Resolution-Exact Algorithms for Link Robots*

Zhongdi Luo[1], Yi-Jen Chiang[2], Jyh-Ming Lien[3], and Chee Yap[1]

[1] Department of Computer Science, New York University, New York, NY, USA
zl562@nyu.edu, yap@cs.nyu.edu
[2] Department of Computer Science and Engineering, New York University, Brooklyn, NY, USA
chiang@nyu.edu
[3] Department of Computer Science, George Mason University, Fairfax, VA, USA
jmlien@cs.gmu.edu

Abstract. Motion planning is a major topic in robotics. Divergent paths have been taken by practical roboticists and theoretical motion planners. Our goal is to produce algorithms that are practical and have strong theoretical guarantees. Recently, we have proposed a subdivision approach based on soft predicates [18], but with a new notion of correctness called *resolution-exactness*. Unlike most theoretical algorithms, such algorithms can be implemented without exact computation.
In this paper, we describe new resolution-exact techniques for planar link robots. The technical contributions of this paper are the design of soft predicates for link robots, a novel "T/R splitting method" for subdivision, and feature-based search strategies. The T/R idea is to give primacy to the translational (T) components, and perform splitting of rotational components (R) only at the leaves of a subdivision tree. We implemented our algorithm for a 2-link robot with 4 degrees of freedom (DOFs). Our implementation achieves real-time performance on a variety of nontrivial scenarios. For comparison, our method outperforms sampling-based methods significantly. We extend our 2-link planner to thick link robots with little impact on performance. Note that there are no known exact algorithms for thick link robots.

Keywords: exact algorithms, subdivision algorithms, motion planning, soft predicates, resolution-exact algorithms, link robots.

1 Introduction

Algorithmic motion planning is a major topic in robotics. In the last 30 years, many techniques have been developed. Divergent paths have been taken by practical roboticists and theoretical motion planners. There are three main approaches to algorithmic motion planning: exact, sampling and subdivision approaches [11]. The exact approach has been developed by Computational Geometers [6] and in computer algebra [2]. The correct implementation of exact

* The conference version of this paper appeared in *Proc. Workshop on the Algorithmic Foundations of Robotics (WAFR 2014)*. This work is supported by NSF Grants CCF-0917093, IIS-096053, CNS-1205260, EFRI-1240459, and DOE Grant DE-SC0004874.

methods is highly non-trivial because of numerical errors. The sampling approach is best represented by Probabilistic Roadmap (PRM) [8] and its many variants (see [17]). It is the dominant paradigm among roboticists today. Subdivision is one of the earliest approaches to motion planning [4]. Recently, we have revisited the subdivision approach from a theoretical standpoint [18,20]. The present work continues this line of development.

Worst-case complexity bounds in motion planning are too pessimistic and ignore issues like large constants, correct implementation of primitives, and adaptive behavior. Roboticists prefer to use empirical criteria to measure the success of various methods. For instance, Choset et al [5, p. 197-198, Figure 7.1] noted that sampling methods (but not exact or subdivision methods) "can handle" planning problems for a certain[4] 10 degrees of freedom (DOFs) planar robot. It roughly means that sampling methods for this robot could terminate in reasonable time on reasonable examples. Of course, this is a far cry from the usual theoretical guarantees of performance. In contrast, not only there are no exact algorithms for this robot, but the usual exact technique of building the entire configuration space is a non-starter. Likewise, standard subdivision methods would frequently fail on so many degrees of freedom. It is suggested [5, p. 202] that the current state of the art PRM-based planners "can handle" 5- to 12-DOF robots; subdivision methods may reach medium-DOF robots (say, 4 to 7 DOFs). According to Zhang et al [21], there are no known good implementations of exact motion planners for more than 3 DOFs. On the other hand, their work [21] shows that subdivision methods "can handle" 4- to 6-DOF robots, including the gear robot that has complex geometry.

The empirical evidence described in the previous paragraph challenges us to come up with a "theoretical response": can we design theoretical algorithms that are practical and which roboticists want to implement? Our answer may be a little surprising: the answer is yes, but we do not come down on the side of exact algorithms. The three approaches (sampling, subdivision and exact) provide increasingly stronger algorithmic guarantees. So the above empirical observations about their relative abilities is not surprising. Barring other issues, one might think we should use the strongest algorithmic method that "can handle" a given robot. Nevertheless, we suggest [18,20] that subdivision is preferable to both exact and sampling methods for two fundamental reasons. First, robotic systems (sensors, actuators, physical constants[5], mechanical dimensions, environment, etc.) are inherently approximate. Exact computation makes little sense in such a setting, while subdivision appear to naturally support approximation. But to systematically design approximate algorithms, we need a replacement for the standard exact model. We introduced the notion of **soft-predicates** as the basis of an approximate computational model. Second, the difficulty of sampling methods with the **halting problem** is a serious issue in the form of "narrow

[4] This robot was treated in Kavraki's thesis (Stanford 1995) but its appearance seems to go back at least to Latombe and Barraquand [1].

[5] All constants of Physics have at most 8 digits of accuracy. The speed of light is an exception: it is exact, by definition.

passage problem." Intuitively, researchers realize that subdivision can overcome this (e.g., [21]), but there are pitfalls in formulating the solution: the usual notion of "resolution completeness" is vague about what a subdivision planner must do if there is NO PATH: one solution may reintroduce the halting problem, while another solution might require exact predicates. To avoid the horns of this dilemma, we introduce the concept of **resolution-exactness**. Taken together, soft predicates and resolution-exactness, free us from exact computation and the halting problem. They lead to new classes of planning algorithms that are not only theoretically sound, but also practical.

Algorithms that provide resolution exactness promise to recover all the practical advantages of the PRM framework, but with stronger theoretical guarantees. However, many challenges lie ahead to realize these goals. We need to test some of the conventional wisdom of roboticists cited above. Is it really true that subdivision is inherently less efficient than sampling methods? This is suggested by the state-of-art techniques, but we do not see an inherent reason. Is randomness the real source of power in sampling methods? There is some debate among roboticists on this point (cf. LaValle et al [10] and Hsu et al [7]). We feel that the current limit of 6 DOFs of subdivision algorithms is a desired barrier to cross

¶1. Contributions of this Paper. With the foundation of resolution-exactness and soft predicates in place [18,20], we need to develop techniques for designing such algorithms. The present paper contributes to this goal. We focus on techniques for the class of articulated robots. Note that even for a 2-link robot with 4 DOFs, the naive splitting of configuration boxes into $2^4 = 16$ is already unacceptable. It is also clear that any such technique must be empirically supported by implementations. We make several contributions in this paper

(A) Soft-predicates for link robots. As envisioned in [18], soft-predicates can exploit a wide variety of techniques that trade-off ease of implementation against efficiency. In this paper, we introduce soft-predicates based on the notion of **length-limited forbidden angles** for link robots.

(B) A "T/R Splitting" technique based on splitting translational and rotational degrees of freedom in different phases. Consider a freely translating k-link planar robot with $k+2$ DOFs. The naive subdivision would split each box into 2^{k+2} children; already for $k = 2$ or 3, this has little chance of being practical. An idea [18] is to consider two regimes: configuration boxes are originally in the "large regime" in which we only split the translational degrees of freedom. When the boxes are sufficiently small, in the "small regime", we split the angular degrees of freedom. But this idea only delays the eventual 2^{k+2}-way splits. We now take this idea to the limit: we perform the angular split only *once*, at the level just before the leaves. This turns out to be a winner.

(C) Extensions: Subdivision algorithms are typically easier to extend than exact algorithms. For instance, let each robot link be thickened by taking the Minkowski sum of a line segment with a disc of radius $\tau \geq 0$. We say the link is thick when $\tau > 0$. We give a simple heuristic implementation for thick robots which shows little performance penalty. Note that there

are no exact algorithms known for such robots. Another easy extension (not implemented) of our 2-link robot is to a k-spider robot. This is easy because the rotational degree of freedom of each of the links are mutually independent.

(D) We implemented a 2-link robot (with 4 DOFs) in C/C++, and our experiments are extremely encouraging: *our planner can solve a wide range of non-trivial instances in real time. Unlike sampling-based planners, we can terminate quickly in case of NO-PATH, and our algorithm does not need any tuning parameters such as the number of samples, or cut-off bounds.* To evaluate our approach further, we also compared with some probabilistic sampling algorithms (PRM [8], Gaussian-PRM [3] and RRT [9]) implemented in OMPL[16]. Preliminary experiments indicate that our subdivision solution outperforms these **significantly**. Our code and datasets are freely distributed with the `Core Library`[6], where various parameter settings for the experiments on some highly non-trivial instances are reproducibly encoded in the Makefile targets. One such instance is shown in Figs. 5 and 6. A video clip showing the animation of the resulting path is available[7].

2 Preliminaries

The basic motion planning problem is this [11]: Let R_0 be a fixed robot living in $\mathbb{R}^k$ ($k = 2, 3$). It defines a configuration space $C_{space} = C_{space}(R_0)$. We may[8] assume $C_{space}(R_0) \subseteq \mathbb{R}^d$ if R_0 has d DOFs. For any obstacle set $\Omega \subseteq \mathbb{R}^k$, we obtain a corresponding free space $C_{free} = C_{free}(\Omega) \subseteq C_{space}$. The basic (exact) motion planning problem for R_0 is thus: the input is

$$I = (\Omega, \alpha, \beta, B_0) \tag{1}$$

where $\Omega \subseteq \mathbb{R}^k$ is a polyhedral set, $B_0 \subseteq C_{space}$ is a region-of-interest, and $\alpha, \beta \in C_{space}$ are start and goal configurations. We want to find a path in $B_0 \cap C_{free}$ from α to β; return NO-PATH if no such path exists. An algorithm for this problem is called an (exact) "planner".

¶2. Fundamentals of Our Subdivision Approach. Our subdivision approach includes the following three fundamental concepts (the details are given in the Appendix):

– Resolution-exactness: this is our replacement for a standard concept in the subdivision literature called "resolution completeness": Briefly, a planner is resolution-exact if there is a constant $K > 1$ such that if there is a path of clearance $> K\varepsilon$, it will return a path, and if there is no path of clearance

[6] http://cs.nyu.edu/exact/core/download/core/.

[7] http://cs.nyu.edu/exact/gallery/2link/2link.html.

[8] It is standard to identify $C_{space}(R_0)$ with a subset $X \subseteq \mathbb{R}^d$. The topology of $C_{space}(R_0)$ is generally different from that of X. In the case of $k = 2$, the correct topology is easy to simulate since S^1 may be regarded as an interval with the endpoints identified.

ε/K, it will return NO-PATH. Here, $\varepsilon > 0$ is an additional input parameter to the planner, in addition to the normal parameters.

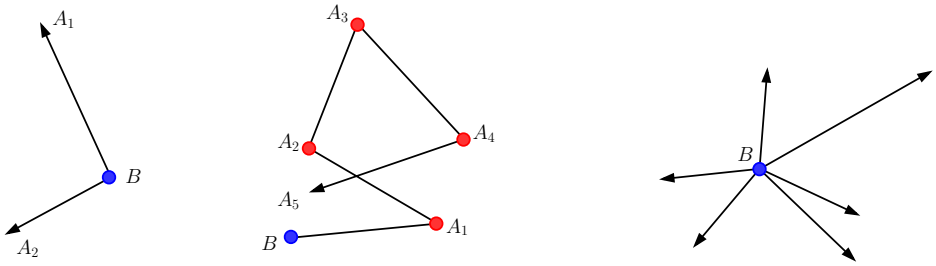
- Soft Predicates: we are interested in predicates that classify boxes. Let $\square\mathbb{R}^d$ be the set of closed axes-aligned boxes in $\mathbb{R}^d$. Let $C : \mathbb{R}^d \to \{+1, 0, -1\}$ be an (exact) predicate where $+1, -1$ are called definite values, and 0 the indefinite value. We extend it to boxes $B \in \square\mathbb{R}^d$ as follows: for a definite value $v \in \{+1, -1\}$, $C(B) = v$ if $C(x) = v$ for every $x \in B$. Otherwise, $C(B) = 0$. Call $\widetilde{C} : \square\mathbb{R}^d \to \{+1, 0, -1\}$ a "soft version" of C if whenever $\widetilde{C}(B)$ is a definite value, $\widetilde{C}(B) = C(B)$, and moreover, if for any sequence of boxes B_i $(i \geq 1)$ that converges monotonically to a point p, $\widetilde{C}(B_i) = C(p)$ for i large enough.
- Soft Subdivision Search (SSS) Framework. This is a general framework for a broad class of motion planning algorithms, in the sense that PRM is also such a framework. One must supply a small number of subroutines with fairly general properties in order to derive a specific algorithm. In PRM, one basically needs a subroutine to test if a configuration is free, a method to connect two free configurations, and a method to generate additional configurations. For SSS, we need a predicate to classify boxes in configuration space as `FREE`/`STUCK`/`MIXED`, a method to split boxes, and a method to test if two `FREE` boxes are connected by a path of `FREE` boxes, and a method to pick `MIXED` boxes for splitting. The power of such frameworks is that we can explore a great variety of techniques and strategies. This is critical for an area like robotics.

¶3. Link Robots. In our previous work [18], we focused on rigid robots. In this work, we look at flexible robots; the simplest such examples are the link robots. Lumelsky and Sun [12] investigated planners for 2-link robots in $\mathbb{R}^2$ and $\mathbb{R}^3$. Sharir and Ariel-Sheffi [14] gave the first exact algorithms for planar k-spider robots.

By a **1-link robot**, we mean a triple $R_1 = (A_0, A_1, \ell)$ where A_0 and A_1 are names for the endpoints of the link, and $\ell > 0$ is the length of the link. Its configuration space is $SE(2) = \mathbb{R}^2 \times S^1$. If $\gamma = (x, y, \theta) \in SE(2)$, then $R_1[\gamma] \subseteq \mathbb{R}^2$ denote the line segment with the A_0-endpoint at (x, y) and the A_1-endpoint at $(x, y) + \ell(\cos\theta, \sin\theta)$. Call $R_1[\gamma]$ the **footprint** of R_1 at γ. Also, $A_0[\gamma], A_1[\gamma] \in \mathbb{R}^2$ denote the endpoints of $R_1[\gamma]$.

For $k \geq 1$, we define a **k-link robot** R_k recursively: R_k will have $k+1$ named points: $A_0, A_1, \ldots, A_k$. We have defined R_1. For $k \geq 2$, R_k is a pair (R_{k-1}, L_k) where $L_k = (X_k, A_k, \ell_k)$, X_k is a named point of R_{k-1}, A_k is the new named point, and $\ell_k > 0$ is the length of the kth link. The configuration space of R_k is $C_{space}(R_k) := \mathbb{R}^2 \times (S^1)^k$, with 2 translational DOFs and k rotational DOFs. See Figure 1 for some examples of such robots (k-chains and k-spiders).

We define the **footprint** of R_k: Let $\gamma = (\gamma', \theta_k) \in C_{space}(R_k)$ where $\gamma' = (x, y, \theta_1, \ldots, \theta_{k-1})$. The footprint of the kth link is $L_k[\gamma]$, defined as the line segment with endpoints $X_k[\gamma']$ and $A_k[\gamma] := X_k[\gamma'] + \ell_k(\cos\theta_k, \sin\theta_k)$. The footprint $R_k[\gamma]$ is the union $R_{k-1}[\gamma'] \cup L_k[\gamma]$.



(a) R_2 (b) Chain Robot (c) Spider Robot

Fig. 1. Some link robots

We say γ is **free** if $R_k[\gamma] \cap \Omega = \emptyset$. As usual, $C_{free}(R_k) \subseteq C_{space}(R_k)$ comprises the free configurations. The **clearance** of γ is defined as $C\ell(\gamma) := \text{Sep}(R_k[\gamma], \Omega)$. Here, $\text{Sep}(X, Y) := \inf\{\|x - y\| : x \in X, y \in Y\}$ denotes the **separation** of two Euclidean sets $X, Y \subseteq \mathbb{R}^2$.

¶4. Feature-Based Approach. Our computation and predicates are "feature based" whereby the evaluation of box primitives are based on a set $\widetilde{\phi}(B)$ of features associated with the box B.

Given a polygonal set $\Omega \subseteq \mathbb{R}^2$, the boundary $\partial\Omega$ may be subdivided into a unique set of **corners** (points) and **edges** (open line segments), called the **features** of Ω. Let $\Phi(\Omega)$ denote this feature set. Our representation of $f \in \Phi(\Omega)$ ensures this **local property of** f: *for any point q, if f is the closest feature to q, then we can decide if q is inside Ω or not.* To see this, first note that if f is a corner, then q is outside Ω iff q is convex corner of Ω. So suppose that f is a wall. Our representation assigns an orientation to f such that q is inside Ω iff q lies to the left of the oriented line through f.

3 The T/R Splitting Method

The simplest splitting strategy is to split a box $B \subseteq \mathbb{R}^d$ into 2^d congruent subboxes. This makes sense for a disc robot, but even for the case of $C_{space} = SE(2)$, this strategy is noticeably slow without additional techniques. In [18], we delay the splitting of rotational dimensions, but the problem of $2^3 = 8$ splits eventually shows up. In this paper, we push the delaying idea to the limit: *we would like to split the rotational dimensions only once, at the leaves of the subdivision tree when the translational boxes have radius at most ε.* Moreover, this rotational split can produce arbitrarily many children, depending on the number of relevant obstacle features. Intuitively, reducing the translational box down to ε for this technique is not severely inefficient because there are only 2 DOFs for translation. Later, we introduce a modification.

The basis for our approach is a distinction between the translational and rotational components of C_{space}. Note that the rotational component is a subspace of a compact space $(S^1)^k$, and thus it makes sense to treat it differently. Given

a box $B \subseteq C_{space}(R_k)$, we write $B = B^t \times B^r$ where $B^t \subseteq \mathbb{R}^2$ and $B^r \subseteq (S^1)^k$ are (respectively) the **translational box** (t-box) and **rotational box** (r-box) corresponding to B.

For any box $B \subseteq C_{space}(R_k)$, let its **midpoint** $m_B = m(B)$ and **radius** $r_B = r(B)$ refer to the midpoint and radius of its translation part, B^t. Suppose the rotational part of B is given by $B^r = \prod_{i=1}^{k}[\theta_i \pm \delta]$.

Suppose we want to compute a soft predicate $\widetilde{C}(B)$ to classify boxes $B \subseteq C_{space}(R_k)$. Following our previous work [18,19], we reduce this to computing a feature set $\widetilde{\phi}(B) \subseteq \Phi(\Omega)$. The **feature set** $\widetilde{\phi}(B)$ of B is defined as comprising those features f such that

$$\mathrm{Sep}(m_B, f) \leq r_B + r_0 \tag{2}$$

where r_0 is farthest reach of the robot links from its base (i.e., A_0). We say B is **empty** if $\widetilde{\phi}(B)$ is empty but $\widetilde{\phi}(B_1)$ is not, where B_1 is the parent of B. We may assume the root is never empty. If B is empty, it is easy to decide whether B is `FREE` or `STUCK`: since the feature set $\widetilde{\phi}(B_1)$ is non-empty, we can find the $f_1 \in \widetilde{\phi}(B_1)$ such that $\mathrm{Sep}(m_B, f_1)$ is minimized. Then $\mathrm{Sep}(m_B, f_1) > r_B$, and by the above local property of features, we can decide if m_B is inside or outside Ω. Here then is our (simplified) *Split*(B) function:

```
Split(B):
    If B is empty,
        Determine if B is free or stuck
    Elif "r(B) > ε"
        T-Split(B)
    Else
        R-Split(B)
```

Here, *T-Split*(B) splits only the translational component B (the rotational component remains the full space, $B^r = (S^1)^k$). Similarly, *R-Split*(B) splits only B^r and leaves B^t intact. The details of *R-Split*(B) are more interesting, and is taken up in the next section.

¶5. Modified T/R Strategy. A possible modification to this T/R strategy is to replace the criterion "$r(B) > \varepsilon$" of *Split*(B) by "$r(B) > \varepsilon$ and $|\widetilde{\phi}(B)| \geq c$", for some (small) constant c. For instance if $|\widetilde{\phi}(B)| = 2$, we might be in a corridor region and it seems a good idea to start to split the angles. The problem with this variation is that the *R-Split*(B) gives only an approximation of the possible rotational freedom in B; if no path is found, we may have to split B^t again, in order to apply *R-Split* to the children of B. This may render it slower than the simple T/R strategy. As our experiments show, a choice like $c = 4$ is a good default.

4 Soft Predicate for Rotational Degrees of Freedom

We design the rotational splitting *R-Split*(B) routine. Recall that this amounts to splitting B^r (leaving B^t intact). First assume the simple case where R_k is

a k-spider. In this case, each link of the robot is independent, so it suffices to consider the case of one link (R_1). Thus $B^r \subseteq S^1$. If this link has length $\ell > 0$, then $R\text{-}Split(B)$ splits the full circle S^1 into a union of free angular intervals. The number of such free angular ranges is equal to the number of features in $\widetilde{\phi}(B)$ within distance ℓ from $m(B)$.

Use the following convention for closed angular ranges: if $0 \leq \alpha_1 < \alpha_2 < 2\pi$, then $[\alpha_1, \alpha_2] := \{\alpha : \alpha_1 \leq \alpha \leq \alpha_2\}$ and $[\alpha_2, \alpha_1] := \{\alpha : 0 \leq \alpha \leq \alpha_1 \text{ or } \alpha_2 \leq \alpha < 2\pi\}$. In any case, if $[\alpha, \alpha']$ is an angular range, we call α (resp., α') the **left** (resp., **right**) **stop** of the range.

For $p, q \in \mathbb{R}^2$, let $Ray(p, q)$ denote the ray originating at p and passing through q, and let $\theta(p, q) \in S^1$ denote its orientation. By convention, the positive x- and y-axes have orientations 0 and $\pi/2$, respectively. If $P, Q \subseteq \mathbb{R}^2$ are sets, let $Ray(P, Q) = \{Ray(p, q) : p \in P, q \in Q\}$.

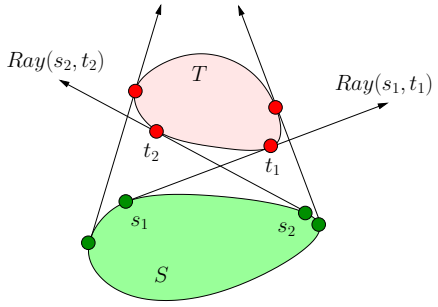


Fig. 2. Common tangent rays: $Ray(p_1, q_1)$ and $Ray(p_2, q_2)$ are the left and right stops of (P, Q).

The main concept we need is the following: for $\ell > 0$, the **length-limited** (or ℓ-limited) **forbidden range** of P, Q is

$$\text{Forb}_\ell(P, Q) := \{\theta(p, q) : p \in P, q \in Q, \|p - q\| \leq \ell\}.$$

If $P \cap Q$ is non-empty, then $\text{Forb}_\ell(P, Q) = S^1$. Hence we will assume $P \cap Q = \emptyset$. We may also assume P, Q are closed convex sets.

Our main task is to provide a compact computational formula for the set $\text{Forb}_\ell(P, Q)$ where P is a box and Q is an edge feature. Without suitable insight, this task can be bogged down in numerous cases, and hard to verify. We present a simplified elegant analysis, initially by considering the case $\ell = \infty$. We simply write $\text{Forb}(P, Q)$ for $\text{Forb}_\infty(P, Q)$. Call $Ray(p, q) \in Ray(P, Q)$ a **common tangent ray** if the line through $Ray(p, q)$ is tangential to P and to Q. Such a ray is **separating** if P and Q lie on different sides of the line through $Ray(p, q)$. If P, Q are not singletons, then there are four common tangent rays, and exactly two of them are separating. We call a separating common tangent

ray a **left stop** (resp., a **right stop**) of (P, Q) if P lies to the right (resp., left) of the ray. Now it is not hard to see that $\text{Forb}(P, Q) = [\theta(p_1, q_1), \theta(p_2, q_2)]$ where $Ray(p_1, q_1)$ and $Ray(p_2, q_2)$ are the left and right stops of (P, Q), as illustrated in Figure 2.

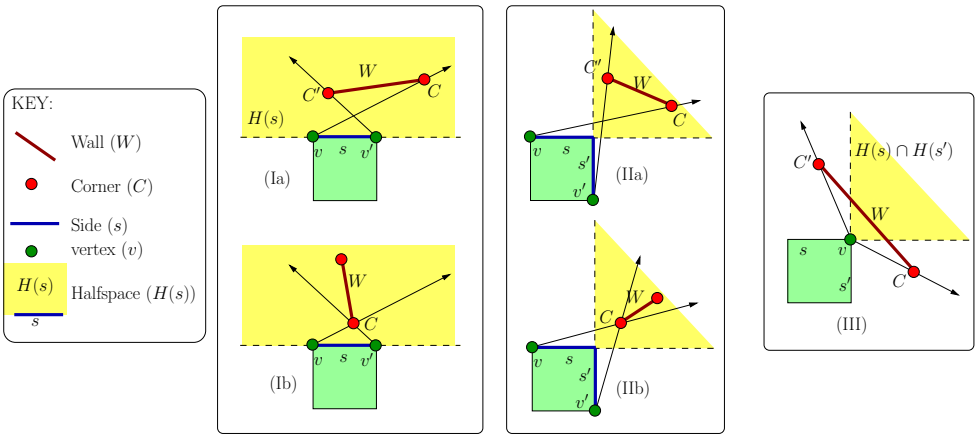


Fig. 3. Forbidden range $\text{Forb}(B^t, W)$ between box B^t and wall W.

We apply these observations to the case where P is a translational box B^t and Q is a wall W. If s is a side of B^t, let $H(s)$ denote the closed half-space bounded by s and that has empty intersection with the interior of B^t. Up to symmetry, there are three cases as seen in Figure 3:

(I) B^t has a unique side s such that $W \subseteq H(s)$.
(II) B^t has two unique sides s and s' such that $W \subseteq H(s) \cap H(s')$.
(III) B^t has two sides s and s' such that $W \subseteq H(s) \cup H(s')$, but is not (I) or (II).

We can now easily compute the forbidden range (refer to Figure 3):

$$\text{Forb}(B^t, W) = \begin{cases} [\theta(v, C), \theta(v', C')] & \text{if Case (Ia) or (IIa),} \\ [\theta(v, C), \theta(v', C)] & \text{if Case (Ib) or (IIb),} \\ [\theta(v, C), \theta(v, C')] & \text{if Case (III).} \end{cases} \tag{3}$$

Next, we must account for the length ℓ. The initial observation is that ℓ-limited forbidden ranges in one of the two forms

$$\text{Forb}_\ell(v, W) \text{ or } \text{Forb}_\ell(s, C) \tag{4}$$

are straightforward to compute:

$$\left.\begin{array}{l} \text{Forb}_\ell(v, W) = \text{Forb}(v, D_\ell(v) \cap W), \\ \text{Forb}_\ell(s, C) = \text{Forb}(D_\ell(C) \cap s, C). \end{array}\right\}$$

where $D_\ell(v)$ and $D_\ell(C)$ are the discs of radius ℓ centered at v and C, respectively. Subsets of S^1 which are expressed in the form (4) are called **cones**. The **cone decomposition** of a subset $F \subseteq S^1$ amounts to writing F as the union of a finite number of such cone sets. For instance, subcase (Ia) in the equation (3) has a cone decomposition comprised of two cones:

$$\text{Forb}_\ell(B^t, W) = [\theta(v, C), \theta(v', C')] = \text{Forb}_\ell(v, W) \cup \text{Forb}_\ell(s, C').$$

The following theorem shows that such a cone decomposition exists in the other cases as well:

Theorem 1. *Any ℓ-limited forbidden range $\text{Forb}_\ell(B^t, W)$ has a cone decomposition comprising at most three cones.*

5 Proof of Theorem 1.

We use the cases in the formula (3) for $\text{Forb}(B^t, W)$ (refer to Figure 3 for notation).

CASE (I) There is a unique side s of B^t such that the wall W lies in the halfspace $H(s)$. We distinguish two subcases: let z denote the intersection of the line through W and line through s. If z lies outside s, then we are in subcase (Ia); otherwise we are in subcase (Ib). The situation where z is undefined because W and s are parallel is treated under subcase (Ia).
First consider subcase (Ia) where C, C' are distinct corners of W. Note that $\text{Forb}(B^t, W) = [\theta(v, C), \theta(v', C')]$ can be written as the union of two angular ranges,

$$\text{Forb}(B^t, W) = \text{Forb}(s, C') \cup \text{Forb}(v, W). \tag{5}$$

However, it could also be written as

$$\text{Forb}(B^t, W) = \text{Forb}(s, C) \cup \text{Forb}(v', W). \tag{6}$$

Can we extend these two representations of $\text{Forb}(B^t, W)$ into a cone decomposition for $\text{Forb}_\ell(B^t, W)$? What if we simply replace $\text{Forb}(s, C')$ by $\text{Forb}_\ell(s, C')$, etc? It turns out that only one of the two extensions is correct. Recall that subcase (Ia) is characterized by the fact that intersection point z lies outside s; wlog, assume that z lies to the left of s as in Figure 4. Suppose $\alpha \in \text{Forb}(B^t, W)$. Then (5) implies that there exists a pair

$$(a, b) \in (s \times C') \cup (v \times W)$$

such that $\theta(a, b) = \alpha$. Similarly, (6) implies that there exists a pair

$$(a', b') \in (s \times C) \cup (v' \times W)$$

such that $\theta(a, b) = \alpha$. One such angle is illustrated in Figure 4 with $(a, b) = (v, b)$ and $(a', b') = (a', C)$. It is easy to verify that this subcase implies

$$\|a - b\| \leq \|a' - b'\|.$$

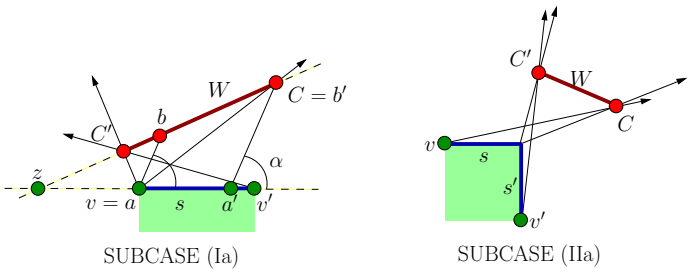


Fig. 4. Length-limited Forbidden Zone Analysis

It follows that

$$\alpha \in \mathrm{Forb}_\ell(B^t, W) \Longleftrightarrow \alpha \in \mathrm{Forb}_\ell(s, C') \cup \mathrm{Forb}_\ell(v, W).$$

In other words, the representation (5) (but not (6)) extends to the ℓ-limited forbidden angles:

$$\mathrm{Forb}_\ell(B^t, W) = \mathrm{Forb}_\ell(s, C') \cup \mathrm{Forb}_\ell(v, W). \tag{7}$$

Note that in case W and s are parallel, both representations (5) and (6) are equally valid.

It remains to treat subcase (Ib), we have $C = C'$ and so the preceding argument reduces to $\mathrm{Forb}_\ell(B^t, W) = \mathrm{Forb}_\ell(C, s)$.

CASE (II) First consider subcase (IIa) where C, C' are distinct corners of W. The analysis of subcase (Ia) can be applied twice to this case, yielding

$$\mathrm{Forb}_\ell(B^t, W) = \mathrm{Forb}_\ell(v, W) \cup \mathrm{Forb}_\ell(s, C') \cup \mathrm{Forb}_\ell(s', C'). \tag{8}$$

For subcase (IIb), we have $C = C'$ and so $\mathrm{Forb}_\ell(v, W)$ can be omitted. Thus $\mathrm{Forb}_\ell(B^t, W) = \mathrm{Forb}_\ell(s, C) \cup \mathrm{Forb}_\ell(s', C)$.

CASE (III) This is simply

$$\mathrm{Forb}_\ell(B^t, W) = \mathrm{Forb}_\ell(v, W). \tag{9}$$

This completes our proof of Theorem 1.

6 Resolution Exactness of our Algorithm

The cone decomposition leads to a simple formula for computing $\mathrm{Forb}_\ell(B^t, W)$. We are ready to describe our *R-Split*(B) operator: Consider the set $\Theta(B) := S^1 \setminus (\bigcup_i \mathrm{Forb}_\ell(B^t, W_i))$ where W_i range over all walls with at least one corner in $\widetilde{\phi}(B)$. Write this set as the union of disjoint angular ranges

$$\Theta(B) := A_1 \cup A_2 \cup \cdots \cup A_k. \tag{10}$$

Each $B^t \times A_i$ is called a **configuration cell** belonging to B, and let $R\text{-}Split(B)$ denote the set of configuration cells belonging to B. Let $B^t \times A$ and $\overline{B}^t \times \overline{A}$ be two configuration cells. We define these cells to be **adjacent** if B^t and $\overline{B}^t$ are adjacent (as translational boxes) and $A \cap \overline{A}$ is non-empty. This definition is justified since, for any $\gamma \in B^t \times A$ and any $\overline{\gamma} \in \overline{B}^t \times \overline{A}$, there is a path from γ to $\overline{\gamma}$. (Here is a proof: By adjacency of $B^t \times A$ and $\overline{B}^t \times \overline{A}$, there is a configuration $\gamma^* \in (B^t \times A) \cap (\overline{B}^t \times \overline{A})$. Then there is a translation and rotation from γ to γ^*, and another translation and rotation from γ^* to $\overline{\gamma}$.) Thus we can reduce motion planning to searching the adjacency graph whose vertices are the configuration cells.

The following lemma is about convergence and effectivity. Clearly, $\bigcup R\text{-}Split(B) \subseteq B \cap C_{free}$. But how good is $\bigcup R\text{-}Split(B)$ as an approximation of $B \cap C_{free}$? This is the question about effectivity of our approximation, and is answered in part(ii) of this lemma.

Lemma 1.
(i) Let $(B_1, B_2, \ldots)$ be a sequence of boxes in C_{space} where $B_i = B_i^t \times S^1$, and B_i^t converges to a point p as $i \to \infty$. Then $\bigcup R\text{-}Split(B_i)$ converges to the set $(p \times S^1) \cap C_{free}$, i.e., the free configurations with the base at p.
(ii) Let $B = B^t \times S^1$. If $\gamma \in B$ has clearance $C\ell(\gamma) > r(B)$, then $\gamma \in \bigcup R\text{-}Split(B)$

Proof. (i) is immediate. To see (ii), let $\gamma = (p, \theta) \in B$. We prove the contrapositive. Suppose $\gamma \notin \bigcup R\text{-}Split(B)$. Then there is some $p' \in B^t$ such that $\gamma' = (p', \theta)$ is not free. But $\mathrm{Sep}(R_1[\gamma], R_1[\gamma']) \leq r(B^t)$. This implies $C\ell(\gamma) \leq r(B^t) = r(B)$. □

Theorem 2. *Assume the T/R method for splitting and R-Split is implemented exactly in our SSS Algorithm for a spider robot R_k. Then we obtain an resolution-exact planner for R_k.*

The proof follows the general approach in [18,20]. Note if we implement $R\text{-}Split(B)$ by a conservative approximation with error that is bounded by $r(B)$, then we obtain a corresponding resolution-exact algorithm (but with larger constant K). Furthermore, if the predicate for each box B is numerically approximated with error at most $2^{-r(B)}$, the resulting algorithm is still resolution-exact [18,20]. In short, exact computation is not necessary. For our present paper, machine accuracy seems to be empirically sufficient for all our examples.

7 Extensions to Thick Links

We could extend the T/R method to spider and chain robots. The efficiency will be minimally impacted in the case of spider robots, but this is less clear for chain robots. In this paper, we implement an extension to links with thickness: each link is now the Minkowski sum of a line segment with a disc of radius $\tau > 0$. Notice that there are no known exact algorithms for thick link robots

(except in the single link case [15]). Let us now define the feature set $\widetilde{\phi}(B)$ of a configuration box B to comprise those features f such that

$$\mathrm{Sep}(m_B, f) \leq r_B + r_0 + \tau. \tag{11}$$

This may be compared to the original criterion (2). When $r(B) \leq \varepsilon$, we must perform *R-Split*(B). This requires us to compute $\mathrm{Forb}_{\ell,\tau}(B^t, W)$, the **$\ell$-limited τ-thick forbidden range** of B^t and W, for various W's. As in the thin case, $\mathrm{Forb}_{\ell,\tau}(B^t, W)$ has a cone decomposition. This reduces to computing the thick cone $\mathrm{Forb}_{\ell,\tau}(v, W)$ (or $\mathrm{Forb}_{\ell,\tau}(s, C)$, but this is similar). We can first compute the thin cone $\mathrm{Forb}_{\ell}(v, W) = [\alpha_1, \alpha_2]$. Then we compute "correction angles" κ_1, κ_2 so that $\mathrm{Forb}_{\ell,\tau}(v, W) = [\alpha_1 - \kappa_1, \alpha_2 + \kappa]$. There is one easy case: suppose a corner C of W determines the angle α_1. Then $\kappa_1 = \arcsin(\tau/d)$ where $d = \|v - C\|$. We have implemented this extension, but as the results show, this has little impact on the performance. In a followup work, we will present the complete analysis of thick link robots.

8 Experimental Results

We have implemented in C/C++ the planner for 2-link robots, both without and with thickness, as described in this paper, and conducted experiments. The platform for the experiments was a workstation with Linux OS, two 3GHz Intel Xeon CPUs and 6GB of RAM.

Our code and datasets are freely distributed with the `Core Library`[9], where various parameter settings for the experiments on some highly non-trivial instances are reproducibly encoded in the Makefile targets. Here we present results on some of these input obstacle sets: eg1, eg2, eg5, eg10, and eg300. Each of these inputs was represented by a set of polygons (not necessarily disjoint), with the dimension of the global environment 512 x 512. For eg300, we generated 300 triangles at random; for other datasets, we generated polygons to form interesting and challenging environments for robot planners. Images of these inputs are shown in Figs. 5-10. Additional experimental results are reported in the Master thesis [13] based on this paper.

For each obstacle set, Table 1 shows two statistics from running our planner: total running time and the total number of tree boxes created. Each run has the parameters (L_1, L_2, T) where L_1, L_2 are the lengths of the 2 links and $T \in \{B, D, G\}$ indicates[10] the search strategy (B = *Breadth First Search* (BFS), D = *Distance + Size*, G = *Greedy Best First* (GBF)). In the left table of Table 1, we pick two variants of T/R splitting: "Simple T/R" means applying *R-Split* when the box size is $< \epsilon$, and "Modified T/R" means applying *R-Split* when the feature set size is small enough (controlled by the parameter c mentioned at the end of Section 3). The choice $c = 4$ is used here. We see that GBF and "Distance + Size" are comparable to each other, and always faster than BFS. Although "Modified T/R" was typically a winner, "Simple T/R" also performed well — the bottom line is that the T/R splitting method, be it "Modified

[9] http://cs.nyu.edu/exact/core/download/core/.

[10] Note that a random strategy is available, but it is never competitive.

T/R" or "Simple T/R", gives a huge performance speed-up. In the right table of Table 1, we compare the performance of robots with various thickness values, where we always used "Modified T/R" with $c = 4$. As can be seen, supporting thickness > 0 is quite easy (in fact quite easy to implement as well), with almost no performance penalty — for some instances (eg5 and eg10) the performance of thickness > 0 was even faster (since thicker robots might result in some boxes to be classified as stuck earlier)! This clearly shows the power of our soft predicates under the resolution-exactness framework.

Obstacle (input)	robot (links)	Modified T/R time (ms)	Modified T/R boxes	Simple T/R time (ms)	Simple T/R boxes
eg1	(50,80,G)	**198.0**	**8232**	198.7	8514
	(50,80,D)	241.1	10886	222.3	10042
	(50,80,B)	486.1	29615	444.0	28802
eg2	(85,80,G)	431.1	23803	564.0	33199
	(85,80,D)	394.5	21400	**367.4**	**20060**
	(85,80,B)	681.8	53393	575.4	48851
eg5	(60,50,G)	655.1	22781	**638.2**	**22617**
	(60,50,D)	751.8	25007	759.4	27185
	(60,50,B)	806.6	40007	803.9	39868
eg10	(65,80,G)	129.6	9060	129.7	9060
	(65,80,D)	**95.2**	**7380**	**95.2**	**7380**
	(65,80,B)	169.6	15434	169.7	15434
eg300	(40,30,G)	**256.6**	**6132**	259.6	6133
	(40,30,D)	267.6	6376	262.6	6337
	(40,30,B)	3125.0	52318	2865.6	49944

robot & input (links)				
	time (ms)	boxes	time (ms)	boxes
(50,80,G)	thickness: 5		thickness: 6 (*)	
eg1, $\epsilon = 4$	280.4	95880	1368.5	62080
(85,80,G)	thickness: 0		thickness: 6	
eg2, $\epsilon = 2$	588.7	35302	1618.2	67023
(43,43,G)	thickness: 0		thickness: 9	
eg5, $\epsilon = 2$	2723.6	84867	2307.9	69774
(45,45,G)	thickness: 0		thickness: 18	
eg10, $\epsilon = 2$	518.3	28129	503.9	19515
(40,30,G)	thickness: 0		thickness: 7 (*)	
eg300, $\epsilon = 2$	944.9	19297	2359.9	33248

Table 1. Statistics of running our algorithms. In the left table, all instances are with thickness 0 and $\epsilon = 4$. In the right table, the thickness and ϵ values are explicitly shown. The instances of "No Path Found" are marked with "(*)".

In Table 2, we compare the performance of our planner for 2-link robots (with thickness 0) with those of sampling-based methods RRT, PRM and Gaussian-PRM (PRM planner with GaussianValidStateSampler) implemented in OMPL[16] (The Open Motion Planning Library) version 0.14.1. For these sampling-based methods, the time limit for solving motion planning problem was set to 300 seconds, and all planner specific parameters were using the OMPL default values. (Note that our planner only has a parameter ϵ (for "Modified T/R" we always used $c = 4$) — therefore, in our method and OMPL the default parameters were used in all experiments.) We report in Table 2 the average results over 31 runs for these sampling-based methods, where we see that overall Gaussian-PRM had the highest success rate within the given running time, while RRT performed the worst. As can be seen, our inputs were very challenging for all these sampling-based methods, and our running times were **significantly faster** than all these methods — For example, comparing with the best running times of the three sampling-based methods, for eg1 (198.0ms vs. 2484ms) we were 12.55 times as fast, for eg2 (367.4ms vs. 3390ms) we were 9.23 times as fast, for eg5 (638.2ms vs. 68865ms) we were 107.91 times as fast, and for eg300 (256.6ms vs. 15885ms) we were 61.91 times as fast. These results show that our new algorithms, in addition

to providing stronger theoretical guarantees, also achieve superior performance gains in practice.

Obstacle	Ours	RRT				PRM				Gaussian-PRM			
(input)	T	N	S	T	STD	N	S	T	STD	N	S	T	STD
eg1	**198.0**	11067	1.000	19559	9744	11060	1.000	6134	6908	5627	1.000	**2484**	1442
eg2	**367.4**	6531	0.710	201980	81597	9320	1.000	**3390**	1735	9314	1.000	3438	1615
eg5	**638.2**	5573	0.581	83967	32758	105506	0.516	106713	33993	72821	0.710	**68865**	38116
eg10	**95.2**	308	1.000	9089	17673	701	1.000	**173**	169	676	1.000	176	161
eg300	**256.6**	3128	1.000	**15885**	12784	9871	1.000	32053	11489	10467	1.000	34151	13480

Table 2. Comparing the performance of our approach with the sampling-based methods RRT, PRM and Gaussian-PRM, for 2-link robots with thickness 0. For our approach, the running time (T, in milliseconds (ms)) is the best instance given in the left table of Table 1 (shown in bold both there and here). For the sampling-based methods, N is the average number of samples, S is the success rate over 31 runs, T is the average running time over 31 runs in milliseconds (ms), and STD is the standard deviation of the running times with respect to T. For each dataset, the best T among the 3 sampling-based methods are also shown in bold.

9 Conclusions

We hope that the focus on soft methods will usher in renewed interest in theoretically sound and practical algorithms in robotics, and more generally in Computational Geometry. Our experimental results for link robots offer hopeful signs that this is possible.

Our basic SSS framework (like PRM) is capable of many generalizations for motion planning. One direction is to consider multiple-query models; another is to exploit the stuck boxes for faster termination in case of NO-PATH. Extensions to kinodynamic planning offer a chance at practical algorithms in this important area where no known theoretical algorithms are practical. Much theoretical and complexity analysis remains open.

It is clear that the theory of soft subdivision methods can be generalized and extended to many traditional problems in Computational Geometry. But it can also extend to new areas that are currently untouchable by our exact computational models, especially those defined by non-algebraic continuous data.

References

1. J. Barraquand and J.-C. Latombe. Robot motion planning: a distributed representation approach. *Int'l. J. Robotics Research*, 10(6):628 – 649, 1991.
2. S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry.* Algorithms and Computation in Mathematics. Springer, 2nd edition, 2006.
3. V. Boor, M. H. Overmars, and F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proc. IEEE Robotics and Automation*, volume 2, pages 1018–1023. IEEE, 1999.
4. 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*, pages 799–806, San Francisco, CA, USA, 1983. Morgan Kaufmann Publishers Inc.
5. 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.
6. 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.
7. 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.
8. 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.
9. J. J. Kuffner Jr and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.
10. 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.
11. S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, 2006.
12. V. Lumelsky and K. Sun. A unified methodology for motion planning with uncertainty for 2d and 3d two-link robot arm manipulators. *Int'l. J. Robotics Research*, 9:89–104, 1990.
13. Z. Luo. Resolution-exact planner for a 2-link planar robot using soft predicates. Master thesis, New York University, Courant Institute, Jan. 2014. Master Thesis Prize 2014.
14. M. Sharir and E. Ariel-Sheffi. On the piano movers' problem: IV. various decomposable two-dimensional motion planning problems. NYU Robotics Report 58, Courant Institute, New York University, 1983.
15. 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.
16. I. Şucan, M. Moll, and L. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012. http://ompl.kavrakilab.org.
17. S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT press, Cambridge, MA, 2005.
18. C. Wang, Y.-J. Chiang, and C. Yap. On Soft Predicates in Subdivision Motion Planning. In *29th ACM Symp. on Comp. Geom. (SoCG'13)*, pages 349–358, 2013. To appear *CGTA, Special Issue for SoCG'13*.
19. C. Yap, V. Sharma, and J.-M. Lien. Towards Exact Numerical Voronoi diagrams. In *9th Proc. Int'l. Symp. of Voronoi Diagrams in Science and Engineering (ISVD)*., pages 2–16. IEEE, 2012. Invited Talk. June 27-29, 2012, Rutgers University, NJ.
20. C. K. Yap. Soft Subdivision Search in Motion Planning. In *Proceedings, Robotics Challenge and Vision Workshop (RCV 2013)*, 2013. **Best Paper Award**, sponsored by Computing Community Consortium (CCC). Robotics Science and Systems Conference (RSS 2013), Berlin, Germany, June 27, 2013. In arXiv:1402.3213v1 [cs.RO]. Full paper from: http://cs.nyu.edu/exact/papers/.
21. L. Zhang, Y. J. Kim, and D. Manocha. Efficient cell labeling and path non-existence computation using C-obstacle query. *Int'l. J. Robotics Research*, 27(11–12), 2008.

APPENDIX 1: Experimental Setup

The following figures (Figs. 5-10) show the GUI interface to our implementation. The right hand control panel allows the user to rerun with new parameters, replay the animation, to replay the splitting of boxes, change ε or the start and goal configurations, choose different global search strategies, modify dimensions of the robot, etc. Subdivision boxes can also be queried.

We have a collection of predefined environments, and for each environment, we selected some interesting parameters and encode them as targets for a Makefile. These targets are named "egX" where X=0,1,2, etc. The examples run in real time, even in case of NO-PATH (a feat that would challenge sampling-based methods). Correctness of the solution is independent of the search strategy. The straightforward randomized strategy tend to be the slowest.

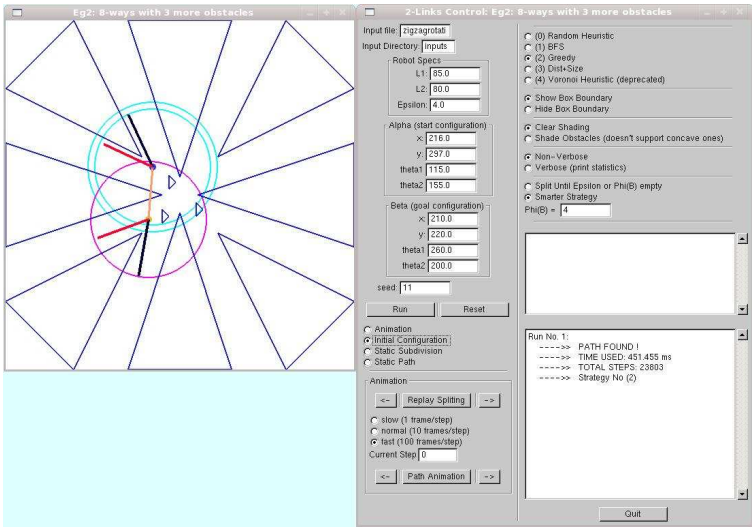


Fig. 5. Initial configuration on the input "eg2". The obstacle set has 8 big triangles forming 8-way "corridors", plus 3 small triangles in the center. We also show the starting and ending configurations of the 2-link robot (enclosed in a single circle and double circles respectively).

APPENDIX 2: Elements of SSS Theory

We describe the three main ingredients of our approach. See [18,20] for more details.

¶6. Resolution-exactness. Resolution-exact algorithms only has meaning relative to some exact planner A. We use the fact that each path has a positive **clearance** (minimum separation of R_0 from Ω as it moves along the path). Then a **resolution exact planner** B is one that takes the same input (see (1)) as A, but has an additional input parameter $\varepsilon > 0$, such that if there is a path of

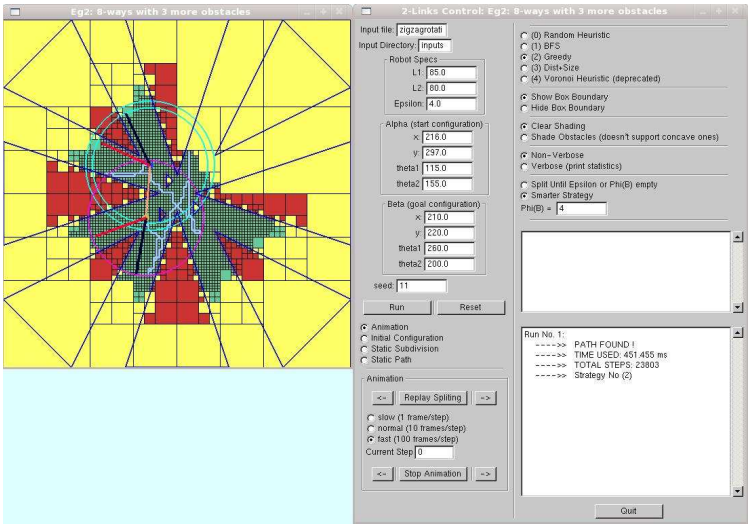


Fig. 6. Final configuration after running on the input "eg2". The resulting path is shown, and the leaf boxes obtained during the subdivision search for the path are displayed and color coded.

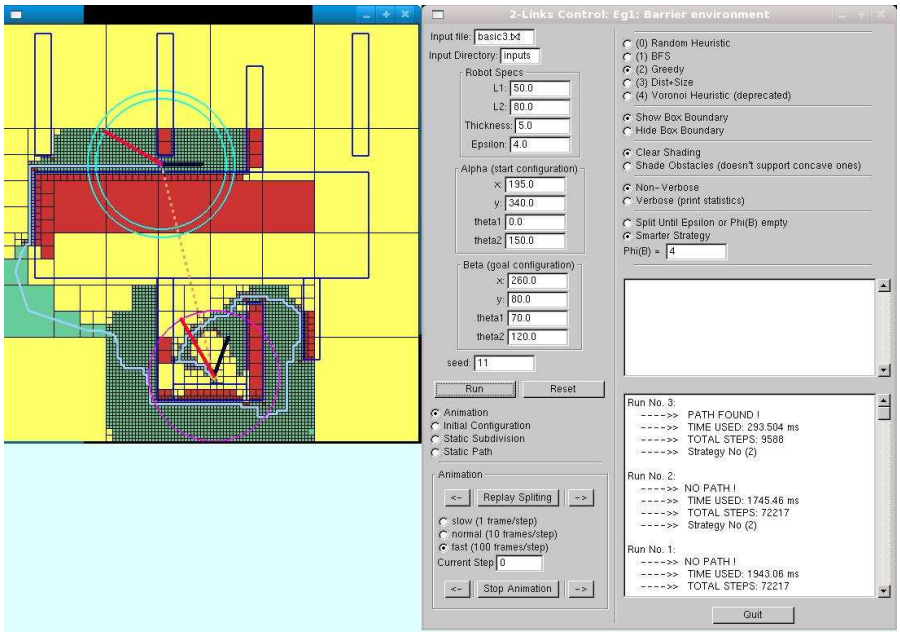


Fig. 7. The "eg1" dataset, which gives a "barrier" environment. Here the robot thickness is 5.

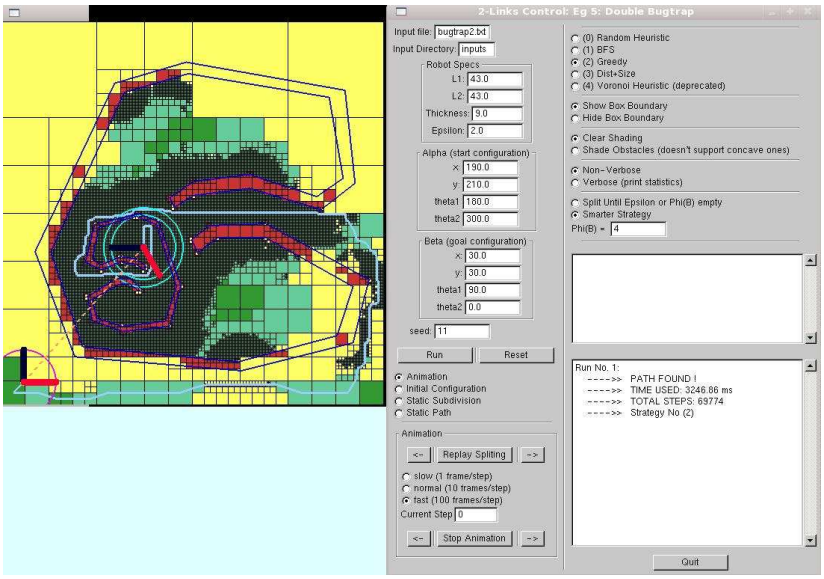


Fig. 8. The "eg5" dataset, which gives a "double bugtrap" environment. Here the robot thickness is 9.

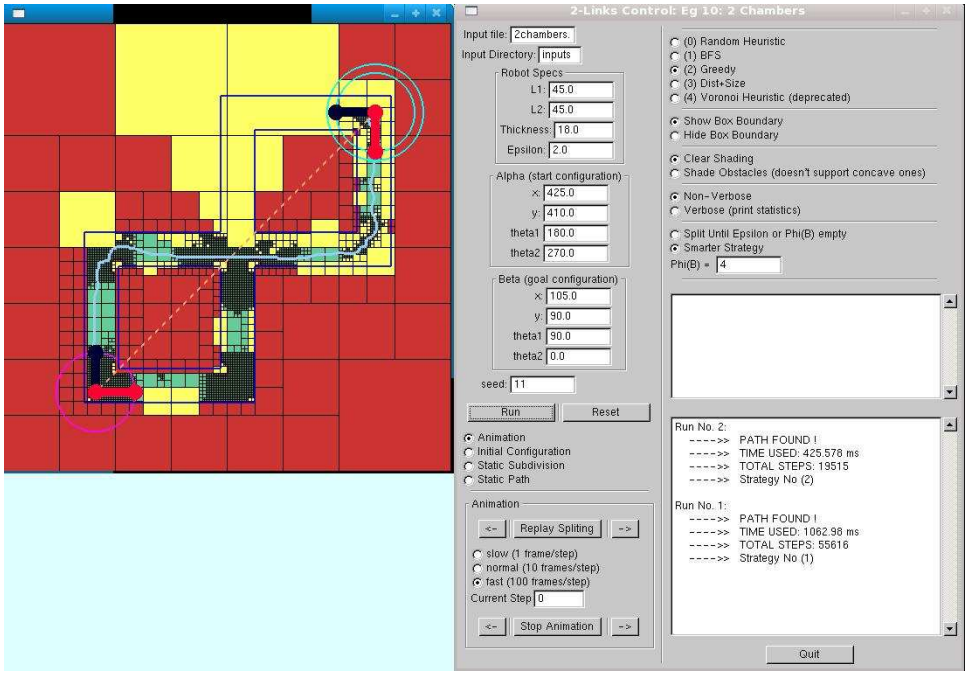


Fig. 9. The "eg10" dataset, which gives a "2 chambers" environment. Here the robot thickness is 18.

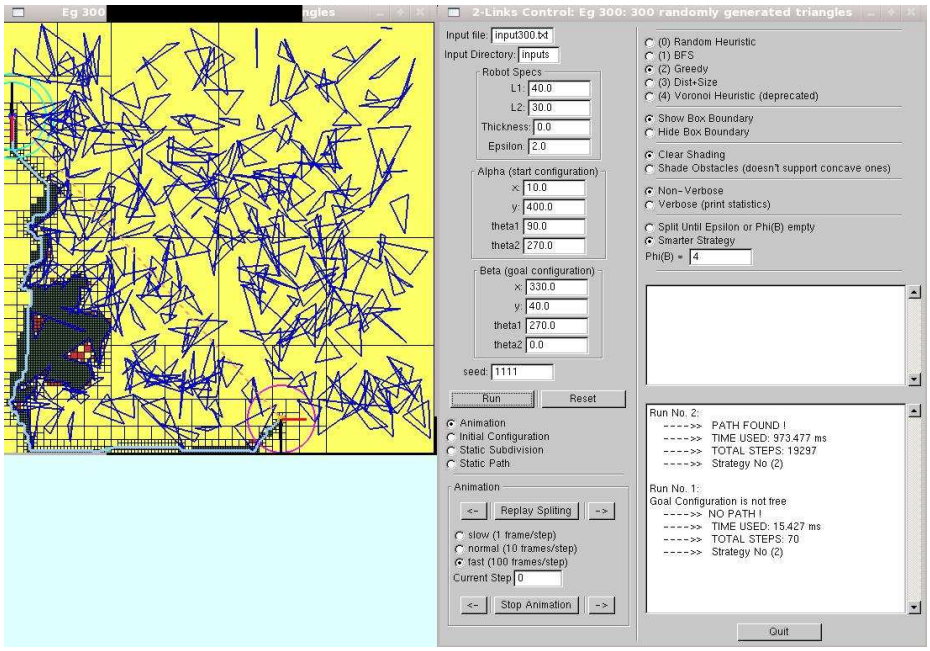


Fig. 10. The "eg300" dataset, which has 300 randomly generated triangles. Here the robot thickness is 0.

clearance $> K\varepsilon$, it will output a path; if there are no paths of clearance ε/K, it will output NO-PATH. Here $K > 1$ depends only on B but not on the inputs. If paths exist, and their clearance lies between ε/K and $K\varepsilon$, then algorithm B can return a path or NO-PATH. Similar definitions can be given for other geometric problems (see [19] for Voronoi diagrams).

¶7. Soft Predicates. The concept of a "soft predicates" is also relative to the some exact predicate. In motion planning, consider the predicate $C : C_{space} \to \{+1, -1, 0\}$ where $C(x) = +1/-1/0$ (resp.) if configuration x is free/semi-free/stuck where semi-free configurations are those on the boundary of C_{free}. Call $+1, -1$ the **definite values**, and 0 the **indefinite value**. Extend the definition to sets $B \subseteq C_{space}$: for a definite value v, define $C(B) = v$ iff $C(x) = v$ for all x. Otherwise, $C(B) = 0$. Typically, $C(x) = \texttt{sign}(c(x))$ where $c : \mathbb{R}^d \to \mathbb{R}$ is some continuous real function. To restrict B to nice sets, let $\square(\mathbb{R}^d)$ denote the set of axes-aligned boxes in $\mathbb{R}^d$. A predicate $\widetilde{C} : \square(\mathbb{R}^d) \to \{-1, 0, +1\}$ is a **soft version of** C if it is conservative and convergent. Conservative means that if $\widetilde{C}(B)$ is a definite value, then $\widetilde{C}(B) = C(B)$. Convergent means that if for any sequence $(B_1, B_2, \ldots)$ of boxes, if $B_i \to p \in \mathbb{R}^d$ as $i \to \infty$, then $\widetilde{C}(B_i) = C(p)$ for i large enough. Note that soft predicates are relatively easy to define and implement using BigFloat arithmetic and intervals. To achieve resolution-exact algorithms, we must ensure $\widetilde{C}$ converges fast enough in this sense: say $\widetilde{C}$ is

effective if there is a constant $\sigma > 1$ such that if $C(B)$ is definite, then $\widetilde{C}(B/\sigma)$ is definite.

¶8. The Soft Subdivision Search Framework. The SSS algorithm amounts to maintaining a subdivision tree $\mathcal{T} = \mathcal{T}(B_0)$ rooted at B_0. Each tree node is a box (Cartesian product of closed intervals). By a (box) **subdivision** of a box B we mean a finite set S of boxes whose union equals B, and where the boxes in S have pairwise disjoint interiors. We assume a procedure $\texttt{S}split(B)$ that takes a box B, computes a subdivision S of B, and attaches each box in S as a child of B in $\mathcal{T}$. In this way, B is "expanded" and no longer a leaf. For example, $\texttt{S}split(B)$ might create 2^d congruent subboxes as children. Initially $\mathcal{T}$ has just the root B_0; we expand $\mathcal{T}$ by repeatedly calling $\texttt{S}split(\cdot)$. Thus, the set of leaves of $\mathcal{T}$ is a subdivision of B_0. The nodes of $\mathcal{T}$ are classified using a soft predicate $\widetilde{C}$ as above. In this context, we prefer to write $\widetilde{C}(B) = \texttt{FREE}/\texttt{STUCK}/\texttt{MIXED}$ instead of $\widetilde{C}(B) = +1/-1/0$ (resp.). Only `MIXED` leaves will be split. We need to maintain three auxiliary data structures:

- A priority queue Q containing `MIXED` boxes with radius $> \varepsilon$. Let $Q.\texttt{getNext}()$ remove the box of highest priority from Q. The tree $\mathcal{T}$ grows by splitting $Q.\texttt{getNext}()$.
- A graph G whose nodes are the `FREE` leaves in $\mathcal{T}$, and whose edges connect pairs of boxes that are adjacent, i.e., that share a $(d-1)$-face.
- A union-find data structure for connected components of G. We assumed that $\texttt{S}split(B)$ will automatically insert the `MIXED` children of B with radius $> \varepsilon$ into Q, and update G and the union-find data structure with the `FREE` children of B.

Here is the basic SSS algorithm: let $Box_{\mathcal{T}}(\alpha)$ denote the leaf box containing α (similarly for $Box_{\mathcal{T}}(\beta)$). Note that there are two while-loops before the MAIN LOOP, and these keep splitting $Box_{\mathcal{T}}(\alpha)$ and $Box_{\mathcal{T}}(\beta)$ until they are `FREE`, or else return NO-PATH.

SSS Framework:
Input: Configurations α, β, tolerance $\varepsilon > 0$, box $B_0 \in \mathbb{R}^d$.
  Initialize a subdivision tree $\mathcal{T}$ with root B_0.
  Initialize Q, G and union-find data structure.
1. While $(Box_{\mathcal{T}}(\alpha) \neq \texttt{FREE})$
    If radius of $Box_{\mathcal{T}}(\alpha))$ is $\leq \varepsilon$, Return(NO-PATH)
    Else $\texttt{S}split(Box_{\mathcal{T}}(\alpha))$
2. While $(Box_{\mathcal{T}}(\beta) \neq \texttt{FREE})$
    If radius of $Box_{\mathcal{T}}(\beta))$ is $\leq \varepsilon$, Return(NO-PATH)
    Else $\texttt{S}split(Box_{\mathcal{T}}(\beta))$
3. While $(Find(Box_{\mathcal{T}}(\alpha)) \neq Find(Box_{\mathcal{T}}(\beta)))$ ◁ *MAIN LOOP*
    If $Q_{\mathcal{T}}$ is empty, Return(NO-PATH)
(*) $B \leftarrow \mathcal{T}.\texttt{getNext}()$
    $\texttt{S}split(B)$
4. Generate a path from α to β using G.

The correctness of our algorithm does not depend on how the priority of Q is designed. We can even use random priority. But in practice, it is clear that Q.`getNext`() is critical and represent the global search strategy.