Chapter 1
Theory of Soft Subdivision Search and Motion Planning

Chee K. Yap*
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
New York, NY 10012, USA
Email: yap@cs.nyu.edu

Abstract

We propose to design motion planning algorithms using two ingredients: the subdivision paradigm coupled with **soft predicates**. Such predicates are conservative and convergent relative to traditional exact predicates (called "hard" in this context). This leads to **resolution-exact** algorithms which can be viewed as a strong form of "resolution complete" algorithms. Resolution-exactness contains inherent indeterminacies and other subtleties. We describe an algorithmic framework, called **Soft Subdivision Search** (SSS) for designing such algorithms. There are many parallels between our framework and the well-known Probabilistic Road Maps (PRM) framework. Both frameworks lead to algorithms that are highly practical, easy to implement, have adaptive and local complexity. The critical difference is that SSS avoids the Halting Problem of PRM.

In a previous paper, we have demonstrated the ease of designing soft predicates for various motion planning problems. In this paper, we generalize and extend some of these results. We show how exact algorithms can be recovered by an extension of our framework. The SSS framework provides a theoretically sound basis for new classes of algorithms in motion planning and beyond. Such algorithms are novel, even in the exact case.

1.1 Introduction

Motion Planning is a fundamental problem in robotics. One of its origins is the "findpath problem" in Artificial Intelligence [6, 5]. In the 1980s, computational geometers began the algorithmic study of motion planning [31, 13], focusing on *exact planners*: assuming the input is exact, such planners return a path if any exists, and report "No Path" otherwise. Schwartz and Sharir [30] observed that the cell decomposition approach is a universal approach for motion planning, and in the algebraic case, is effectively reducible to Collin's cylindrical algebraic decomposition. We introduced the concept of retraction motion planning in [27, 26]. In the first survey

* This work is supported by NSF Grant CCF-0917093.

on algorithmic robotics [31], we observed that the retraction approach is also universal (again, this is effective in the algebraic case). After the work of Canny [7], the retraction approach became popularly known as the "roadmap approach". In the 1990's the roadmap approach takes another turn.

¶1. Theory. Today, exact motion planning continues to be actively investigated (e.g., [12]). A fairly up-to-date account from the perspective of real algebraic geometry may be found in [3]. Some of these algorithms represent major theoretical advancements. Nevertheless their impact on practical robotics is quite modest: thus [36] noted that exact implementations have been limited to 3 degrees of freedom, and for simple robots only. Various sub-algorithms and supporting data structures needed in exact motion planning have been implemented in CGAL [8]. For example, the recent exact algorithm for the Voronoi diagram of lines in space is regarded as a significant advance [15]; but true goal here is Voronoi diagram of polyhedral objects. Exactness has tremendous cost in terms of computational complexity: it implicitly requires algebraic numbers. Direct manipulation of algebraic numbers is impractical. But for many basic problems, a weaker form under the paradigm of Exact Geometric Computation (EGC) is sufficient [32]. Nevertheless, the usual expedient is to replace exact arithmetic by machine arithmetic, leading to the ubiquitous problems of numerical non-robustness [33]. Even ignoring efficiency issues, there is a fundamental but less well-known barrier: *the Turing computability of exact algorithms for most non-algebraic problems is unknown* [34]. This barrier exists in most problems beyond kinematic motion planning. But see [9] for a rare case of a non-algebraic motion planning problem that is provably computable; this positive result is possible thanks to deep results in transcendental number theory.

¶2. Practice. Since the mid 1990's, the method of Probabilistic Road Maps (PRM) has become dominant among roboticists. Its basic formulation comes from Kavraki, Švestka, Latombe and Overmars [19]. PRM is not a particular algorithm but a **algorithmic framework** for motion planners. Many variants of this framework are known: Expansive-Space Tree planner (EST), Rapidly-exploring Random Tree planner (RRT), Sampling-Based Roadmap of Trees planner (SRT), and many more. Quoting Choset et al [11, p.201]: "*PRM, EST, RRT, SRT, and their variants have changed the way path planning is performed for high-dimensional robots. They have also paved the way for the development of planners for problems beyond basic path planning.*"

In his invited talk at the recent workshop[2] on open problems in this field, J.C. Latombe stated that the major open problem of PRM is that it does not know how to terminate when there is no path. In practice, one simply times-out the algorithm, but this leads to problems such as the "Climbers Dilemma" [14, p. 4] described in the work of Bretl (2005). We call this the **Halting Problem of PRM**. This is a known issue for researchers, and is the extreme form of the so-called "Narrow Passage Problem" [11, p. 201]. Latombe's talk suggested promising approaches

[2] IROS 2011 Workshop on Progress and Open Problems in Motion Planning, September 30, 2011, San Francisco.

such as Lazy PRM [4]; other lines of attack include explicit detection of the non-existence of paths [2]. The theoretical basis for PRM algorithms is that they are probabilistic complete [18]. The Halting Problem is inherent in probabilistic completeness.

¶3. Common Ground. We seek a common ground that provides stronger guarantees than probabilistic completeness, but avoids the inordinate demands of exactness. Fortunately for our subject, exactness is a mismatch for the needs of robotics. This is clear from the remark that physical devices and sensors have limited accuracy. Practitioners are acutely aware of this. Yet it does not absolve us from mathematical precision if we wish the theoretical development of robotic algorithms to thrive. This tension between the needs of practice and of theory has led to their divergent paths described above. So we turn to the idea of "resolution complete" algorithms, noting that the 1983 paper of Brooks and Lozano-Perez [6] was already on this track. It is known that resolution complete algorithms can avoid the Halting Problem (e.g., [36]). Unfortunately the notion of resolution completeness is seldom scrutinized, and is capable of many interpretations. In [10] we pointed out some untenable, or lacking, interpretations. As remedy, we introduced a version called **resolution-exactness**, and proved basic properties of such algorithms. Surprisingly, we show that resolution-exactness has an inherent indeterminacy, even for deterministic algorithms using exact predicates. But the indeterminacy is mild in comparison to that of probabilistic completeness. Unlike the determinacy of exact algorithms, this indeterminacy seems a perfect match for the requirements of robotics.

There are two ingredients of resolution-exact algorithms. The first is subdivision of configuration space. We organize the subdivision into a **subdivision tree**. In 2 and 3 dimensions, such trees are usually called quadtrees and octrees. Tree nodes correspond to subsets of configuration space with simple shapes such as boxes or simplices. The notion of grid search is often identified with resolution complete algorithms (e.g., see the Wikipedia entry on Motion Planning). Although grids are superficially similar to subdivision, we stress that typical grid-based methods are inadequate for resolution-exactness. The second ingredient is a **classification predicate** to decide if a node is free or not. Such predicates could be computed exactly in the algebraic case; that would be the reflex viewpoint of a computational geometer, but it is not where we want to be. Our key insight is this: *in the presence of subdivision, exact predicates can be replaced by suitable approximations*. We came by this viewpoint through a series of related work on subdivision algorithms (e.g., [28, 23, 29]). Such approximations are formalized as **soft predicates** in [10]. There we show through a series of motion planning examples, the relative ease of designing soft predicates, and claimed that they are practical.

Let us address this claim. Since the implementation of our algorithms is currently underway, our evidence for practicality is indirect: first, our subdivision infrastructure is based on well-understood and practical data structures (subdivision tree, union-find, etc). Next, the soft predicates we designed to go with subdivision are mostly reduced to estimating distances between two features, where a feature is a point, line segment or a triangle in space. Moreover, these predicates can be easily

and correctly implemented (see ¶12 below). Thus there are no implementation gaps for our algorithms. The argument so far centers on *implementability*. But how can we be sure that these implementations will be *practically efficient*? Here, we invoke the evidence of prior resolution-based work. We mention key papers such as Zhu and Latombe (1991) [37], Barhehenn and Hutchinson (1995) [1], and Zhang, Kim and Manocha (2008) [36]. Of course, since these work preceded our formulation, we must reinterpret their methods using our new perspective. In fact, it is illuminating and fruitful to revisit these papers from our current perspective. In short, through the implementability and practical efficiency of resolution-exact algorithms, we may have found a common ground for theory and practice.

¶4. Our Goals. The current paper aims to clearly expose the foundations of resolution exactness. There are three themes: (1) We first take a leaf from the success of PRM research: the simplicity and generality of PRM framework ensures that implementers of this framework can get easy access to a whole family of algorithms, just by modifying one or more components in the framework. This leads us to formulate an analogous framework for our approach, called **soft subdivision search** (SSS). (2) Next, we generalize the setting of our previous results [10]: for instance, the basic setting of a free space embedded in configuration space, $C_{free} \subseteq C_{space}$, can be replaced by an open subset Y of a normed linear space X. The boxes used in our subdivision trees can be replaced by other shapes such as simplices. (3) Finally, we want to revisit exact algorithms from a subdivision viewpoint: each SSS algorithm takes an input resolution parameter $\varepsilon > 0$, in addition to the normal inputs of path planning. It is essential that ε is positive. But if we admit $\varepsilon = 0$, the resolution-exact algorithm may become non-halting like PRM. We show how to to fix this problem. Interestingly, such exact algorithms are novel and seems more implementable than usual exact algorithms.

All proofs are given in a separate Appendix.

¶5. Preliminaries. We establish some notations for standard concepts. To focus on the key ideas, this paper will assume the simplest formulation of the motion planning problem: point-to-point kinematic motion planning for any particular robot R_0 moving in a physical space $\mathbb{R}^k$ ($k = 2, 3$) amidst a static obstacle $\Omega \subseteq \mathbb{R}^k$. The configuration space $C_{space} = C_{space}(R_0)$ will be appropriately embedded in $\mathbb{R}^d$ ($d \geq k$) (see [21, p.128] for discussions of embedding issues). The **footprint map** is $Fp : C_{space} \to 2^{\mathbb{R}^k}$ where $Fp(\gamma) \subseteq \mathbb{R}^k$ is the physical space occupied by robot R_0 in configuration γ. E.g., for a rigid robot, $Fp(\gamma)$ is a rotated, translated copy of R_0. Then a configuration $\gamma \in C_{space}$ is **free** iff $Fp(\gamma) \cap \Omega = \emptyset$. The set of free configurations $C_{free} = C_{free}(R_0, \Omega)$ is an open subset of C_{space}, assuming Ω is a closed set. But central to our theory is the boundary $\partial(C_{free})$ of C_{free}. Configurations in $\partial(C_{free})$ are said to be **semi-free**. A **motion** is a continuous function $\mu : [0, 1] \to C_{space}$, and its range $\mu[0, 1]$ is called the **trace**. The motion is **free** if its trace is contained in C_{free}. A **path** refers to a free motion. So the basic motion planning problem for robot R_0 is this: given start α and goal β configurations, and Ω (defining C_{free}), find a path from α to β if one exists, and report

"No Path" otherwise. A key tool is the **clearance function**, $C\ell : C_{space} \to \mathbb{R}_{\geq 0}$ where $C\ell(\gamma)$ is the separation of the footprint at γ from Ω, $C\ell(\gamma) := \text{sep}(Fp(\gamma), \Omega)$ where $\text{sep}(A, B) = \inf\{\|a - b\| : a \in A, b \in B\}$ is the **separation** between two sets $A, B \subseteq \mathbb{R}^k$. Thus $C\ell(\gamma) > 0$ iff $\gamma \in C_{free}$. The **clearance** of a motion μ is the minimum $C\ell(\mu(t))$ for $t \in [0, 1]$.

1.2 Two Frameworks for Motion Planning.

In this paper we use the terminology of **algorithmic framework** to discuss broad classes of algorithms, and view PRM as such a framework. An algorithm within the framework is[3] just a specific instantiation, using particular data structures, strategies and subroutines. We will give a formulation of the PRM framework and our SSS framework, and compare them.

¶6. The PRM Framework. Here is a formulation of PRM, following LaValle [21, Section 5.4.1]: the goal is to find a path connecting $\alpha, \beta \in C_{space}$. We maintain a graph $G = (V, E)$ where $V \subseteq C_{free}$ and edges in E correspond to paths connecting the vertices of the edge. We may assume that α, β are in V. We need two predicates, $Free(u)$ to test if a configuration u is free, and $Connect(v, u)$ to test if the (straight) motion from v to u is free. Finally, assume some **termination criterion** that is comprised of two parts: success-criterion (found a path from α to β) and a failure-criterion (time-out or other condition).

PRM FRAMEWORK:
While (termination criterion fails):
1. Vertex Selection Method (VSM):
   Choose a vertex v in V for expansion.
2. Configuration Generation Method (CGM):
   Generate some $u \in C_{space}$ (perhaps near v)
3. Local Planning Method (LPM):
   If $Free(u)$,
   Add u to V
   If $Connect(v, u)$, add (v, u) to E.

Return success or failure accordingly.

Step 1 (VSM) is usually controlled by some priority queue representing the search strategy. Step 2 (CGM) is the probabilistic step. But CGM could also be deterministic, e.g., controlled by a dense sampling sequence [20]. LaValle would call this the "Sampling Framework" to avoid any prior commitment to randomness. But we say "PRM Framework" for specificity, and in honor of the most well-known formulation of such approaches. In Step 3 (LPM), u is discarded if it is not free; another

[3] To be sure, there are degrees of specificity. The most specific instantiation might be called "implementation" of some less specific "algorithm".

method is to generate a free configuration u' such that the subpath, from v to u', of the direct v to u path is free. As noted in [11, p.198], the practical success of PRM stems from the fact that the predicate $Free(u)$ is relatively cheap. There is a large literature on computing this predicate, under the heading of **collision detection**. Indeed, the $Connect(v, u)$ predicate is often reduced to $Free(u)$: if a "sufficiently dense" sampling of configurations from v to u is free, just assume there is a path from v to u.

By varying this simple framework, we could capture most of the known variations mentioned earlier. The original PRM is framed in terms of a road-map stage followed by a query stage; so the above version is closer to the "BasicPRM" of [18]. But our discussion of the "PRM Framework" is intended to cover such variations.

¶7. What confers power to PRM? The practical advantages of PRM is widely recognized, and it is natural to assume that randomness is the source of this power. LaValle et al [20] examine this question and concluded that sampling rather than randomness is the true source of power. Hsu et al [18] argue for the essential role of randomness. Independent of this debate, we offer another reason for the success of PRM: the PRM framework allows one to easily modify the constituent components (sampling strategy, connection strategy, freeness predicate, etc) to obtain a variety of algorithms that meet diverse needs. The basic infrastructure is kept relatively stable. This is possible thanks to the *simplicity* and *generality* of the PRM framework. Just as important in practice, the framework is also very *forgiving*: you could implement the constituent components approximately or even wrongly, and the software implementation[4] may not necessarily fail (crash or loop). These properties are in sharp contrast to the usual exact algorithms which are far from simple and not too forgiving of errors [32]. In recognition of this, we would like to propose an analogous framework for the subdivision approach.

¶8. The SSS Framework. For a fixed robot R_0, the motion planning input is an initial box $B_0 \subseteq C_{space}$, the obstacle $\Omega \subseteq \mathbb{R}^k$, the start and goal configurations $\alpha, \beta \in C_{space}$, and a **resolution parameter** $\varepsilon > 0$. We are interested in "resolution-restricted" search for a path from α to β *inside* B_0. As noted in the introduction, our main data structure is a subdivision tree, $\mathcal{T}$. The root is B_0 and each tree node is a subbox of B_0. The algorithm amounts to a while-loop that "grows" $\mathcal{T}$ in each iteration by expanding some leaf until we find a path or conclude "No Path". Here are the supporting subroutines and data structures: assume a **predicate** $\widetilde{C}$ that classifies each node in $\mathcal{T}$ into `FREE`/`STUCK`/`MIXED`, with the property that $\widetilde{C}(B) = \texttt{FREE}$ implies $B \subseteq C_{free}$ and $\widetilde{C}(B) = \texttt{STUCK}$ implies $B \cap \overline{C_{free}} = \emptyset$. We maintain a priority queue $Q = Q_{\mathcal{T}}$ comprising those `MIXED`-leaves whose length $\ell(B)$ (defined below) is at least ε. Let $Q.\mathbf{GetNext}()$ return a leaf B of highest priority. This B is given to another subroutine $\mathbf{Expand}(\texttt{B})$ which subdivides B into two or more subboxes. These subboxes become the children of B (so B is no longer a leaf). For now, assume $\texttt{Expand}(B)$ always split B into 2^d congruent subboxes. After splitting,

[4] The hardware implementation, however, might have catastrophic consequences. But here, we rely on the fact that most robot systems are fail-safe.

$\widetilde{C}$ is immediately called to classify these subboxes. Boxes that are FREE need further processing: assume a union-find data structure D to maintain the connected components of the FREE leaves of $\mathcal{T}$. Say two boxes B, B' are **adjacent** if $B \cap B'$ is a $d-1$ dimensional set. This defines a graph whose vertices are the FREE boxes, and edges representing their adjacency relation. D maintains the connected components of this graph. We first insert each new FREE leaf B into D, and call $Union(B, B')$ for any FREE B' that is adjacent to B. Assume $Find(B)$ returns the connected component of B, and write "$Box(\alpha)$" to denote any leaf of $\mathcal{T}$ that contains $\alpha \in C_{space}$.

SSS FRAMEWORK
1. ▷ *Initialization.*
   While ($\widetilde{C}(Box(\alpha)) \neq$ FREE)
      If $Box(\alpha)$ has length $< \varepsilon$, Return ("No Path")
      Else Expand($Box(\alpha)$)
   While ($\widetilde{C}(Box(\beta)) \neq$ FREE)
      ... do the same for β ...
2. ▷ *Main Loop:*
   While ($Find(Box(\alpha)) \neq Find(Box(\beta))$)
      If Q is empty, Return("No Path")
      $B \leftarrow Q$.GetNext()
      Expand(B)
3. Compute a FREE channel P from $Box(\alpha)$ to $Box(\beta)$
      Generate a path $\overline{P}$ from P and Return($\overline{P}$)

Resolution approaches can be wasteful when it is non-adaptive. In SSS, the resolution increases is naturally adaptive (we only expand at mixed cells). The resolution literature sometimes claimed incorrectly that the size of $\mathcal{T}$ is (must be) exponential in the depth. A counter example is [29] where we prove that tree size is only polynomial in the depth for certain subdivision algorithms for root isolation. Our formulation can recapture the approach of Zhu and Latombe [37], Barbehenn and Hutchinson [1], or Zhang, Kim and Manocha (2008) [36] as follows: these papers expand along a "mixed channels" (i.e., path comprising FREE or MIXED boxes). We could define GetNext to expand similarly. It turns out (see [10]) that our computation of $\widetilde{C}$ could exploit the subdivision tree $\mathcal{T}$. LaValle observed this curious property of our method, calling it "opening up the blackbox" of collision testing.

¶9. Similarities. There are many similarities between PRM and SSS, especially in their contrasts with exact algorithms.
1. Both have two key subroutines, representing (i) the global search strategies and (ii) free-ness testing. In PRM, the two subroutines are the vertex selection method (VSM), and the $Free(u)$ predicate, respectively. In SSS, they are GetNext(), and the predicate $\widetilde{C}(B)$.
2. An advantage of SSS and PRM is the possibility of finding paths *before* the entire C_{space} has been fully explored. Indeed, Hsu, Latombe and Kurniawati [18, p. 640] remarked that "foundational choice made in PRM planning is to avoid computing the exact shape of the free space". Most exact methods require an expensive a preprocessing phase to compute a full description of free space.

3. Integrated path planning: both frameworks naturally compute a path, i.e., a parametrized curve in C_{free}. E.g., Step 3 of SSS converts a channel of free boxes into a path. But exact algorithms often focus on computing a symbolic path in some algebraic cell complex, assuming that some numerical subroutine will convert it into a path.
4. We have viewed PRM as a probabilistic framework, and SSS as a deterministic one. But both frameworks admit deterministic or probabilistic algorithms. In the future, we plan to explore the probabilistic side of SSS.

¶10. Differences.
I. Foremost, SSS algorithms do not suffer from a Halting Problem.
II. PRM needs the predicate $Connect(v, u)$ to connect two nodes. The analogue in SSS simply amounts to checking if two `FREE` boxes are adjacent.
III. The search strategy in PRM resides in the Vertex Selection Method (VSM) and Configuration Generation Method (CGM). In SSS, it resides in `GetNext`() and `Expand`(). Sampling strategies is a major research question in PRM [20, 18]. Sampling in SSS seems to be more easily controlled, thanks to the nature of subdivisions. For example, a trivial randomized strategy in SSS is to pick any `MIXED` leaf with equal probability. Two deterministic SSS strategies are breadth-first search (BFS) and A-star/Dijkstra search [1]. We can have **hybrid strategies**: given two or more strategies, we just cycle through each one in turn. If one of them is randomized, then our hybrid will also gain any advantage of randomness.

1.3 Soft Classifiers

The $Connect(v, u)$ predicate is often implemented heuristically. LaValle [21, p. 177] discussed certified methods for this test based on Lipschitz constants. Such certified tests is a generalization of the $Free(u)$ predicate for a single configuration. We now consider a different generalization based on sets; it is basically the viewpoint of interval arithmetic [28].

We first generalize the setup in the Preliminary (¶5). Suppose X is a normed linear space with norm $\|\cdot\|$ (e.g., $X = \mathbb{R}^d$). Fix a subset $\square X$ of the powerset 2^X (e.g., $\square X$ is the set of boxes in X). Call $\square X$ a **test domain** if it has these properties:

- Each $B \in \square X$ is a full-dimensional closed bounded polytope in X. We call B a **test cell** (or simply "cell"). We define an interior point $c(B)$ called its **center**.
- $\square X$ is closed under translation and dilation: if B is a cell, then so is $t + \sigma \cdot B$ for any $t \in X, \sigma > 0$. Here, $t + B$ denotes the translation of B by t, and $\sigma \cdot B$ denotes the dilation of B by ratio σ at the center $c(B)$.

Note that for $X = \mathbb{R}^d$, if $d = 1$, then cells are just closed intervals with distinct endpoints. For $d > 1$, we have many more possibilities.

By a **classifier** we mean any function $C : \square X \to \{\texttt{IN}, \texttt{ON}, \texttt{OUT}\}$. So a classifier is a special kind of predicate that "classifies" every test cell. These values[5] correspond

[5] These values may also be called `EMPTY`/`MIXED`/`FULL`, as in original Brooks-Perez paper. They reflect the 3-valued nature of geometric predicates (as opposed to 2-valued logical predicates).

(respectively) to FREE/MIXED/STUCK of the previous section. Let $Y \subseteq X$ be any open subset of X. Call C a Y**-classifier** if for all $B \in \square X$,

$$\begin{cases} C(B) = \texttt{IN} & \Longrightarrow B \subseteq Y \\ C(B) = \texttt{OUT} & \Longrightarrow B \cap \overline{Y} = \emptyset \end{cases} \tag{1.1}$$

where $\overline{Y}$ denote the closure of Y. Thus a trivial classifier is one that is identically ON, $C(B) \equiv \texttt{ON}$. If the two implications of (1.1) are replaced by logical equivalences ("if and only if" conditions) then we call C an **exact** Y**-classifier**, denoted by C_Y. Note that singletons $p \in X$ are not test cell, and so $C(p)$ is not defined. Nevertheless, the exact Y-classifier has a unique extension to points where $C_Y(p) = \texttt{IN}$ if $p \in Y$, $= \texttt{ON}$ if $p \in \partial Y$, and $= \texttt{OUT}$ otherwise. This extension is justified as follows: write "$\lim_i B_i \to p$" to indicate an infinite decreasing sequence $B_1 \subseteq B_2 \subseteq \cdots$ that converges to $p \in X$. It is easy to see that $\lim_i B_i \to p$ implies that the sequence $C_Y(B_1), C_Y(B_2), \ldots$ eventually stabilizes to the value $C_Y(p)$. We denote this by writing "$\lim_i C_Y(B_i) \to C_Y(p)$". We can now define our key concept: a Y-classifier $\widetilde{C}$ is said to be **soft** if

$$\lim_i B_i \to p \quad \Longrightarrow \quad \lim_i \widetilde{C}(B_i) \to C_Y(p).$$

Thus, a soft predicate converges to the exact (or "hard") predicate in the limit.

¶11. How to compute soft classifiers? Two standard ideas of resolution-based methods are (a) splitting cells to reduce complexity, and (b) using numerical approximation. Typically, (a) is determined by an arbitrary resolution parameter but [10] demonstrated the use of inherent adaptive splitting criteria. Here we focus on (b). Let $C\ell : X \to \mathbb{R}$ be a continuous function. Call $C\ell$ a **(generalized) clearance function** of the set $\{x \in X : C\ell(x) > 0\}$ (it is generalized because $C\ell$ can be negative). Because of the splits in (a), $C\ell$ need only be defined "locally". Recall $\square\mathbb{R}$ is the set of closed intervals; consider an **interval function** $\square C\ell : \square X \to \square\mathbb{R}$. There is an classifier associated with $\square C\ell$, namely, $\widetilde{C}(B) = \begin{cases} \texttt{IN} & \text{if } \square C\ell(B) > 0, \\ \texttt{OUT} & \text{if } \square C\ell(B) < 0, \\ \texttt{ON} & \text{else.} \end{cases}$
We call $\square C\ell$ a **box function** for $C\ell$ if it is **conservative** (i.e., $C\ell(B) \subseteq \square C\ell(B)$) and **convergent** (i.e., $\lim_i B_i \to p$ implies $\lim_i \square C\ell(B_i) \to C\ell(p)$). The following is straightforward.

Lemma 1. *Let $C\ell : X \to \mathbb{R}$ be a clearance function of a set Y. If $\square C\ell : \square X \to \square\mathbb{R}$ is a box function for $C\ell$, then its associated classifier is a soft Y-classifier.*

In practice, it is easier to design classifiers that focus only on the IN or the OUT decisions. So we call an interval function $\square C\ell : \square X \to \mathbb{R}$ a **positive box function** for Y if $B \subseteq Y$ implies $C\ell(B) \subseteq \square C\ell(B)$, and $\lim_i B_i \to p \in Y$ implies $\lim_i \square C\ell(B_i) \to C\ell(p) > 0$. We similarly define **negative box function** for Y. See [10] (implicitly in [36]) for concrete examples of positive and negative classifiers.

¶12. Implementability. Correct implementation of algorithmic primitives is a central concern of EGC [8]. It remains central for SSS theory. We now indicate why

the soft predicates which we designed in [10] are easily, efficiently and correctly implementable. To use numerical approximations, we need a dense subset $\mathbb{D}$ of $\mathbb{R}$ with good computational properties [34, ¶16]. A simple choice are the **dyadic numbers** $\mathbb{D} = \{m2^n : m, n \in \mathbb{Z}\}$, called BigFloats in software. To exploit hardware arithmetic, we use the technique of **(numerical) filters** in EGC [32]. Basically, filters perform machine arithmetic, but track error bounds to ensure safe decisions. The filter fails when any overflow or underflow is detected, at which point we switch to BigFloats. Using our `Core Library` [35], such filter techniques are automated so that users can write a "standard" `C++` program to implement their predicates.

In a future implementation paper, we will give a careful account of the soft predicates designed in [10], but here is an overview of how to do filters using estimated error bounds. Such bounds suggest that our filters will rarely fail in the typical motion planning experiments. Assume the **Standard Model** of floating point arithmetic [16, p. 44] whereby, for any operation $x \circ y$ ($\circ \in \{+, -, \times, \div, \sqrt{\cdot}\}$) we have $x \widetilde{\circ} y = (x \circ y)(1 \pm \mathbf{u})$ where $x \widetilde{\circ} y$ denote the corresponding approximate arithmetic, $\mathbf{u}$ is unit round-off error, and we use the notation "$\widetilde{x} = x(1 \pm \epsilon)$" to mean $\widetilde{x} = x(1 + \theta)$ for some $\theta \in [-\epsilon, +\epsilon]$ (thus θ is an implicit constant). Note that $\mathbf{u} = 2^{-53} \simeq 1.11 \times 10^{-16}$ for IEEE double precision. The IEEE Standard for hardware arithmetic, and the widely available BigFloat package called `MPFR` [25] follows the Standard Model. For instance, to compute the distance $\|p - q\|$ between two dyadic points, assuming $p \neq q$, then in the Standard Model we can approximate $\|p - q\|$ with relative error of γ_3 where $\gamma_n := \frac{n\mathbf{u}}{1-n\mathbf{u}}$ (see [16] for this γ-analysis). We have $\gamma_n < 2n\mathbf{u}$ unless n is extremely large (e.g., $n > 2^{52}$ for IEEE double). Moreover, if $\ell(x, y) = ax + by + c = 0$ is the equation of a line, then its distance to an arbitrary dyadic point (x_0, y_0) is $|\ell(x_0, y_0)|/\sqrt{a^2 + b^2}$ and this has a relative error of γ_9. This assumes that a, b, c are exact. But if the line is defined by two dyadic points (x_i, y_i) ($i = 1, 2$), then $a = (y_2 - y_1), b = (x_2 - x_1), c = y_1(x_2 - x_1) - x_1(y_2 - y_1)$. Our computation of the distance will now have a relative error of γ_{18} instead of γ_9. The extension of such estimates to the case of rotation or in 3-D will increase n, but remains well under control. To obtain an upper or lower bound on a numerical expression such as $|\ell(x_0, y_0)|/\sqrt{a^2 + b^2}$, we just multiply its computed value by a factor of $(1 + \widetilde{\gamma}_n)$ or $(1 - \widetilde{\gamma}_n)$ where $\widetilde{\gamma}_n$ is an machine upper bound on γ_n. If $n < 128$, say, then $\gamma_n < 2^{-45}$. Barring under or overflows, it means 45 bits of the mantissa are correct; this should suffice for typical applications.

1.4 Dyadic Subdivision Trees

Clearly subdivision trees are capable of many generalizations. So far, we assume that a node is a box in $\mathbb{R}^d$, and it splits into 2^d congruent children. We want to allow non-congruent shapes, and a variable number of children. One motivation is to exploit "anisotropic subdivisions". E.g., in subdivision algorithms for isotopic approximation of curves and surfaces [23, 24, 22], we show empirically that "anisotropic subdivision" could lead to dramatic speedups.

We consider **generalized subdivision trees** whose nodes are cells from a test domain $\square X$ where $X = \mathbb{R}^d$. By a **subdivision** we mean a finite subset $\mathcal{S}$ of $\square X$

such that the intersection of any two distinct cells in $\mathcal{S}$ has dimension $\leq d-1$. If the dimension of intersection is exactly $d-1$, we say the two cells are **adjacent** to each other. A **channel** is a sequence of cells where consecutive pairs are adjacent. We call $\mathcal{S}$ a **subdivision of** $|\mathcal{S}|$ where $|\mathcal{S}|$ denotes the union of the cells in $\mathcal{S}$. By a k**-split** (or split) of a cell $B \in \square X$ we mean a subdivision $\{B_1, \ldots, B_k\}$ of B with $k \geq 2$ cells. We say the split is **dyadic** if each vertex of the B_i's is either a vertex of B or the midpoint of an edge of B. A **dyadic subdivision tree** is a subdivision tree $\mathcal{T}$ in which the children of each internal node forms a dyadic split of its parent. If $\mathcal{T}$ is finite, then the set of leaves of $\mathcal{T}$ forms a **dyadic subdivision** of the root. Dyadic subdivisions for boxes were exploited in [23, 24]. Why dyadic subdivision? In ¶12, we indicated the key role of dyadic numbers. Now each vertex v occurring in a dyadic subdivision tree is a linear combination of the vertices $v_1, \ldots, v_m$ of the root B_0, $v = \sum_{i=1}^{m} \alpha_i v_i$ where each α_i is a dyadic number. We say v is **dyadic relative to** B_0. If B_0 is dyadic, then v is dyadic.

Our definition of a test cell B requires the concept of a center $c(B)$ in the interior of B. A candidate for $c(B)$ is the center of the **circumball**, i.e., unique smallest ball containing B. But this center may not lie in the interior of B. So we first define the **inner radius** $r_0(B)$ of B as the largest radius of a ball contained in B. Then the **incenter** $ic(B)$ comprises the centers of balls of radius $r_0(B)$ that are contained in B. E.g., the incenter of a non-square rectangle is a line segment. Clearly, $ic(B)$ is convex; the center of the circumball of $ic(B)$ is taken to be the **center** $c(B)$. Thus $c(B) \in ic(B)$, and is unique. The smallest ball centered at $c(B)$ and containing B is called the **outer ball** of B, and its radius $r(B)$ is called the **(outer) radius** of B. Let the **aspect ratio** $\rho(B)$ and **width** $w(B)$ (resp., **length** $\ell(B)$) refer (resp.) to $r(B)/r_0(B)$ and the minimum (resp., maximum) length of an edge of B.

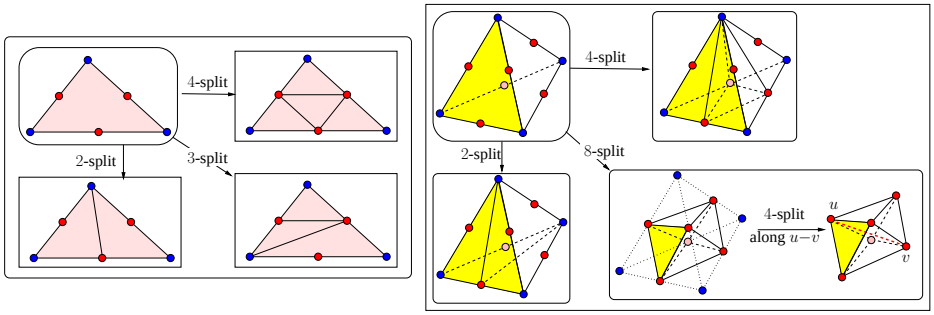

Fig. 1.1 Dyadic splits of (a) triangle and (b) tetrahedron

We turn to **dyadic subdivision schemes**. The dyadic scheme for boxes is discussed in [23, 24, 10]. We briefly consider **dyadic simplicial schemes**. As illustrated in Figure 1.1(a), a triangle has three kinds of dyadic splits: the 4-, 3-, and 2-splits. Dyadic splits of a tetrahedron is more complicated — just three kinds are illustrated in Figure 1.1(b). See [23, 24] for a method to choose among different splits.

1.5 Basic Properties of SSS

We prove some general results about SSS planners for the basic motion planning problem (see ¶5). An "SSS planner" is an algorithm obtained by instantiating the various subroutines in the SSS Framework. The 3 key subroutines are $\widetilde{C}$, `Expand`, and `GetNext`. Our basic assumptions about them are:

- **(C0: Softness)** $\widetilde{C}$ is a soft classifier for C_{free}.
- **(C1: Bounded dyadic expansion)** `Expand` splits a cell dyadically into a bounded number of subcells, each with a bounded number of vertices. Moreover, each edge of a cell must subdivide after a bounded number of splits.

Note that we made no assumptions on `GetNext` here because the needed properties are embedded in the SSS framework: the main property that `GetNext` returns a `MIXED`-leaf with length $\ell(B) \geq \varepsilon$. *Theoretically, it is cleaner to replace "$\ell(B) \geq \varepsilon$" by "$r(B) \geq \varepsilon$" throughout. Our theorems below use this variation.* Although our goal is soft classifiers (C0), as a proof strategy, we will initially assume $\widetilde{C}$ is the exact C_{free}-classifier. In this case, we say our planner is **exact**. (C1) ensure that the boxes in any infinite path converge to a point. Alternative formulations of (C1) are possible. E.g., All cells have bounded aspect ratios. This seems non-trivial to ensure, say, in simplicial subdivision.

Theorem 1. *Every SSS planner halts.*

Thus halting is in-built, depending only on (C1) not (C0). Next we give the minimal correctness property of SSS planners.

Theorem 2 (Exact SSS). *Assuming an exact SSS planner:*
(a) If there is no path, the planner outputs "No Path".
(b) If there is a path with clearance $\geq 2\varepsilon$, the planner outputs a path.

¶13. Resolution Exactness. Loosely interpreted, current "resolution complete" algorithms provide perhaps the equivalent of Theorem 2. But there are three desiderata. First, we want to remove the assumption of exactness in $\widetilde{C}$ (the literature generally assume exactness in its analysis). Second, we would like to weaken the hypothesis of Theorem 2(a) to "if there is no path with clearance ε/K" for some input-independent $K > 1$. Finally, we want to strengthen the conclusion of Theorem 2(b) to output a path of clearance ε/K. This leads to our next definition.

A planner for R_0 is said to be **resolution-exact** if there exists a constant $K > 1$ such that (i) if there is a path from α to β of clearance $K\varepsilon$, it returns a path with clearance ε/K; (ii) if there no path with clearance ε/K, it returns "No Path". Call K an **accuracy constant**.

This definition admits an indeterminacy in the planner. In case the maximum clearance of a path lies in the range $(\varepsilon/K, K\varepsilon)$, a resolution-exact planner may return either a path or "No Path". We show this indeterminacy is unavoidable in [10].

We now require a bit more of soft predicates. Call a Y-classifier $\widetilde{C}$ **effective** if it is[6] **monotone** (i.e., $\widetilde{C}(B) \neq \texttt{ON}$ implies $\widetilde{C}(B') \neq \texttt{ON}$ for all $B' \subseteq B$), and there is a (effectivity) constant $\sigma > 1$ such that if $C_Y(B) = \texttt{IN}$ then $\widetilde{C}(B/\sigma) = \texttt{IN}$.

Lemma 2. *Let $\widetilde{C}$ have effectivity constant $\sigma > 1$. If there is a path of clearance $(1+\sigma)\varepsilon$ then the SSS Planner will output a path.*

This lemma is still not enough for resolution-exactness. We need additional properties of cells in our subdivision, and a simple solution is to exploit the nice properties of boxes:

Theorem 3 (Resolution-Exact). *Let $\widetilde{C}$ be effective and the cells be boxes with bounded aspect ratios. Then the SSS Planner is resolution-exact.*

1.6 What About Exact Algorithms?

Can the SSS framework produce[7] exact algorithms? The answer is yes. But we first point out a non-solution, *using an Exact Planner with the resolution parameter $\varepsilon = 0$*. Using Exact SSS re-introduces the need for algebraic computation. Our normal[8] formulation of SSS requires $\varepsilon > 0$. When $\varepsilon = 0$, indeterminacy is removed, but at a high price: if there is no path, then SSS will not halt. Even if there is a path, we may not find it because of non-halting; this could be fixed by imposing a "generalized BFS" property on `GetNext`.

We now present a solution that exploits resolution-exactness. It is based on the theory of constructive zero bounds [33], and does not need an Exact Planner.

Theorem 4. *If the inputs numbers describing $R_0, \Omega, \alpha, \beta$ are all algebraic numbers, there is an effectively computable number $\delta = \delta(R_0, \Omega, \alpha, \beta) > 0$ with this property: if there is a path from μ from α to β, then the clearance of μ is $> \delta$.*

One way to derive such a δ is to bound the degree and height of algebraic quantities arising in any motion planning algorithm. Then δ could be taken as the root separation for these algebraic quantities. A more careful computation of δ can proceed as follows: using the fact that if there is a path, then there is a path in some "retract" [31] (basically a Voronoi diagram augmented by some paths). This retract is algebraic, and the minimum clearance along the retract could be expressed by the solution of a suitable set of polynomial constraints involving the input data. A zero bound can be computed from a list of these constraints. These bounds depend on the representation used for input angular or rotational parameters.

---

[6] Monotonicity is not strictly necessary, but it simplifies our arguments. Moreover, implementations can normally ensure monotonicity. In the interval literature, it is sometimes called "isotone".

[7] We are indebted to Steve LaValle for asking this question at IROS 2011 Workshop on Progress and Open Problems in Motion Planning in September 30, 2011, San Francisco.

[8] For that matter, we also assumed the accuracy constant K is strictly greater than 1.

Theorem 5. *Suppose we have a resolution-exact planner with accuracy parameter $K > 1$. If we fix the resolution parameter ε to be $\leq \delta(R_0, \Omega, \alpha, \beta)/K$, then the planner is exact.*

¶14. Alternative approach to Exact SSS algorithms. A more practical approach is to avoid zero bounds, and to minimize the role of algebraic computation. As in [10], we maintain a set of features $\phi(B)$ that are within an influence region of B, and another $\phi^-(B) \subseteq \phi(B)$. Our soft classifiers reduce to checking the emptiness of $\phi(B)$ or non-emptiness $\phi^-(B)$. But an exact predicate must ultimately compute the true value of $C_{C_{free}}(B)$. The idea is do this *only* when $\phi(B)$ is "simple", otherwise we split B. Certainly, $|\phi(B)| = O(1)$ may be regarded as simple. Unfortunately, because of input degeneracies, this condition is not enough. Other options for simplicity are possible, but they depend on the nature of R_0. This leads to new exact algorithms that seem more practical than traditional ones.

1.7 Conclusion

In this paper, we described the SSS framework for designing resolution-exact algorithms. We argued that it shares many of the attractive properties of the successful PRM framework. The ideas of resolution-limited algorithms is certainly very old. But to our knowledge, the simple[9] properties of soft classifiers have never been isolated, nor have concepts of resolution-limited computation been carefully scrutinized. We believe focus on these "simple ideas" will open up new classes of algorithms that are practical *and* theoretically sound, not only in motion planning.

There are many open questions concerning SSS framework. Like PRM, many variations of SSS are possible. Perhaps the biggest theoretical challenge is the complexity analysis of adaptive subdivision [29]. Here are some other topics:

- Our SSS framework detects "No Path" by exhaustion. We could speed this up by looking for non-existence of M-paths [36], but it is a challenge to design efficient techniques (this is connected to issues in computational homology).
- We need better subdivision schemes for $SE(3)$ (see [10] for $SE(2)$). Beyond kinematic spaces, good subdivision is even less understood.
- Design and analysis of adaptive search strategies, including randomized or hybrid ones. How efficiently can we update the "dynamic" A-star or Dijkstra search structures of [1]?
- Design and implement new SSS algorithms; compare them with PRM.
- An intriguing question is whether SSS match the performance of PRM in practice. Conventional wisdom says that PRM can provide practical solutions for problems high[10] degrees-of-freedom (DOF) while resolution methods can only

[9] Several reviewers of our previous work sees only the safeness part of soft classifiers. They fail to note that previous work are silent about convergence. Of course, convergence is standard in numerical computing.

[10] Here, we do not consider specialized planning problems (molecules, snakes, humanoids, etc) where the DOF can go much higher than what is discussed here.

reach medium DOF. This seems to be supported by current implementation. Choset [11, p. 202] suggests that the state-of-art PRM can handle DOF in the range $5-12$. They noted that a 10 DOF planar robot from Kavraki (1995) cannot be tackled by other methods. The resolution-based algorithms of Zhang et al [36] involve planar robots (except for one 3D robot). But we believe the full potential of *adaptive* subdivision methods have hardly been reached. So this intriguing question begs for more experiments.

¶15. ACKNOWLEDGMENTS. I am indebted to Yi-Jen Chiang, Danny Halperin, Steve LaValle, and Vikram Sharma for helpful discussions.

References

1. M. Barbehenn and S. Hutchinson. Toward an exact incremental geometric robot motion planner. In *Proc. Intelligent Robots and Systems 95.*, volume 3, pp. 39–44, 1995. 1995 IEEE/RSJ Intl. Conf., 5–9, Aug 1995. Pittsburgh, PA, USA.
2. J. Basch, L. Guibas, D. Hsu, and A. Nguyen. Disconnection proofs for motion planning. In *IEEE Int'l Conf. on Robotics Animation*, pp.No Path1765–1772, 2001.
3. S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*. Algorithms and Computation in Mathematics. Springer, 2003.
4. R. Bohlin and L. Kavraki. A randomized algorithm for robot path planning based on lazy evaluation. In P. Pardalos, S. Rajasekaran, and J. Rolim, editors, *Handbook on Randomized Computing*, pp. 221–249. Kluwer Academic Publishers, 2001.
5. M. Brady, J. Hollerbach, T. Johnson, T. Lozano-Perez, and M. Mason. *Robot Motion: Planning and Control*. MIT Press, 1982.
6. R. A. Brooks and T. Lozano-Perez. A subdivision algorithm in configuration space for findpath with rotation. In *Proc. 8th Intl. Joint Conf. on Artificial intelligence - Volume 2*, pp. 799–806, San Francisco, CA, USA, 1983. Morgan Kaufmann Publishers Inc.
7. J. F. Canny. *The complexity of robot motion planning*. ACM Doctoral Dissertion Award Series. The MIT Press, Cambridge, MA, 1988. PhD thesis, M.I.T.
8. CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.
9. E.-C. Chang, S. W. Choi, D. Kwon, H. Park, and C. Yap. Shortest paths for disc obstacles is computable. *Int'l. J. Comput. Geometry and Appl.*, 16(5-6):567–590, 2006. Special Issue of IJCGA on Geometric Constraints. (Eds. X.S. Gao and D. Michelucci).
10. Y.-J. Chiang and C. Yap. Numerical subdivision methods in motion planning. Poster, IROS Workshop on Progress and Open Problems in Motion Planning. San Francisco, Sep 30, 2011.
11. H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Boston, 2005.
12. M. S. el Din and É. Schost. A baby steps/giant steps probabilistic algorithm for computing roadmaps in smooth bounded real hypersurface. *DCG*, 45(1):181–220, 2011.
13. 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, pp. 755–778. CRC Press LLC, 1997.
14. K. Hauser. *Motion planning for legged and humanoid robots*. PhD thesis, Stanford University, Dec 2008. Department of Computer Science.
15. M. Hemmer, O. Setter, and D. Halperin. Constructing the exact Voronoi diagram of arbitrary lines in three-dimensional space. In *Algorithms ESA 2010*, volume 6346 of *Lecture Notes in Computer Science*, pp. 398–409. Springer Berlin / Heidelberg, 2010.
16. N. J. Higham. *Accuracy and stability of numerical algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, second edition, 2002.

17. D. Hsu, J. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Pro. of IEEE International Conference on Robotics and Automation*, pp. 2719–2726, 1997.
18. D. Hsu, J.-C. Latombe, and H. Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *Int'l. J. Robotics Research*, 25(7):627–643, 2006.
19. L. Kavraki, P. Švestka, 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.
20. S. LaValle, M. Branicky, and S. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *Int'l. J. Robotics Research*, 23(7/8):673–692, 2004.
21. S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, 2006.
22. L. Lin. *Adaptive Isotopic Approximation of Nonsingular Curves and Surfaces*. Ph.D. thesis, New York University, Sept. 2011.
23. 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.
24. L. Lin, C. Yap, and J. Yu. Non-local isotopic approximation of nonsingular surfaces, 2011. Submitted.
25. MPFR Homepage, Since 2000. URL `http://www.mpfr.org/`. MPFR is a C++-library for multi-precision floating-point computation with exact rounding modes.
26. C. Ó'Dúnlaing, M. Sharir, and C. K. Yap. Retraction: a new approach to motion-planning. *ACM Symp. Theory of Comput.*, 15:207–220, 1983.
27. 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.
28. S. Plantinga and G. Vegter. Isotopic approximation of implicit curves and surfaces. In *Proc. Eurographics Symp. on Geom. Processing*, pp. 245–254, New York, 2004. ACM Press.
29. M. Sagraloff and C. K. Yap. A simple but exact and efficient algorithm for complex root isolation. In I. Z. Emiris, editor, *36th ISSAC*, pp. 353–360, 2011. June 8-11, San Jose, CA.
30. J. T. Schwartz and M. Sharir. On the piano movers' problem: II. General techniques for computing topological properties of real algebraic manifolds. *Advances in Appl. Math.*, 4:298–351, 1983.
31. C. K. Yap. Algorithmic motion planning. In J. Schwartz and C. Yap, editors, *Advances in Robotics, Vol. 1: Algorithmic and geometric issues*, pp. 95–143. Lawrence Erlbaum Associates, 1987.
32. C. K. Yap. Towards exact geometric computation. *Comput. Geometry: Theory and Appl.*, 7:3–23, 1997.
33. C. K. Yap. Robust geometric computation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 41, pp. 927–952. Chapman & Hall/CRC, Boca Raton, FL, 2nd edition, 2004.
34. C. K. Yap. In praise of numerical computation. In S. Albers, H. Alt, and S. Näher, editors, *Efficient Algorithms*, volume 5760 of *Lect. Notes in C.S.*, pp. 308–407. Springer-Verlag, 2009.
35. J. Yu, C. Yap, Z. Du, S. Pion, and H. Bronnimann. Core 2: A library for Exact Numeric Computation in Geometry and Algebra. In *3rd Proc. Int'l Congress on Mathematical Software (ICMS)*, pp. 121–141. Springer, 2010. LNCS No. 6327.
36. L. Zhang, Y. J. Kim, and D. Manocha. Efficient cell labelling and path non-existence computation using C-obstacle query. *Int'l. J. Robotics Research*, 27(11–12), 2008.
37. D. Zhu and J.-C. Latombe. New heuristic algorithms for efficient hierarchical path planning. *IEEE Transactions on Robotics and Automation*, 7:9–20, 1991.

1.8 APPENDIX

This appendix contain proofs for our theorems (for WAFR review process).

¶16. Non-Splitting Criterion. Recall that in these theorems, we assume that the criterion "$r(B) < \varepsilon$" is used to stop splitting a box B. The SSS Framework uses the alternative criterion "$\ell(B) < \varepsilon$". Why this difference? Generally, $\ell(B)$ is more easily computable than $r(B)$, so we expect to use $\ell(B)$ in implementations. However, the theory is cleaner if we use $r(B)$. Note that $\ell(B)$ is just the distance between two vertices of B while $r(B)$ involves the center $c(B)$. For instance, if B is a dyadic box, then $\ell(B)$ is a dyadic number while $r(B)$ is a square-root. Thanks to the (C1) assumption that cells have bounded complexity, we know that $r(B) = \Theta(\ell(B))$, so these two criteria are closely related.

Theorem 1. *Every SSS planner halts.*

Proof. Property (C1) implies that in any infinite path $(B_i : i \geq 0)$ of an SSS subdivision tree $\mathcal{T}$, we have $\lim_i \ell(B_i) \to 0$. If we use the "$\ell(B) < \varepsilon$" criterion for non-splitting, then halting is immediate. But if we use the "$r(B) < \varepsilon$" criterion, we also obtain $\lim_i r(B_i) \to 0$, thanks to the fact that cells have bounded complexity (bounded number of vertices). **Q.E.D.**

Remark: (C1) ensures an important property. Take any infinite subdivision tree $\mathcal{T}_\infty$ with no leaves. Then every infinite path in $\mathcal{T}_\infty$ converges to a point. Such a subdivision tree is said to be "complete".

Let $D_m(r)$ denote the ball in X centered at m with radius r.

Theorem 2 (Exact SSS)
Assuming an exact SSS planner:
(a) If there is no path, the planner outputs "No Path".
(b) If there is a path with clearance $\geq 2\varepsilon$, the planner outputs a path.

Proof. Let $\mathcal{T}$ be the subdivision tree at termination.
(a) At termination, we either report a path or output "No Path". Reporting a path would contradict our assumption of no path.
(b) Suppose $\mu : [0,1] \to C_{space}$ is a path from α to β. By way of contradiction, suppose SSS outputs "No Path". This implies that every mixed leaf satisfies $r(B) < \varepsilon$. Consider the set $\mathcal{A}$ of leaves of $\mathcal{T}$ that intersect $\mu[0,1]$ (the trace of μ). If $B \in \mathcal{A}$, there exists $t \in [0,1]$ such that $\mu(t) \in B$. This implies B is either free or mixed. We claim that B is free. If B is mixed, then $r(B) < \varepsilon$ and there is a point $p \in B$ that is semi-free. Then $\|\mu(t) - p\| \leq \|\mu(t) - c(B)\| + \|c(B) - p\| < 2\varepsilon$. Thus p is free since it is contained the ball $D_{\mu(t)}(2\varepsilon)$, and μ has clearance 2ε. This contradicts the assumption that B is mixed. This proves our claim. Now we may form an channel of free cells from α to β using cells in $\mathcal{A}$. So SSS would have reported a path, contradicting our assumption of "No Path". **Q.E.D.**

Lemma 3.
Let $\widetilde{C}$ have effectivity constant $\sigma > 1$. Then if there is a path of clearance $(1+\sigma)\varepsilon$

then the SSS Planner will output a path.

Proof. The proof is similar to that of (b) in the previous theorem, except that now we use the values FREE/MIXED computed by $\widetilde{C}$ instead of the exact notions of free/mixed. Suppose $\mu : [0,1] \to C_{space}$ is a path of clearance $(1+\sigma)\varepsilon$ and, by way of contradiction, our algorithm outputs "No Path". For any leaf B, if there is some $t \in [0,1]$ such that $\mu(t) \in B$, then B is either FREE or MIXED (not STUCK). We claim B is FREE. By way of contradiction, assume B is MIXED. Thus $r(B) < \varepsilon$ and $\|c(B) - \mu(t)\| < \varepsilon$. The ball $D_{\mu(t)}((1+\sigma)\varepsilon)$ is free. Then $\sigma B \subseteq D_{c(B)}(\sigma\varepsilon) \subseteq D_{\mu(t)}((1+\sigma)\varepsilon)$. Thus σB is free and hence $\widetilde{C}(B) =$ FREE. This contradicting proves our claim. Therefore the set of leaves that covers $\mu[0,1]$ must be FREE. **Q.E.D.**

¶17. Conformal Channel of Boxes. For the next theorem, we consider box subdivision. We develop some preliminary concepts.

Let B_1, B_2 be two adjacent boxes. We say B_1, B_2 are **conformally adjacent** if $B_1 \cap B_2$ is a face of both B_1 and B_2. In general, $F = B_1 \cap B_2$ is not a face of B_1, but a subset of a face of B_1. But there is a dyadic subdivision D_1' of B_1 such that F becomes a face of some B_1' in D_1'. Likewise, let D_2' be the subdivision of B_2 such that F a face of some $B_2' \in D_2'$. Now, B_1' and B_2' are conformally adjacent with $B_1' \cap B_2' = F$.

We are not done yet. Say a subdivision D is **conformal** if every pair of adjacent boxes in D are conformally adjacent. Also, let the **width** $w(D)$ of D be the minimum of the widths of boxes in D. Note that $D_1' \cup D_2'$ is not necessarily conformal. A **uniform (dyadic) subdivision** D of a box B is a dyadic subdivision of B in which every box in D are congruent. Clearly, uniform subdivisions are conformal. We may choose D_i to be the uniform subdivision of B_i such that F is a face of some $B_i' \in D_i$. Moreover, we can choose D_i to be the unique minimal subdivision with this property. Then $w(D_1 \cup D_2) = w(\{B_1, B_2\})$. Let us capture this in a lemma:

Conformal Adjacency Lemma.
If B_1, B_2 are adjacent, there exists minimal subdivisions D_i of B_i ($i = 1, 2$) such that $D_1 \cup D_2$ is conformal and there exists boxes $B_i' \in D_i$ where $F = B_1 \cap B_2$ is a face of B_i'. Moreover, $w(\{B_1, B_2\}) = w(D_1 \cup D_2)$.

We extend the notion of conformality to channels. We say a channel $P = (B_1, \ldots, B_m)$ is **conformal** if any two consecutive boxes in P are conformally adjacent. The channel P is **simple** if $D = \{B_1, \ldots, B_m\}$ is a subdivision.

Conformal Channel Lemma.
Let $P = (B_1, \ldots, B_m)$ be a simple channel of boxes with $\alpha \in B_1$ and $\beta \in B_m$. Let $F_i = B_i \cap B_{i+1}$ ($i = 1, \ldots, m-1$). Then there exists dyadic subdivision D_i of B_i ($i = 1, \ldots, m$) such that:

- *$D_1 \cup \cdots \cup D_m$ is conformal.*
- *$w(\{B_1, \ldots, B_m\}) = w(D_1 \cup \cdots \cup D_m)$.*

- *Let B_1'' be any box in D_1 that contains α and B_m' be any box in D_m that contains β. There are boxes B_i' and B_i'' in D_i ($i = 1, \ldots, m$) such that $B_i' \cap B_{i+1}''$ is equal to F_i ($i = 1, \ldots, m-1$). Note that B_i' and B_i'' might be the same.*
- *There is a conformal channel from B_i' to B_i'' comprising of boxes in D_i.*

Theorem 3 (Resolution-Exactness) *Let $\widetilde{C}$ be effective and the subdivision cells are boxes with bounded aspect ratios. Then the SSS Planner is resolution-exact.*

Proof. Suppose there is a path of clearance $(1+\sigma)\varepsilon$. The proof of Lemma 3 shows there is a channel P of free boxes from $B(\alpha)$ to $B(\beta)$. Each box in P has radius $\geq \varepsilon/2$. Because the aspect ratios are bounded, the width $w(P)$ of boxes in P are at least ε/K for some $K > 1$. By the Conformal Channel Lemma, there is a conformal subdivision D of the boxes in P such that $w(D) = w(P)$ and there is a channel P^* of boxes in D from $B(\alpha) \in D$ to $B(\beta) \in D$. Now P^* is conformal since D is conformal. There is an obvious "canonical path" μ from the center of $B(\alpha)$ to $B(\beta)$. We see that the clearance of μ is at least $w(P) \geq \varepsilon/K$.

To obtain a path μ^* from α to β, we can extend μ with two "end segments", viz., the straightline segments from α to $c(B(\alpha))$ and from $c(B(\beta))$ to β. We must now prove that these end segments have clearance ε/K. By symmetry, consider a point p on the end segment from α to $c(B(\alpha))$. We know that $D_\alpha((1+\sigma)\varepsilon)$ is free. If q is any point at distance ε/K from p, then

$$\|\alpha - q\| \leq \|\alpha - p\| + \|p - q\| \leq \varepsilon/2 + \varepsilon/K < 2\varepsilon < (1+\sigma)\varepsilon.$$

This proves that q is free, so μ^* has clearance at least ε/K. **Q.E.D.**

Theorem 4. *If the inputs numbers describing $R_0, \Omega, \alpha, \beta$ are all algebraic numbers, there is an effectively computable number $\delta = \delta(R_0, \Omega, \alpha, \beta) > 0$ with this property: if there is a motion from μ from α to β, then the clearance of μ is $> \delta$.*

The truth of this theorem is not in question. We omit the tedious details of calculating δ in this version. The sketch in the text indicates some ways for doing this.

Theorem 5. *Suppose we have a resolution-exact planner with accuracy parameter $K \geq 1$. If we fix the resolution parameter ε to be $\leq \delta(R_0, \Omega, \alpha, \beta)/K$, then the planner is exact.*

Proof. If there exists a path, then there exists a path of clearance $> \delta \geq K\varepsilon$. By the correctness of our resolution-exact planner, if there is a solution path with clearance $> K\varepsilon$, then our algorithm will return a path. Conversely, if there is no path, then there is no path of clearance $< \varepsilon/K$. By the correctness of our resolution-exact planner, if there is a solution path with clearance $> K\varepsilon$, then our algorithm will return "No Path". **Q.E.D.**