5-30-2015

Cell Representations of the Configuration Space for Planning Optimal Paths

Ajay Kannan
Dartmouth College

Cell Representations of the Configuration Space for Planning Optimal Paths

Ajay Kannan
Advisor: Devin Balkcom

May 30, 2015

Abstract

This paper proposes sampling techniques to approximate the configuration space for optimal motion planning. We sample valid configurations in the workspace and construct path subconvex cells in the free configuration space. The radius of each cell is calculated using lower bounds on the robot's minimum time to collision. Using theorems about path convexity, the shortest paths found between any two points in the decomposed space are guaranteed to be safe. Experimental results are provided for a planar arm.

Dartmouth Computer Science Technical Report TR2015-776

Contents

1 Introduction **3**
1.1 Related Work 3
1.2 Main Contribution 4

2 Path Subconvexity and Coverage Guarantees **4**
2.1 Path convex and subconvex sets 5
2.2 Optimal Path Guarantees with $\mathcal{C}_\epsilon$ Coverage 6
2.3 Requirements to Guarantee $\mathcal{C}_\epsilon$ Coverage 6

3 Constructing Path Subconvex Balls **6**
3.1 Case Study: Robotic Arm 7

4 $\mathcal{C}_\epsilon$ Coverage Algorithms **8**
4.1 Cell Decomposition 8
4.2 Experimental Results 8

5 Algorithm Adaptations **9**
5.1 Techniques to Increase Cell Size 9
5.2 Distributing Work 14

6 Conclusions **15**

7 Acknowledgments **16**

1 Introduction

This paper proposes an algorithm for configuration space decomposition that is useful for planning optimal paths. Each cell in the decomposition is path subconvex (defined by Balkcom *et al.* [1]) to the free space. This property guarantees a collision-free shortest path between any two points in the same connected component of the decomposition. The outcome of this algorithm is a set of cells that can be queried for lowest-cost paths. These lowest-cost paths are optimal over all possible paths that do not venture closer than the tolerable error ϵ to obstacles in the configuration space.

With an approximation of the free configuration space in hand, previously difficult motion planning problems become tractable. Optimal motion plans can be found by (1) creating a connectivity graph of points within and along cell boundaries and then (2) pathfinding with a graph algorithm. The decomposition can also be leveraged for other tasks, such as obtaining diverse sets of paths around obstacles, creating motion plans with varying margins for error, and determining free space connectedness. While some of these tasks are achievable using different flavors of point-sampling planners, it is often unclear when to stop sampling to achieve a desired level of accuracy.

1.1 Related Work

Approximate cell decomposition (ACD), hybrid motion planning, and optimal motion planning algorithms are relevant to the work presented here. ACD algorithms divide the configuration space into cells and label them as free, collision, or mixed (for example, see [7]). A similarity between ACD and the algorithm developed in this paper is that both construct approximations of the free space using cells. However, ACD algorithms are tangential in focus to the algorithm presented here. Most importantly, generic ACDs are not concerned with path convexity. Thus, generic decompositions do not provide enough information to guarantee path safety for some robots. Path convexity and provably safe optimal trajectories, however, are the crux of this paper.

Some have sought hybrid approaches to combine the benefits of decomposition and sampling-based motion planning. Zhang, Kim, and Manocha's PRM-augmenting decomposition and Saltzman, Hemmer, Raveh, and Halperin's manifold sampling are two such examples. Zhang *et al.* place free point samples in mixed cells, as determined by a hierarchical ACD. Because sampling only occurs in mixed cells, the roadmap is far denser near boundaries than in a regular PRM [6]. Furthermore, they avoid generating large numbers of cells by decomposing a mixed cell only when a path cannot be found via the roadmap of samples [6]. Their algorithm provides information about path feasibility and is more time and memory efficient than the one presented here. However, PRM-augmented decomposition does not provide an optimal path guarantee that is stronger than a PRM's asymptotic guarantee.

Saltzman *et al.* suggest planning paths using manifold sampling [4]. Their algorithm consists of using simple, overlapping manifold samples to create a connectivity graph for similar purposes as a PRM, but with fewer samples. Like the algorithm in this paper, each sample captures space information rather than just a single point. Their paper does not seek to provide guarantees for optimal motion planning.

There are also approaches such as PRM* that provide optimal motion planning without cell decomposition. PRM* provides asymptotic optimality by specifying the distance for which roadmap connections must be checked as a function of the number of samples [3]. Finally, there has been work on optimal steering methods for some specific systems, such as variants of wheeled vehicles (as mentioned in Balkcom *et al.* [1]). This paper carves out a niche in that it is a decomposition approach that provides optimality guarantees for generalized robotic systems with a finite amount of sampling.

1.2 Main Contribution

We propose sampling configurations in the workspace and constructing metric cells completely contained within the free space. These cells can be used for path planning with optimality guarantees with a degree of error set as a parameter.

We construct a free cell by calculating a lower bound on the minimum time to collision from a valid sampled configuration. The time to collision is given by dividing the minimum distance between the robot and an obstacle in the work space by an upper bound on the maximum speed of the robot. Using bounds on time to collision and maximum speed is a practically-driven choice. It is often not feasible to precisely calculate these values for robots.

Given a configuration space C, the skeleton of the algorithm we propose is as follows:

1. Sample a point $q \in \mathcal{C}$.
2. If $q \in \mathcal{C}_{free}$, compute a lower bound on the minimum time to collision t_{col}. Optionally, if $q \in \mathcal{C}_{obs}$, compute a lower bound on the minimum time to escape collision t_{escape}.
3. Let R denote the set of reachable configurations within t_{col} (or t_{escape}). Choose a set $S \subseteq R$ which is path subconvex with respect to R.
4. Create an L_∞ cell within S.

There are many computational costs to be mindful of in this approach. Naturally, we wish to maximize the size of free space cells. With larger cells, the hierarchical sampling process can terminate earlier to acquire the same coverage of the configuration space. This also leads to less data processing costs when using cells for motion planning problems. The size of the cells depends on the quality of the lower bound on time to collision. However, computing high quality lower bounds on time to collision may increase computation time. Thus, a balancing act between quality of the bound and time cost arises.

Though time and memory costs can be improved to some extent, the number of cells will still be large for the vast majority of complex robots with many obstacles. Because this decomposition is a one-time task for a particular robot and set of static obstacles, we envision scenarios where it may be reasonable to invest a lot of computing power towards this task. Hence, parallelization results are provided and discussed.

2 Path Subconvexity and Coverage Guarantees

This section provides crucial theoretical background necessary for this algorithm's safety and optimality guarantees. This groundwork yields two important outcomes:

1. Let S be a set of cells path subconvex to the free configuration space covering $\mathcal{C}_\epsilon$, the free configuration space more than ϵ workspace distance from obstacles. Then S contains optimal motion plans between any two points in $\mathcal{C}_\epsilon$ that venture no closer than ϵ to obstacles.

2. We can narrow the search for optimal paths contained within $\mathcal{C}_\epsilon$ given set S.

The definitions and theorems provided in this section were originally described in Balkcom *et al.* [1].

2.1 Path convex and subconvex sets

To define path convexity, we must first formalize the definition of a steering method. A steering method S is a set of continuous curves in $\mathcal{C}_{free}$ such that $\forall x, y \in \mathcal{C}_{free}$, there exists a curve in S from x to y (provided such a curve is possible). S is optimal under metric d if each curve from x to y is minimum cost over all possible curves from x to y, according to d.

Now we define path convexity over closed sets of the configuration space. Let S be an optimal steering method on a metric space $\mathcal{C}$. Then a set $X \in \mathcal{C}$ is path convex under S if all curves in S lie entirely within X [1]. To illustrate simple cases in which path convexity does and does not hold, consider the solid hemisphere example given by [1], shown in Figure 1.

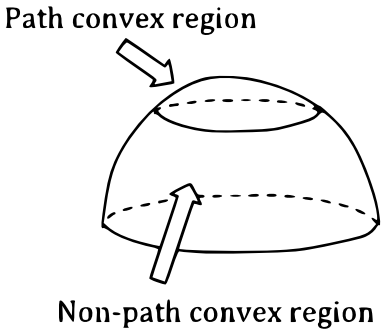


Figure 1: Consider the points on the curved surface of a solid hemisphere. With Euclidean distance as the metric, shortest paths between points close to the equator leave the curved region, taking shortcuts via the flat base. Hence, the curved region is not path convex. Now consider a smaller set of the curved region of the hemisphere: all points within 15 degrees of the pole. This set is path convex because using the bottom half of the hemisphere does not provide a viable shortcut.

Non-path convex regions occur for enough robots that it makes sense to define a softer convexity requirement. Hence, path subconvexity is defined as follows: let S be an optimal steering method on a metric space $\mathcal{C}$. Then a set $X \in \mathcal{C}$ is subconvex to $Y \in \mathcal{C}$ if all curves in S lie within Y. In the scenario provided above, the curved region of the solid hemisphere is path subconvex to the entire surface of the hemisphere, including the flat base.

Consider a metric ball with radius r centered at $x \in \mathcal{C}$ according to some metric d, notated as $B_r^d[x]$. The following theorem, stated and proved by Balkcom *et al.*, states that a ball subconvex to $B_r^d[x]$ can always be found.

Theorem: Let S be an optimal steering method under metric d on metric space $\mathcal{C}$. Furthermore let $x \in \mathcal{C}$. Then $B^d_{r/2}[x]$ is subconvex to $B^d_r[x]$.

These metric balls are central to the decomposition algorithm.

2.2 Optimal Path Guarantees with $\mathcal{C}_\epsilon$ Coverage

Suppose we cover C_ϵ with cells that are path subconvex to the free space. Further suppose that the cells overlap only at their boundaries. Then Balkcom *et al.* provide a method to find an optimal path that lies within that set of cells. Formally, the statement is as follows:

Theorem: Let $\mathcal{C}_\epsilon$ be covered with cells that are path subconvex to the free space. Assume the local metric is continuous at the boundaries of the metric cells. Furthermore, choose any two points that lie within the space covered by union of the metric balls, p_{start} and p_{end}. Then there exists an optimal path described by a non-repeating sequence of cells $c_1, c_2, ..., c_n$ and one point within each cell $p_1, p_2, ..., p_n$ (except for the first and last cell which additionally contains the starting and ending points). This optimal trajectory is given by $p_{start}, p_1, p_2, p_3, ..., p_n, p_{end}$.

Suppose we densely sample points within cells and on cell boundaries to create a roadmap. Then this theorem allows us to prune the optimal path search.

2.3 Requirements to Guarantee $\mathcal{C}_\epsilon$ Coverage

The previous sections yield a method for finding optimal motion plans over $\mathcal{C}_\epsilon$. However, algorithms for obtaining a tiling of $\mathcal{C}_\epsilon$ are not immediately obvious. The following theorem proves ACD algorithms can obtain $\mathcal{C}_\epsilon$ coverage with finite sampling.

Theorem: Let $e(q)$ be the minimum Euclidean distance between a robot in configuration q and a workspace obstacle. Let v_{max} be the maximum speed of any point on the robot over all configurations of the robot. Furthermore let the function $h(e(q))$ yield a radius for a hypercube subconvex to $B^d_{e(q)/v_{max}}[q]$. For any $\epsilon > 0$ and hypercube cell H with center q and radius s, H contains no points within $\mathcal{C}_\epsilon$ if $e(q) < \epsilon/2$ and $s < h(\epsilon/2)$.

In an approximate cell decomposition approach, this theorem tells us mixed cells that satisfy the two conditions do not need to be further decomposed to provide $\mathcal{C}_\epsilon$ coverage.

3 Constructing Path Subconvex Balls

Here we present an algorithm to compute cells that are path subconvex to the free space. In short, workspace information is used to compute a lower bound on minimum time to collision given a particular configuration. Minimum time to collision then yields conservative estimates of maximum allowable deviations of configuration space parameters given maximum parameter speeds. Applying the theorem from Section 2.1, we divide the radius of this reachable set by two and place a hypercube within this path subconvex region. Note that a conservative radius for the metric balls is calculated so that no ball intersects with the collision space.

Time to collision for a particular configuration q can be lower bounded with two pieces of information: $e(q)$ and v_{max} (defined in Section 2.3). A lower bound on minimum time to

collision is $t_{col} = e(q)/v_{max}$. As Balkcom *et al.* [1] note, finding a constant v_{max} upper bound requires that the robot obeys Lipschitz continuity. In short, this means that the maximum workspace velocity of all points constituting the robot over all configurations is bounded by a constant and the rate of change in any path in the configuration space (parameterized by the chosen metric) is bounded by a constant.

3.1 Case Study: Robotic Arm

A planar robot arm example is provided to demonstrate the algorithms in this paper and also give experimental results.

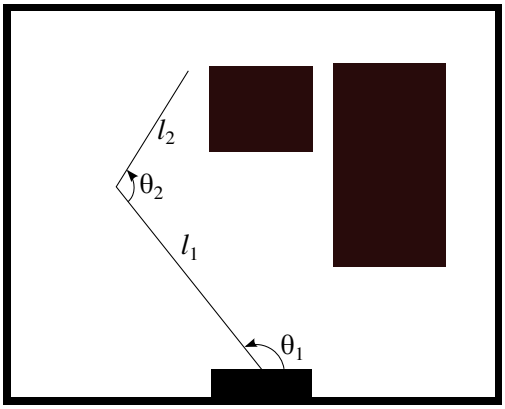


Figure 2: Two joint robot arm with two rectangular obstacles in a two-dimensional world. Note: this diagram only shows notation. The actual scenarios tested are shown in Figure 6.

Consider a polygonal chain robot arm with n segments and some number of obstacles in the workspace. Denote the length of each segment i with l_i, and let each joint j_i on the robot correspond to the far endpoint of the ith segment. Furthermore, let the angle of the ith joint (with respect to the previous joint's angle) be denoted as θ_i, with each joint's maximum angular velocity denoted by ω_i. Then the position of the ith joint

$$(x_{j_i}, y_{j_i}) = (\sum_{k=1}^{i}[l_k cos(\theta_{s_i})], \sum_{k=1}^{i}[l_k sin(\theta_{s_i})])$$

where $\theta_{s_i} = \sum_{m=1}^{k} \theta_m$.
Fixing a robot arm's dimensions, the configuration of the arm is captured by $\langle \theta_1, \theta_2, \theta_3, ..., \theta_n \rangle$.

Because the range of motion of this arm within a specific time interval is expensive to compute deterministically, the $e(q)/v_{max}$ naive bound is a starting point for ball construction. In this scenario, v_{max} can be computed by assuming the arm is maximally extended with all joints rotating in the same direction. We then have that

$$v_{max} \leq \sum_{i=1}^{n}[l_i \sum_{m=1}^{i} \omega_m]$$

The lower bound on time to collision is $e(q)/v_{max}$.

4 $\mathcal{C}_\epsilon$ Coverage Algorithms

4.1 Cell Decomposition

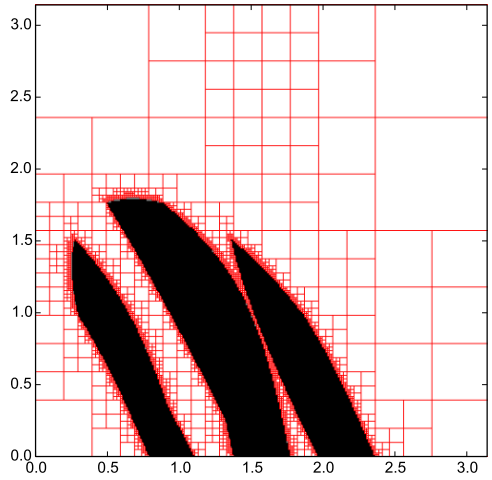


Figure 3: An example of using a hierarchical decomposition approach for a two chain planar arm robot. The black represents obstacle space, scanned using brute force techniques. The white boxes outlined in red are free space cells. The gray, present in the small gap between the collision space and the free cells, represents mixed cells in which decomposition halted.

As given by Section 2.3, only cells with center q such that $e(q) > \epsilon/2$ or radius $s > h(\epsilon/2)$ must be considered to achieve $\mathcal{C}_\epsilon$ coverage. Like many naive cell decomposition approaches, we could simply tile the space with hypercubes of side length $h(\epsilon/2)$. However, a hierarchical decomposition approach would use fewer cells by placing larger cells in safer areas of the configuration space (shown in Figure 4). Hence, a hierarchical decomposition algorithm is given in Algorithm 1. This hierarchical algorithm leads to decompositions like in Figure 3.

Hierarchical cell decomposition was selected to cover the space instead of non-ACD algorithms for two main reasons. First, other sampling techniques tested (grid, uniform random, and medial axis sampling) tend to create cells that overlap in regions far from obstacles. These other sampling methods also struggle to cover areas close to obstacles. These phenomena can be seen in Figure 5. Hierarchical sampling allows for large, non-overlapping cells in regions far from obstacles and also ensures coverage of areas with smaller boxes. Secondly, the boundaries of cells align in hierarchical decomposition, allowing us to use the main result from Section 2.2. We can run a pruned pathfinding algorithm with a roadmap consisting of samples on and inside cell boundaries.

4.2 Experimental Results

The scenarios tested are shown in Figure 6. As shown in Table 1, the number of cells and time required for the hierarchical decomposition to complete increases rapidly with

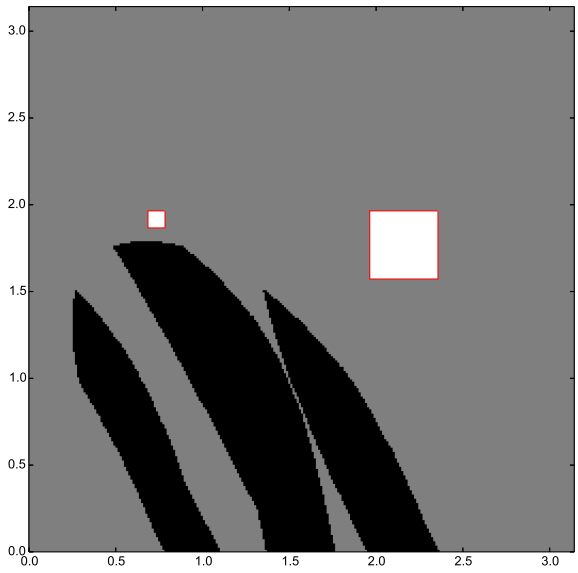


Figure 4: Configuration space for a two joint arm in the $\theta_1\theta_2$ space. Cells (displayed in white) placed closer to obstacles (in black) tend to be smaller than those far away from configuration space obstacles.

dimension. Decreasing ϵ also increases time and memory costs. Notice that for two different values of ϵ in the 3-joint arm, the decomposition is the same. This is because $\epsilon = 0.3$ and $\epsilon = 0.4$ require descending to the same level in the decomposition tree. Figure 7 shows the graph algorithm and resulting path for the 2-joint arm example.

5 Algorithm Adaptations

5.1 Techniques to Increase Cell Size

Sometimes it may be beneficial to invest additional computing power to increase the radius of the metric balls. The simplest idea is to merge neighboring cells if all are free. Another idea is to increase the size of the safe cells. The safe radius computed in Section 3 is very conservative, and in many cases, the balls can be made significantly bigger. While this safe radius can be adapted on a robot-by-robot basis, the following is an iterative approach to expanding cells in the general case.

The crux of the algorithm is performing a binary search for the maximal lower bound on time to collision, given the the $e(q)/v_{max}$ bound construction (see Algorithm 2). The lower limit of the search range is the naive $e(q)/v_{max}$ value, and the upper limit can be any reasonable value. For example, in a planar robot, the upper limit of the range could be set to the time it takes the slowest joint to make a full revolution. At each step in the binary search, the current t_{col} is used to recompute v'_{max}, the fastest moving point on the robot

Algorithm 1: Hierarchical Decomposition Algorithm

input : configuration space $\mathcal{C}$ with dimension n, error tolerance ϵ
output: hierarchical decomposition tree
$root \leftarrow$ tree node representing $\mathcal{C}$;
$toDecompose \leftarrow$ queue($root$);
while *toDecompose is not empty* **do**
  $currentNode \leftarrow toDecompose.dequeue()$;
  $q \leftarrow$ center of $currentNode$;
  if $currentNode \subseteq B^d_{r/2}[q]$ **then**
    Mark $currentNode$ as free;
  else if $e(q) < \epsilon/2$ *and* $currentNode$ *radius* $s < h(\epsilon/2)$ **then**
    Mark as mixed, do not decompose
  else
    $children \leftarrow 2^n$ children tree nodes of $currentNode$;
    $toDecompose.enqueue(children)$;
return $root$;

Joints	Epsilon	Tree Nodes	Cells	Time (s)
2	0.01	1.0×10^6	17,412	2
2	0.05	$< 1 \times 10^6$	4,230	.2
2	0.1	$< 1 \times 10^6$	2,076	.2
3	0.1	2.8×10^8	3.9×10^6	489
3	0.2	4.1×10^7	9.2×10^5	93
3	0.3	6.0×10^6	2.2×10^5	26
3	0.4	6.0×10^6	2.2×10^5	26
4	0.2	1.1×10^9	9.5×10^7	1871
4	0.4	1.1×10^8	1.3×10^7	365

Table 1: Decompositions for the 2, 3, and 4-joint arms at various error tolerances.

over all configurations reachable in t_{col}. Using the current value of $e(q)/v'_{max}$, we determine if the robot is definitely safe or there could be a collision. In the former case, the time to collision estimate is increased. In the latter, the time to collision is decreased. The algorithm continues until some threshold in search range is met.

The reason this algorithm works is that maximum velocity can be better bounded given a configuration and a maximum amount of time to move. For example, in a robot arm, suppose that the arm is folded in on itself (see Figure 8). Then if the time to collision is estimated to be small, the arm cannot fully extend. Instead of using the maximum velocity of the end effector when outstretched, which is the maximum velocity of the robot arm over all configurations, we instead only have to compute the maximum velocity of the maximum extent of the arm given some small amount of time.

Another framework for increasing ball size is to leverage robot motion constraints. In the binary search algorithm, v_{max} is an upper bound on the maximum velocity over all points on the robot over all configurations reachable in the time allotted. Using system-specific

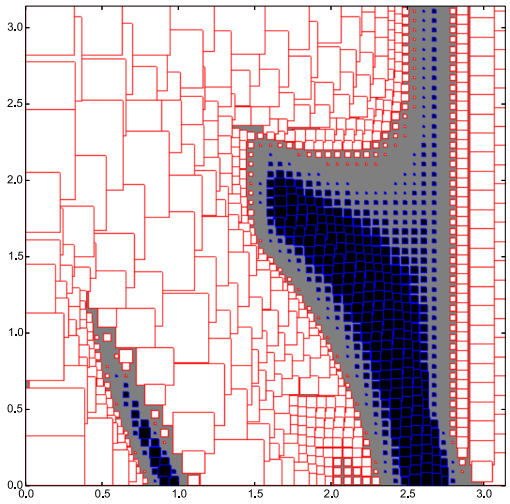


Figure 5: Grid sampling tends to oversample very safe areas of the configuration space and undersample areas closer to obstacles. Free space boxes are shown in white and collision boxes are shown in black. Uncovered areas are shown in grey.

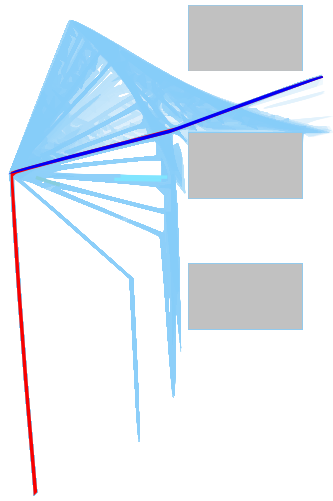
robot movement constraints, the time to collision bound can be further improved. The best time to collision estimate using the $e(q)/v_{max}$ formulation is calculating time to collision for each point on the robot with information about the maximum velocity vectors and obstacle collision distances in all directions. Though this is not computationally feasible in most cases, it may be feasible to group sets of points together to calculate this information for a tighter minimum time to collision bound. For example, splitting the planar arm into small sections and calculating the minimum time to collision for each piece of the arm can yield bigger boxes.

Both strategies were undertaken to improve the minimum time to collision lower bound for the robot arm. Calculating the maximum extent of an arbitrary length arm given a current configuration and a set of constraints is not an analytically solvable problem, as mentioned in [2]. However, the following approach was used to obtain an upper bound on maximal extent given a time to collision estimate (see experimental results in Table 2):

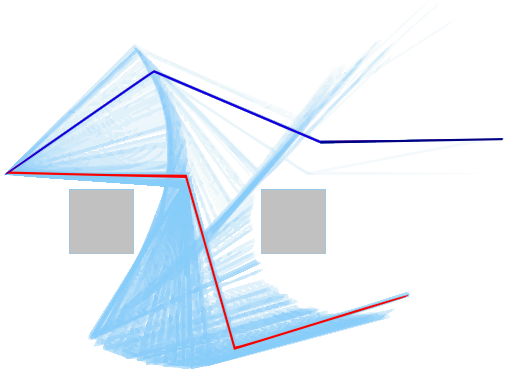
1. Calculate the maximum length of each consecutive pair of segments given a time to collision estimate. Let n be the number of segments in the robot. Consider the robot as a collection of $ceil$(n/2) segments where each segment in the new arm is the length of the maximum extent of the pair of segments it represents.

2. Using the new arm segment lengths from Step 1, calculate $e(q)$ and v_{max} for the first segment, then the union of the first and second segment, then the union of the first through third segments, and so on. In effect, we consider the arm robot as a collection of arm robots with different numbers of segments. This provides a coarse method of localizing maximum velocity and closest collision to gain a better time to collision lower bound.

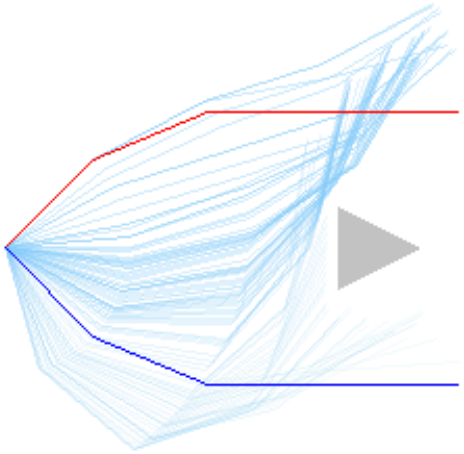
3. If any subsection of the arm's $e(q)$ and v_{max} values result in possible collision, reduce the time to collision. If all arm subsections are safe, then increase the time to collision.



(a) 2R Arm



(b) 3R Arm



(c) 3R Arm

Figure 6: These diagrams depict the arm scenarios tested. The starting configuration is given in dark blue. The end goal configuration is given in red. The arm is anchored on the left side. The gray polygons are obstacles. The path is shown by the light blue lines. The amount of change between each blue line is proportional to the sidelength of the box.

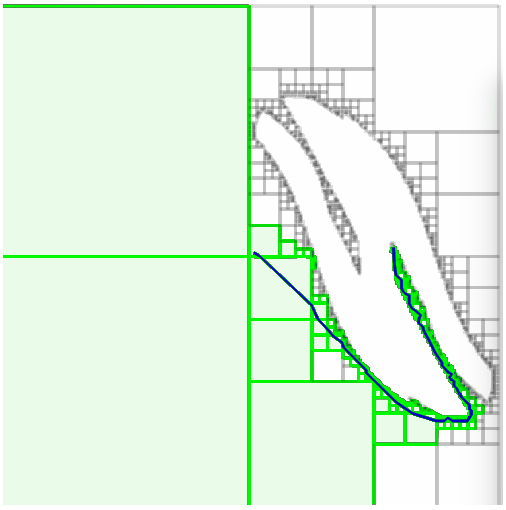


Figure 7: The configuration space and path for the 2-joint arm scenario given in Figure 6. The boxes outlined in green depict cells that were searched during A* to find the path shown in blue.

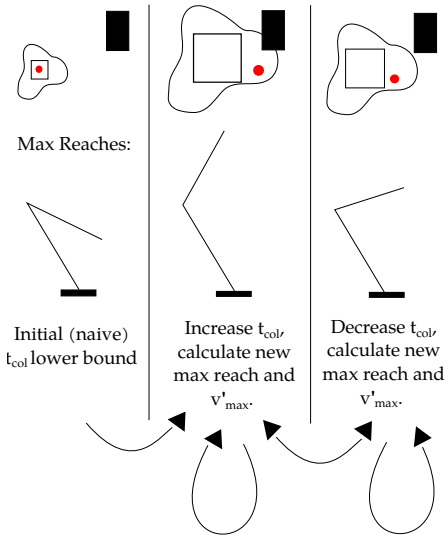


Figure 8: A diagram for the binary search algorithm on t_{col}. The red point demarcates the configuration with the highest v'_{max} upper bound. The closed curved region is the actual reachable set of configurations with the given estimate for t_{col}. The black rectangle is a configuration space obstacle. The square outlined in black represents the cell sample we construct. We increase and decrease t_{col} by decreasing amounts until some threshold is met. We move to the third state if $e(q)/v'_{max} < t_{col}$ and move to the second state otherwise.

Algorithm 2: Binary search method for time to collision computation

input : configuration space $\mathcal{C}$, q, tol (error tolerance in binary search)
output: improved lower bound for time to collision
$lower \leftarrow e(q)/v_{max}$;
$upper \leftarrow$ maximum time to collision;
while $upper - lower > tol$ **do**
  $m \leftarrow \frac{lower+upper}{2}$;
  $v'_{max} \leftarrow$ upper bound of v_{max} in $B^d_m[q]$;
  if $v'_{max} \times m > e(q)$ **then**
    $upper \leftarrow m$
  else
    $lower \leftarrow m$
return $lower$;

Joints	Epsilon	Iterative?	Cells	Time (s)
2	0.01	No	30,899	2
2	0.01	Yes	30,180	2
2	0.05	No	8,283	0.2
2	0.05	Yes	7,578	0.2
2	0.1	No	4,466	0.2
2	0.1	Yes	3,774	0.2
3	0.2	No	4,947,292	94
3	0.2	Yes	4,841,676	93
3	0.4	No	1,459,932	27
3	0.4	Yes	1,386,996	26

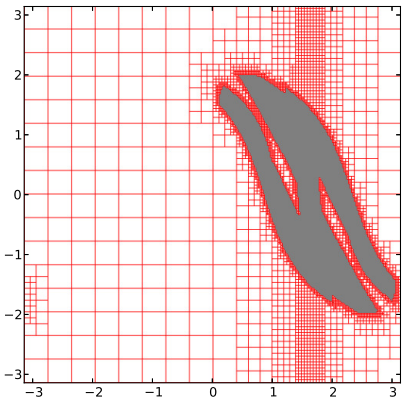
Table 2: Relative performance of decomposition with and without iterative widening using the methods mentioned in Section 5.1. Note: merging cells has been disabled.

Experimentation suggests that this iterative approach provides some gains, as seen in Table 2 and Figure 9).

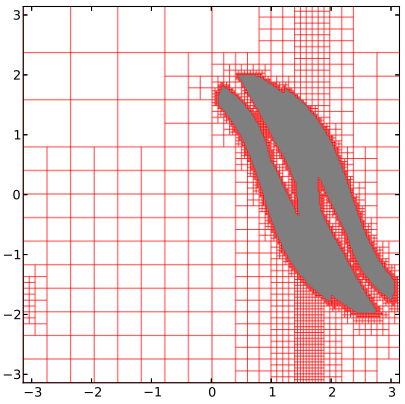
5.2 Distributing Work

Decomposition must only occur once for a particular robot and a set of static obstacles. Thus, it is sometimes reasonable to expend a large amount of computing power for decomposition. Hierarchical decomposition is an embarrassingly parallel task, but collecting the information back for optimal path searches is more challenging.

In this paper, we examine using MPI for distributing work. Let there be one master node and n worker nodes. The master node begins splitting the configuration space until there are at least n mixed cells to be further decomposed by the n worker nodes. The master node then sends cells that need to be further decomposed to the worker nodes. The worker nodes take over this decomposition until completion. The results from this model are shown in Table 3. In cases where configuration space complexity is not uniform, some nodes terminate



(a) No widening applied



(b) Iterative widening applied

Figure 9: A side by side comparison of the boxes yielded from iterative widening. Because of the tree approach, many cells stay the same size, but there are improvements in the upper quadrants after widening.

Joints	Epsilon	MPI?	Decomposition Time (s)	Total Time (s)
2	0.002	Yes	70	161
2	0.002	No	159	159
2	0.003	Yes	17	18
2	0.003	No	33	33
2	0.005	Yes	5	5
2	0.005	No	8	8

Table 3: Experimental results using MPI without work stealing. The total time includes both decomposition time and the time for the master to collect all nodes.

quickly and a few others work for much longer periods of time. In these cases, work stealing may be beneficial.

Because shared memory is limited when using multiple cores, it is not a viable option for large decomposition tasks. Communicating large volumes of fully decomposed cells from workers to the master node sometimes makes sense. However, in cases with large numbers of workers each with large numbers of cells to transmit to the master node, the master node may take long periods of time to collect all information. In these scenarios, it makes sense to send back summarized information or distribute pathfinding.

6 Conclusions

This paper detailed an algorithm to decompose the configuration space into cells that are path subconvex to the free space. Configuration space cells are constructed via a time to collision lower bound obtained using workspace information. Theoretical results proven by Balkcom *et al.* [1] show that because these cells are path subconvex to the free space, motion

plans over this space are guaranteed to be safe.

This framework for optimal motion planning is easily customizable to a variety of scenarios. There are many generalized and robot-specific algorithmic choices left to the user's discretion, as highlighted in the arm robot case study. Iterative approaches to expanding cells may be useful, especially when the lower bounds computed are very conservative. In addition, leveraging system-specific v_{max} and locality of maximum velocity increases cell sizes.

Though decomposition is time and memory intensive, it is only required once per a pairing of robot and a set of static workspace obstacles. However, it is untested as to how feasible these decomposition methods can be made in robots with over four degrees of freedom.

Current work seeks to improve time and memory costs of this approach. Path subconvexity approximation could largely reduce the number of cells necessary to tile the space. In this approach, we identify regions of the space well approximated by the local planner instead of guaranteeing subconvexity in these regions. The regions can be obtained via sampling points within an undecomposed cell, connecting them with the local planner, and then estimating whether the lowest cost paths within the cell are close to the local planner distance estimates, with some degree of error tolerance.

A second potential time improvement comes from creating a better A* search algorithm. Roadmap nodes are placed on the boundaries of cells. However, since the number of boundaries increases rapidly as the number of dimensions increases, the search uses many samples, and thus a lot of time and memory. A possible improvement is to search for paths over a constant (small) number of samples per boundary, then iteratively shorten. The shortening procedure may impose the constraint that edges used in the original path must be used in the final path. In this scenario, additional samples can be placed along used boundaries, and the Viterbi algorithm can be used to get a shorter path along the chosen cell boundaries.

In addition, one of the most time-intensive pieces of the algorithm is calculating minimum distances to obstacles from the robot. For each ball, at least one distance calculation has to be performed. For the binary search for cell radius size, additional distance calculations must be carried out. Improving collision detection time using precomputation or other clever tricks could dramatically speed up decomposition. In addition, more work should focus on improving the construction of the time to collision bound, since this impacts both the computation costs and memory costs of the decomposition and path planning algorithms.

Finally, work in obtaining diverse sets of paths around obstacles and paths with margin would follow naturally from the configuration space information collected by this decomposition algorithm. It would be an interesting experiment to incorporate obstacle growth as in [5] to handle path planning with moving obstacles.

7 Acknowledgments

First, I would like to thank Professor Balkcom. His support and guidance were invaluable, and his ideas for path subconvexity made this paper possible. Professor Balkcom and Weifu Wang proved the three theorems in Section 2. I would also like to thank Yu-Han Lyu and Yinan Zhang for their contributions. Yu-Han increased the speed of the decomposition algorithm, and Yinan coded the A* search and the graphics framework for Figure 6.

References

[1] Balkcom, Devin, Ajay Kannan, Yu-Han Lyu, Weifu Wang, Yinan Zhang. "Metric cells: towards complete search for optimal trajectories."

[2] Borcea, Ciprian, and Ileana Streinu. "Extremal Reaches in Polynomial Time." Proceedings of the 27th Annual Symposium on Computational Geometry (2011).

[3] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," International Journal of Robotics Research, vol. 30, no. 7, pp. 846–894, June 2011.

[4] Salzman, Oren, Michael Hemmer, Barak Raveh, and Dan Halperin. "Motion Planning via Manifold Samples." Algorithmica 67.4 (2013).

[5] Jur van den Berg, Sachin Patil, Ron Alterovitz. Motion planning under uncertainty using iterative local optimization in belief space. Int. Journal of Robotics Research, vol. 31(11), pp. 1263-1278, 2012.

[6] Zhang, Liangjun, Young Kim, and Dinesh Manocha. "A Hybrid Approach for Complete Motion Planning." Intelligent Robots and Systems (2007).

[7] D. Zhu and J. Latombe. "Constraint reformulation in a hierarchical path planner." Proceedings of International Conference on Robotics and Automation, pp. 1918–1923, 1990.