. Moll, J. Badger, and L. E. Kavraki, "A scalable motion planner for high-dimensional kinematic systems," *Int. J. Robot. Res.*, vol. 39, no. 4, pp. 361–388, 2020.
[59] Y. Luo, H. Bai, D. Hsu, and W. S. Lee, "Importance sampling for online planning under uncertainty," *Int. J. Robot. Res.*, vol. 38, no. 2-3, pp. 162–181, 2019.
[60] J. Mainprice, N. Ratliff, M. Toussaint, and S. Schaal, "An interior point method solving motion planning problems with narrow passages," in *Proc. IEEE Int. Conf. Robot Human Interactive Commun.*, 2020, pp. 547–552.
[61] M. Manak, "Voronoi-based detection of pockets in proteins defined by large and small probes," *J. Comput. Chem.*, vol. 40, no. 19, pp. 1758–1771, 2019.
[62] A. Mandalika, S. Choudhury, O. Salzman, and S. Srinivasa, "Generalized lazy search for robot motion planning: Interleaving search and edge evaluation via event-based toggles," in *Proc. Int. Conf. Autom. Plan. Scheduling*, 2019, pp. 745–753.
[63] Z. McCarthy, T. Bretl, and S. Hutchinson, "Proving path non-existence using sampling and alpha shapes," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2012, pp. 2563–2569.
[64] M. Moll, I. A. Şucan, and L. E. Kavraki, "Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization," *IEEE Robot. Autom. Mag.*, vol. 22, no. 3, pp. 96–102, Sep. 2015.
[65] A. Orthey and M. Toussaint, "Rapidly-exploring quotient-space trees: Motion planning using sequential simplifications," *Int. Symp. Robot. Res.*, 2019.
[66] A. Orthey and M. Toussaint, "Visualizing local minima in multi-robot motion planning using multilevel morse theory," in *Proc. Workshop Algorithmic Found. Robot.*, 2020.

[67] A. Orthey, A. Escande, and E. Yoshida, "Quotient-space motion planning," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2018, pp. 8089–8096.

[68] A. Orthey, O. Roussel, O. Stasse, and M. Taïx, "Motion planning in irreducible path spaces," *IEEE Robot. Auton. Syst.*, vol. 109, pp. 97–108, Nov. 2018.

[69] A. Orthey, S. Akbar, and M. Toussaint, "Multilevel motion planning: A. fiber bundle formulation," 2020, *arXiv:2007.09435 [cs.RO]*.

[70] T. Osa, "Multimodal trajectory optimization for motion planning," *Int. J. Robot. Res.*, vol. 39, no. 8, pp. 983–1001, 2020.

[71] L. Palmieri, S. Koenig, and K. O. Arras, "RRT-based nonholonomic motion planning using any-angle path biasing," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2016, pp. 2775–2781.

[72] J. Pearl, "Heuristics: Intelligent search strategies for computer problem solving," Reading, MA, USA: Addision Wesley, 1984.

[73] S. M. Persson and I. Sharf, "Sampling-based A* algorithm for robot path-planning," *Int. J. Robot. Res.*, vol. 33, no. 13, pp. 1683–1708, 2014.

[74] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Discrete search leading continuous exploration for kinodynamic motion planning," in *Proc. Robot.: Sci. Syst.*, 2007, pp. 326–333.

[75] F. T. Pokorny, M. Hawasly, and S. Ramamoorthy, "Topological trajectory classification with filtrations of simplicial complexes and persistent homology," *Int. J. Robot. Res.*, vol. 35, no. 1–3, pp. 204–223, 2016.

[76] A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip, "Motion planning networks: Bridging the gap between learning-based and classical motion planners," *IEEE Trans. Robot.*, vol. 37, no. 1, pp. 48–66, Feb. 2021.

[77] W. Reid, R. Fitch, A. H. Göktoğan, and S. Sukkarieh, "Sampling-based hierarchical motion planning for a reconfigurable wheel-on-leg planetary analogue exploration rover," *J. Field Robot.*, vol. 37, no. 5, pp. 786–811, 2019.

[78] W. Reid, R. Fitch, A. H. Göktoğgan, and S. Sukkarieh, "Motion planning for reconfigurable mobile robots using hierarchical fast marching trees," *in Algorithmic Foundations of Robotics XII*. Berlin, Germany: Springer, 2020, pp. 656–671.

[79] M. Rickert, A. Sieverling, and O. Brock, "Balancing exploration and exploitation in sampling-based motion planning," *IEEE Trans. Robot.*, vol. 30, no. 6, pp. 1305–1317, Dec. 2014.

[80] J. Röwekämper, G. Tipaldi, and W. Burgard, "Learning to guide random tree planners in high dimensional spaces," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2013, pp. 1752–1757.

[81] M. Saha, J.-C. Latombe, Y.-C. Chang, and F. Prinz, "Finding narrow passages with probabilistic roadmaps: The small-step retraction method," *Auton. Robots*, vol. 19, no. 3, pp. 301–319, 2005.

[82] B. Sakcak, L. Bascetta, G. Ferretti, and M. Prandini, "An admissible heuristic to improve convergence in kinodynamic planners using motion primitives," *IEEE Control Systems Letters*, vol. 4, no. 1, pp. 175–180, Jan. 2020.

[83] B. Sakcak, L. Bascetta, G. Ferretti, and M. Prandini, "Sampling-based optimal kinodynamic planning with motion primitives," *Auton. Robots*, vol. 43, no. 7, pp. 1715–1732, 2019.

[84] O. Salzman, "Sampling-based robot motion planning," *Commun. ACM*, vol. 62, no. 10, pp. 54–63, 2019.

[85] O. Salzman and D. Halperin, "Asymptotically near-optimal RRT for fast, high-quality motion planning," *IEEE Trans. Robot.*, vol. 32, no. 3, pp. 473–483, Jun. 2016.

[86] O. Salzman, M. Hemmer, and D. Halperin, "On the power of manifold samples in exploring configuration spaces and the dimensionality of narrow passages," *in Algorithmic Foundations of Robotics X*, E. Frazzoli, T. Lozano-Perez, N. Roy, and D. Rus, Eds. Berlin, Germany: Springer, 2013, pp. 313–329.

[87] A. Schweikard and F. Schwarzer, "Detecting geometric infeasibility," *Artif. Intell.*, vol. 105, no. 1/2, pp. 139–159, 1998.

[88] S. Sekhavat, P. Svestka, J.-P. Laumond, and M. H. Overmars, "Multilevel path planning for nonholonomic robots using semiholonomic subsystems," *Int. J. Robot. Res.*, vol. 17, no. 8, pp. 840–857, 1998.

[89] A. Sintov, S. Macenski, A. Borum, and T. Bretl, "Motion planning for dual-arm manipulation of elastic rods," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 6065–6072, Oct. 2020.

[90] N. E. Steenrod, *The Topology of Fibre Bundles*. Princeton, NJ, USA: Princeton Univ. Press, 1951.

[91] M. P. Strub and J. D. Gammell, "Adaptively informed trees (AIT*): Fast asymptotically optimal path planning through adaptive heuristics," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2020, pp. 3191–3198.

[92] B. M. K. Styler and R. Simmons, "Plan-time multi-model switching for motion planning," in *Proc. Int. Conf. Autom. Plan. Scheduling*, 2017.

[93] I. A. Şucan and L. E. Kavraki, "A sampling-based tree planner for systems with complex dynamics," *IEEE Trans. Robot.*, vol. 28, no. 1, pp. 116–131, Feb. 2012.

[94] I. A. Şucan, M. Moll, and L. Kavraki, "The open motion planning library," *IEEE Robot. Autom. Mag.*, vol. 19, no. 4, pp. 72–82, Dec. 2012.

[95] J. Szkandera, I. Kolingerová, and M. Maňák, "Narrow passage problem solution for motion planning," *in Proc. Int. Conf. Comput. Sci.,*2020, pp. 459–470.

[96] S. Tonneau, A. D. Prete, J. Pettré, C. Park, D. Manocha, and N. Mansard, "An efficient acyclic contact planner for multiped robots," *IEEE Trans. Robot.*, vol. 34, no. 3, pp. 586–601, Jun. 2018.

[97] D. Uwacu, R. Rex, B. Wang, S. Thomas, and N. M. Amato, "Annotated-skeleton biased motion planning for faster relevant region discovery," 2020, *arXiv:2003.02176*.

[98] J. P. Van denBerg and M. H.Overmars, "Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners," *Int. J. Robot. Res.*, vol. 24, no. 12, pp. 1055–1071, 2005.

[99] A. Varava, J. F. Carvalho, F. T. Pokorny, and D. Kragic, "Free space of rigid objects: Caging, path non-existence, and narrow passage detection," *in International Journal of Robotics Research*. Berlin, Germany: Springer, 2020.

[100] S. Vats, V. Narayanan, and M. Likhachev, "Learning to avoid local minima in planning for static environments," in *Proc. Int. Conf. Autom. Plan. Scheduling*, 2017, pp. 572–576.

[101] E. Vidal, M. Moll, N. Palomeras, J. D. Hernández, M. Carreras, and L. E. Kavraki, "Online multilayered motion planning with dynamic constraints for autonomous underwater vehicles," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 8936–8942.

[102] V. Vonásek and R. Pěniĝka, "Sampling-based motion planning of 3D solid objects guided by multiple approximate solutions," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2019, pp. 1480–1487.

[103] Y. Yang and O. Brock, "Efficient motion planning based on disassembly," in *Proc. Robot.: Sci. Syst.*, Jun. 2005. [Online]. Available: http://roboticsproceedings.org/rss01/p14.html

[104] A. Yershova and S. M. LaValle, "Motion planning for highly constrained spaces," *Robot Motion Control*, vol. 396, pp. 297–306, 2009.

[105] E. Yoshida, "Humanoid motion planning using multi-level DoF exploitation based on randomized method," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2005, pp. 3378–3383.

[106] L. Zhang and D. Manocha, "An efficient retraction-based RRT planner," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2008, pp. 3743–3750.

[107] L. Zhang, Y. J. Kim, and D. Manocha, "A simple path non-existence algorithm using c-obstacle query," *in Algorithmic Foundation of Robotics VII*. Berlin, Germany: Springer, 2008, pp. 269–284.

[108] L. Zhang, J. Pan, and D. Manocha, "Motion planning of human-like robots using constrained coordination," in *Proc. IEEE Int. Conf. Humanoid Robots*, 2009, pp. 188–195.