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 complete 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.**

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].

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. 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 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. Call α a **free** configuration if $R_0[\alpha] \cap \Omega$ is empty. Let $Free(R_0, \Omega)$ denote the set of free configurations. A **motion** from α to β is a continuous map $\mu : [0,1] \rightarrow Free(R_0, \Omega)$ with $\mu(0) = \alpha$ and $\mu(1) = \beta$.

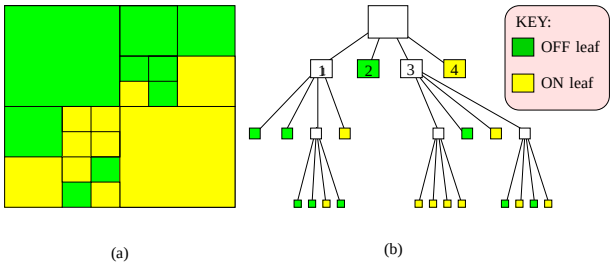

Fig. 1. (a) Subdivision of a region (yellow). (b) Its Subdivision Tree

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]. 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 $B \subseteq Free(R_0, \Omega)$, (ii) **blocked** if $B \cap Free(R_0, \Omega) = \emptyset$, and (iii) **mixed** otherwise. Initially, assume a "box predicate" C to perform this classification: for any box B, $C(B)$ returns the desired value in $\{\texttt{FREE}, \texttt{BLOCKED}, \texttt{MIXED}\}$. 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'.

This work is supported by NSF Grant CCF-0917093 and DOE Grant DE-SC0004874. 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.

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 $\epsilon > 0$, box $B_0 \in \mathbb{R}^d$.
Output: Path from α to β in $Free(R_0, \Omega) \cap B_0$.
Initialize a subdivision tree $\mathcal{T}$ with only a root B_0.
1. While ($Box_{\mathcal{T}}(\alpha) \neq$ FREE)
   If (Expand $Box_{\mathcal{T}}(\alpha)$ fails) Return("No ϵ-Path").
2. While ($Box_{\mathcal{T}}(\beta) \neq$ FREE)
   If (Expand $Box_{\mathcal{T}}(\beta)$ fails) Return("No ϵ-Path").
3. While ($Find(Box_{\mathcal{T}}(\alpha)) \neq Find(Box_{\mathcal{T}}(\beta))$)
   If $Q_{\mathcal{T}}$ is empty, Return("No ϵ-Path")
(*) $B \leftarrow \mathcal{T}.getNext()$
   Expand B
4. Compute a path P from $Box_{\mathcal{T}}(\alpha)$ to $Box_{\mathcal{T}}(\beta)$.
Return(P)

The path P in Step 4 is easy to generate in this framework (as noted, this aspect is a big win over pure algebraic methods). The routine $\mathcal{T}.getNext()$ in Step (*) is not fully specified, but critical. To ensure completeness, a simple solution is to return any mixed leaf of minimum depth. Completeness means that *if there is a free motion, our algorithm would find a free path if ϵ is small enough.* But there are many other interesting heuristics for $\mathcal{T}.getNext()$: If $getNext()$ is random, we could view this as *a form of the probabilistic roadmap method*. If $getNext()$ always returns a mixed box that is adjacent to a free box in the connected component of $Box_{\mathcal{T}}(\alpha)$, we get a form of Dijkstra's shortest-path algorithm. Another idea is to use some entropy criteria. Recent work on shortest-path algorithms in GIS road systems offers many other heuristics. We will explore all these.

III. WHAT IS NEW?

Subdivision algorithms in motion planning were employed in, e.g., [9]. So subdivision alone is not a novelty. Our use of Union-Find is quite interesting since the operations are extremely fast. But our true interest lies in relaxing the assumption of an exact predicate $C(B)$. Let $\widetilde{C}(B)$ be a box predicate that returns a value in $\{$FREE, BLOCKED, FAIL$\}$. We say that $\widetilde{C}$ **approximates** C if (1) it is **safe**, i.e., $\widetilde{C}(B) \neq$ 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 $C(\gamma) \neq$ 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 $\widetilde{C}(B) =$ FAIL; if [S1] holds, then $\widetilde{C}(B) =$ BLOCKED; otherwise, [S0] holds and clearly B is either free or blocked. But how do we decide? In fact, $\widetilde{C}(B) =$ FREE (resp., 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. We see that (1) $\widetilde{C}$ is extremely easy to implement and (2) it is an approximation of C. To complete our scheme, when $\widetilde{C}(B) =$ 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, eds., *Handbook of Discrete and Computational Geometry*, chapter 41, pp. 755–778. CRC Press LLC, 1997.
[4] L. Kavraki, P. Švestka, J.-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.
[8] S. Plantinga and G. Vegter. Isotopic approximation of implicit curves and surfaces. In *Proc. Eurographics Symposium on Geometry Processing*, pages 245–254, 2004.
[9] J. H. Reif and H. Wang. Nonuniform discretization for kinodynamic motion planning and its applications. *SIComp.*, 30:161–190, 2000.
[10] M. Sagraloff and C. K. Yap. A simple but exact and efficient algorithm for complex root isolation. In *36th ISSAC*, pages 353–360, 2011.
[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.
[12] C. K. Yap. Algorithmic motion planning. In J. Schwartz and C. Yap, eds., *Advances in Robotics, Vol. 1: Algorithmic and geometric issues*, vol. 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. *Disc. Comp. Geom.*, 2:365–394, 1987.
[14] C. K. Yap. In praise of numerical computation. In S. Albers, H. Alt, and S. Näher, eds., *Efficient Algorithms*, vol. 5760, *Lecture Notes in Computer Science*, pp. 308–407, 2009.