Numerical Subdivision Methods in Motion Planning

Yi-Jen Chiang and Chee Yap

***Abstract*— We propose to design new algorithms for motion planning problems based on the Domain Subdivision paradigm, but coupled with numerical primitives. Although weaker than exact algebraic primitives, our primitives are safe and are exact in the limit. Our algorithms are practical, easy to implement, and have adaptive complexity. A simple but useful example of our approach is presented here. In contrast to the popular PRM, our algorithms are resolution complete.**

I. INTRODUCTION

A central problem of robotics is motion planning [5]. In the early 80's there was strong interest in this problem among computational geometers [3]. This period saw the introduction of strong algorithmic techniques with complexity analysis, and the careful investigation of the algebraic C-space. We introduced the retraction method [7], [11] into motion planning. In a survey of algorithmic motion planning [12], we first established the universality of the retraction method. This method is now commonly known as the road map approach, popularized by Canny [1] who showed that its algebraic complexity is in single exponential time. Typical of Computational Geometry, these exact motion planning algorithms assume a computational model in which exact primitives are available in constant time. Implementing these primitives exactly is non-trivial (certainly not constant time), involving computation with algebraic numbers. In the 90's, interest shifted back to more practical techniques, such as the probabilistic roadmap method (PRM) [4] and its many variants [5, Chapter 5].

In this paper, we propose new algorithms based on the classic subdivision paradigm, combined with numerical primitives. Probabilistic forms of our approach can serve as an alternative to PRM. But even the deterministic form offer advantages over PRM. Our solutions are practical as well as theoretically sound. The basic paradigm is to iteratively subdivide an initial configuration domain $B_0 \subseteq \mathbb{R}^d$ (given as a box) into subdomains. This process grows a subdivision tree rooted at B_0, by expanding carefully chosen leaves. In 2-D Euclidean space, such trees are known as quadtrees, as illustrated in Figure 1(b). Examples of our approach may be found in related recent work (e.g., [8], [10], [6], [14]).

II. SUBDIVISION MOTION PLANNING

In this section, we illustrate our approach with a basic motion planning problem. Fix a rigid robot $R_0 \subseteq \mathbb{R}^d$ and

Yap is supported by NSF Grant CCF-0917093. Chiang is supported by DOE Grant DE-SC0004874. Presented at IROS 2011 Workshop on Progress and Open Problems in Motion Planning, San Francisco, Sep 20, 2011. Chiang is with the Department of Computer Science and Engineering, Polytechnic Institute of NYU, yjc@poly.edu. Yap is with the Department of Computer Science, NYU, New York, NY 10012, USA, yap@cs.nyu.edu.

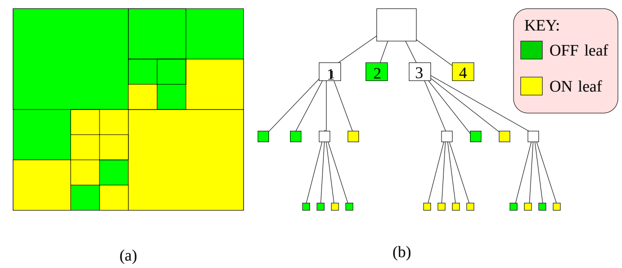

Fig. 1. (a) Subdivision of a region (yellow). (b) Its Subdivision Tree

an obstacle set $\Omega \subseteq \mathbb{R}^d$. Both R_0 and Ω are closed sets. Initially assume R_0 is a d-dimensional ball of radius $r_0 > 0$. In this case, the C-space of R_0 is $\mathbb{R}^d$. If α is a configuration, let the **placement** of R_0 at α be the set $R_0[\alpha]$ comprising those points in $\mathbb{R}^d$ occupied by R_0 in configuration α. A configuration α is **free** if $R_0[\alpha] \cap \Omega$ is empty; α is **blocked** if $R_0[\alpha]$ intersects the interior of Ω; α is **semi-free** if it is neither free nor blocked. Let $Free(R_0, \Omega)$ denote the set of free configurations. A **motion** from α to β is a continuous map $\mu : [0,1] \to Free(R_0, \Omega)$ with $\mu(0) = \alpha$ and $\mu(1) = \beta$.

Consider the problem of computing a motion from α to β. The best exact solution is based on roadmaps (i.e., retraction approach). Historically, the case $d = 2$ was the first exact roadmap algorithm [7]. For polygonal Ω, the roadmap is efficiently computed as the Voronoi diagram of line segments [13]. This algorithm remains very useful in applications that allows pre-computation as in games. For $d = 3$, an exact solution is not practical: the exact Voronoi diagram of polyhedral objects is a highly non-trivial current topic of research (e.g., [2]).

In our subdivision approach, the main data structure is a subdivision tree (see Figure 1). If $\mathcal{T}$ is a subdivision tree rooted at a box B_0, then its set of leaves is a collection of subboxes that forms a **subdivision** of B_0, i.e., the interiors of any two subboxes are disjoint, and their union is B_0. Let $Split(B)$ denote the unique subdivision of B comprising 2^d congruent subboxes. Boxes are considered as closed sets of full dimension d. Two boxes B, B' are **adjacent** if $B \cap B'$ is a face F of B or of B'. The dimension of F is exactly 1 less than that of B. Given any point $\alpha \in B_0$, let $Box_{\mathcal{T}}(\alpha)$ denote any leaf box of $\mathcal{T}$ that contains α. A box B is classified as (i) **free** if every configuration in B is free, i.e., $B \subseteq Free(R_0, \Omega)$; (ii) **blocked** if every $\alpha \in B$ is blocked; and (iii) **mixed** otherwise. Note that a mixed B can contain free, blocked or semi-free configurations. Moreover, if B degenerates into a single configuration γ, then γ is mixed (as a box) iff γ is semi-free (as a configuration).

Initially, assume a "box predicate" C to perform this classification: for any box B, $C(B)$ returns the desired value in $\{\mathtt{FREE}, \mathtt{BLOCKED}, \mathtt{MIXED}\}$. Given a subdivision tree $\mathcal{T}$, let $V(\mathcal{T})$ denote the set of free leaves in $\mathcal{T}$. We define an undirected graph $G(\mathcal{T})$ with vertices $V(\mathcal{T})$ and edges connecting pairs of adjacent free leaves. We maintain the connected components of $G(\mathcal{T})$ using a **Union-Find** data structure on $V(\mathcal{T})$: given $B, B' \in V(\mathcal{T})$, $Find(B)$ returns the index of the component containing B, and $Union(B, B')$ merges the components of B and of B'.

We associate with $\mathcal{T}$ a priority queue $Q = Q_{\mathcal{T}}$ to store all the mixed leaves. Let $\mathcal{T}.getNext()$ remove a box in Q of the highest "priority". This priority is discussed below. Assume a subroutine to "expand" any box $B \in Q$ as follows: the expansion fails and returns false if the size of B is smaller than a specified tolerance $\epsilon > 0$. Otherwise, each $B' \in Split(B)$ is made a child of B in $\mathcal{T}$. If B' is free, we update $V(\mathcal{T})$ and its union-find structure; if B' is mixed, we insert B' into Q. Finally we return true. Now we are ready to present a simple but useful exact subdivision algorithm:

```
EXACT FINDPATH:
Input: Configurations α, β, tolerance ε > 0, box B_0 ∈ ℝ^d.
Output: Path from α to β in Free(R_0, Ω) ∩ B_0.
      Initialize a subdivision tree T with only a root B_0.
1.    While (Box_T(α) ≠ FREE)
          If (Expand Box_T(α) fails) Return("No Path").
2.    While (Box_T(β) ≠ FREE)
          If (Expand Box_T(β) fails) Return("No Path").
3.    While (Find(Box_T(α)) ≠ Find(Box_T(β)))
          If Q_T is empty, Return("No Path")
(*)       B ← T.getNext()
          Expand B
4.    Compute a physical path P from Box_T(α) to Box_T(β).
      Return(P)
```

There are two comments to be made: First, the path P in Step 4 is easy to generate in our framework: this aspect is a major advantage over PRM and algebraic methods where physical. In PRM, physical paths are usually approximated by sampling free configurations between the endpoints of an edge, with no guarantees. In algebraic methods, it is assumed that another numerical process will produce the physical path from an algebraic description.

Second, the routine $\mathcal{T}.getNext()$ in Step (*) is not fully specified, but critical. In fact, it is the **strategy** that drives the search. A simple solution to ensure **resolution completeness** is the Breadth First Search (BFS) strategy, i.e., $\mathcal{T}.getNext()$ returns any mixed leaf of minimum depth. Resolution completeness has two parts: *If there is a free motion of clearance 2ϵ, our algorithm would find a free path of clearance $\geq \epsilon$. Conversely, if there is no free path of clearance $\epsilon/2$, our algorithm will return "No Path"*. We mention a few other interesting strategies. Most of these these strategies are not resolution complete by themselves, but we can make them resolution complete by mixing them with BFS. For instance, we can alternate between BFS and these strategies. Or, we can use a weighting function to combine their respective priorities.

To begin, we could use the **Randomized Strategy**, and this could be viewed as a form of PRM. But unlike the usual PRM, we have resolution completeness (assuming a mix with BFS). Another is the **Dijkstra strategy**: $getNext()$ returns a mixed box that is adjacent to some free box in the connected component of $Box_{\mathcal{T}}(\alpha)$, analogous to Dijkstra's shortest-path algorithm. This can be generalized to the A*, where we introduce a suitable potential function to bias the search towards the goal (the obvious potential is the direct distance between the center of a box to the goal). Even better is the bi-directional A* strategy. Another idea is to use some entropy criteria. Recent work on shortest-path algorithms in GIS road systems offers other heuristics. We plan to explore these.

III. WHAT IS NEW?

Subdivision algorithms have been used before in motion planning, e.g., [9]. So subdivision alone is not a novelty. The use of Union-Find is interesting since the operations are extremely fast, but this has been used [5]. Our true interest lies in relaxing the assumption of the exact predicate $C(B)$. All previous subdivision algorithms have assumed exact predicates, and this is a serious impediment to their usability. Let $\widetilde{C}(B)$ be a box predicate that returns a value in $\{\mathtt{FREE}, \mathtt{BLOCKED}, \mathtt{FAIL}\}$. We say that $\widetilde{C}$ **approximates** C if (1) it is **safe**, i.e., $\widetilde{C}(B) \neq \mathtt{FAIL}$ implies $\widetilde{C}(B) = C(B)$, (2) it is **convergent**, i.e., if $\{B_i : i = 1, 2, \ldots, \infty\}$ converges to a configuration γ and $C(\gamma) \neq \mathtt{MIXED}$, then $\widetilde{C}(B_i) = C(\gamma)$ for large enough i.

We now design an approximate box predicate $\widetilde{C}$ assuming Ω is a polyhedral set, and the boundary of Ω is partitioned into a simplicial complex comprising open cells of each dimension. These cells are called **features** of Ω. For $d = 3$, the features of dimensions $0, 1, 2$ (resp.) are called **corners**, **edges** and **walls**. Let $m(B)$ and $r(B)$ denote its midpoint and radius of box B respectively, where $r(B)$ is the distance from $m(B)$ to any corner of B. Also, let $D_m(r)$ denote the closed ball centered at m with radius r. We maintain with each box B the set $S(B)$ of features that intersect $D_{m(B)}(r_0 + r(B))$. We call B **simple** if either [S0] its set $S(B)$ of the maintained features is empty, or [S1] $r_0 > r(B)$ and some feature intersects the ball $D_{m(B)}(r_0 - r(B))$. We now define the approximate predicate $\widetilde{C}$: if B is non-simple, then $\widetilde{C}(B) = \mathtt{FAIL}$; if [S1] holds, then $\widetilde{C}(B) = \mathtt{BLOCKED}$; otherwise, [S0] holds and clearly B is either free or blocked. But how do we decide? In fact, $\widetilde{C}(B) = \mathtt{FREE}$ (resp., $\mathtt{BLOCKED}$) iff $D_{m(B)}(r_0 + r(B))$ is exterior (resp., interior) relative to the obstacle Ω. To distinguish these two cases, we just check the wall features maintained in the parent box $p(B)$ of B (noting that $S(p(B))$ is non-empty). To do this check, we may assume that each wall w is oriented so that we know (locally) which side of w is inside Ω. First, observe that $\widetilde{C}$ is designed to be extremely easy to implement, since all the tests boils down to one operation: the distance from a point to an obstacle feature. Second, $\widetilde{C}$ is an approximation of C. To complete our scheme, when $\widetilde{C}(B) = \mathtt{FAIL}$ (i.e., B is non-simple), we put B to Q for future expansion.

Conclusion. In the full paper, we explore variants of $\widetilde{C}$. Our general philosophy can be extended to more complicated C-spaces such as $SE(2)$ and $SE(3)$ and non-holonomic planning. Combined with suitable $\mathcal{T}.getNext()$ heuristics, the complexity of our algorithms can be highly adaptive. We plan to implement and compare our method with other approaches, including those with exact predicates and probabilistic approaches.

REFERENCES

[1] J. Canny. Computing roadmaps of general semi-algebraic sets. *The Computer Journal*, 36(5):504–514, 1993.

[2] H. Everett, D. Lazard, S. Lazard, and M. S. E. Din. The Voronoi diagram of three lines. *Discrete and Comp. Geom.*, 42(1):94–130, 2009. See also 23rd SoCG, 2007.

[3] D. Halperin, L. Kavraki, and J.-C. Latombe. Robotics. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 41, pages 755–778. CRC Press LLC, 1997.

[4] L. Kavraki, C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robotics and Automation*, 12(4):566–580, 1996.

[5] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, 2006.

[6] L. Lin and C. Yap. Adaptive isotopic approximation of nonsingular curves: the parameterizability and nonlocal isotopy approach. *Discrete and Comp. Geom.*, 45(4):760–795, 2011.

[7] C. Ó'Dúnlaing and C. K. Yap. A "retraction" method for planning the motion of a disc. *J. Algorithms*, 6:104–111, 1985. Also, Chapter 6 in *Planning, Geometry, and Complexity*, eds. Schwartz, Sharir and Hopcroft, Ablex Pub. Corp., Norwood, NJ. 1987.

[8] S. Plantinga and G. Vegter. Isotopic approximation of implicit curves and surfaces. In *Proc. Eurographics Symposium on Geometry Processing*, pages 245–254, New York, 2004. ACM Press.

[9] J. H. Reif and H. Wang. Nonuniform discretization for kinodynamic motion planning and its applications. *SIAM J. Computing*, 30:161–190, 2000.

[10] M. Sagraloff and C. K. Yap. A simple but exact and efficient algorithm for complex root isolation. In *36th Int'l Symp.Symbolic and Alge.Comp. (ISSAC)*, pages 353–360, 2011. June 8-11, San Jose, California.

[11] M. Sharir, C. O'D'únlaing, and C. Yap. Generalized Voronoi diagrams for moving a ladder II: efficient computation of the diagram. *Algorithmica*, 2:27–59, 1987. Also: NYU-Courant Institute, Robotics Lab., No. 33, Oct 1984.

[12] C. K. Yap. Algorithmic motion planning. In J. Schwartz and C. Yap, editors, *Advances in Robotics*, *Vol. 1: Algorithmic and geometric issues*, volume 1, pages 95–143. Lawrence Erlbaum Associates, 1987.

[13] C. K. Yap. An $O(n \log n)$ algorithm for the Voronoi diagram for a set of simple curve segments. *Discrete and Comp. Geom.*, 2:365–394, 1987. Also: NYU-Courant Institute, Robotics Lab., No. 43, May 1985.

[14] C. K. Yap. In praise of numerical computation. In S. Albers, H. Alt, and S. Näher, editors, *Efficient Algorithms*, volume 5760 of *Lecture Notes in Computer Science*, pages 308–407. Springer-Verlag, 2009.