A New Path-Finding Algorithm Using Parallelepiped Decomposition

Edwin Olson
eolson@mit.edu

Abstract: A volume-decomposing path finding algorithm is developed. It uses parallelepiped decomposition, composing paths through the vertices of parallelepiped intersections. It exhibits completeness and low computational requirements while producing nearly optimal paths.

Disclaimer: This algorithm was developed as a solution to a problem set for a course at MIT. Its scope, therefore, is substantially less than that of a formal paper. For example, citations have been omitted, as have proofs (occasionally because a proof is missing.) Contact me for more information.

A New Path Finding Algorithm Using Parallelepiped Decomposition

Observations

The challenge in path finding is to find an algorithm that will yield a reasonable solution in a reasonable amount of time. The search space is ludicrously large and unconstrained; in 3D, a naïve approach would lead to a branch width of 6, one corresponding to every direction from every pixel in 3D space. The large distances required would result in an extraordinarily large search tree. We must find a way to reduce this search space *without losing generality/completeness* and *while producing nearly optimal solutions*.

World Modeling

One approach is to develop a model of the world as a set of convex, 3-dimensional regions containing no obstacles. The robot can move freely within any such region since it contains no obstacles and is convex. Further, the robot can move to any other region that intersects the region it is currently in.

We can apply a search algorithm to such an interconnection of region to yield a path from one point to another. We begin by finding a region containing the start point, and perform a search (such as A*) by examining all the regions that intersect with the start region. These paths are then extended by examining the regions that intersect with those regions, and so on. When a path ends in a region containing the end point, a correct path has been found.

Generating a list of convex regions, however, is also quite complicated. We approximate these convex regions by considering parallelepipeds. A parallelepiped is a 3-dimensional box. We choose these boxes in a way that optimizes the volume of the boxes, since a greater volume corresponds to a polygon connecting more points.

These boxes can be generated as follows: a random point in 3-space is generated. Then a box is formed by trying to grow each wall of the box in turn. For example, we try growing the box in the +x direction. Then in the +y direction. Then +theta, -x, -y, -theta. This ordering helps ensure that we grow the largest volume boxes.

A complete solution can be guaranteed by attempting to grow a box at every possible coordinate. The parallelepiped modeling works well for polygonal obstacles, however, it can generate a large number of boxes on irregularly shaped (or curved) surfaces when operating in this high-precision mode. However, most problems will not need this level of precision; random box generation as described above works very well.

The number of parallelepipeds can be pruned substantially, which can make complete solution finding much more practical. We can (obviously) prune duplicate parallelpipeds, a significant savings since many points will grow into the same box. Boxes of volume one can also be discarded, since they cannot connect two or more points and thus cannot

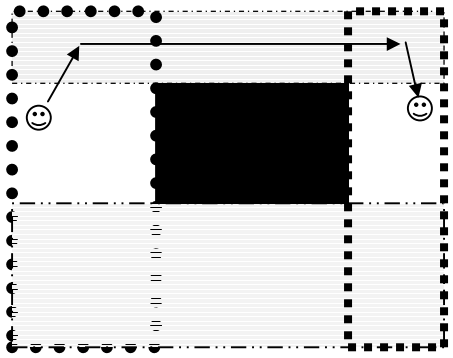
be part of a path. Boxes can also be eliminated if they do not contain the start or end point and connect to only one more box—that box is conceptually a dead end.

We can extend the dead-end concept further. Suppose that box B intersects a set S of other boxes. If each of these boxes also intersects a superset of S, then box B can be eliminated since it provides no new connectivity. Note that the simple dead-end case above is simply a special case of this rule since a box intersects itself, as is the case of a box of volume one.

The next step is to give every box a list of boxes that it intersects with. This will be necessary to perform the search. A *significant* advantage of using parallelepipeds versus arbitrary convex regions is that finding the intersections is trivial, and the intersection is just another parallelepiped.

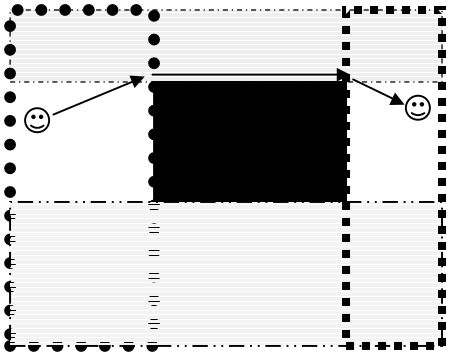
Searching for a Solution

A search can then be performed. For searches such as A* which require a "quality rating" for a path in order to find an optimal solution, it is necessary to pick a set of waypoints—discrete points through which the robot will travel. We will assume that the robot will follow the path formed by the lines connecting the waypoints. For a quick-and-dirty solution, one can set the waypoints to be the center of the intersections of the boxes through which the path travels.

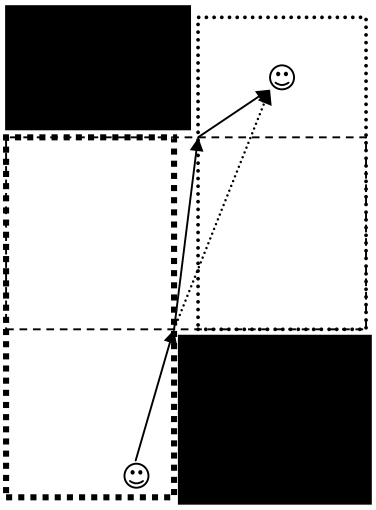


In the 2D space to the left, the black obstacle partitions the world into four maximally large boxes. A path is then plotted from the start point to the end point by traveling through the center of the intersections of the boxes. This results in a reasonable path.

It is clear that this path is not optimal. The path can be substantially optimized by increasing the number of waypoints considered while searching for the minimal length path from start to end. Another particularly good set of waypoints to consider are the vertices of the parallelepiped intersection of the boxes. In 3-space, this corresponds to 8 additional possible paths through every intersection, as opposed to the single path considered above. It is advantageous to retain the center point of the intesection for a total of 9 possible waypoints. This will generate nearly optimal solutions, but fails to generate the optimal path on certain configurations. However, the generated path and the optimal path differ by only a small amount.



The path formed by considering all the vertices of the intersections can be much better (optimal in this case.)



An illustration of a non-optimal solution (solid line) generated by the algorithm. The optimal solution is also plotted with a dashed line.

The author wrote a program that can use either method; a command line option enabled using all the vertices of the intersections in the search. Due to time constraints during implementation, a simple breadth-first search, rather than the more efficient A* method, is used.

These algorithms perform quite well; they find very good solutions in a reasonable amount of time. They can scale well to more complicate universes by adjusting the number of boxes used in the search (an sub optimal solution can often be found using a subset of all possible boxes), or by adjusting the number of waypoints used.

A solution found with a small number of waypoints can be locally optimized after the search, which can substantially decrease the distance of the path without additional searching. This algorithm is as follows: Pick a waypoint. Replace this waypoint with another point from the same intersection that reduces the total distance of the path. Loop until a no more reductions in total distance are possible. This algorithm terminates since the total distance is changing monotonically. In many (if not all) cases, this algorithm will change a sub optimal solution into the optimal solution. The author has not developed a proof.

Compared to naïve implementations of convex volume (as discussed in Embodied Intelligence, the MIT 6.836 text), this set of algorithms provides superior performance with minimal added cost. In fact, these algorithms provide the computational simplicity of convex volume decomposition with almost the performance of visibility graphs (which are optimal.) It can also compute paths more optimally than approximate cell decomposition.

Discussion of Real World Limitations

It is obvious that these algorithms will suffer when incomplete or noisy information is available. Incomplete information can result in unexpected collisions in all of the algorithms. Noisy information can make a visibility graph algorithm extremely inefficient as noise is perceived as additional vertices. For example, an additional pixel sticking out from a straight edge would result in 3 additional vertices. The algorithm discussed above is robust to this sort of noise since it depends on open channels only. An isolated noise value would, however, result in the unnecessary partitioning of boxes.

When incomplete information is available, additional care must be taken to maximize the possibility of finding a solution. Suppose that valid information is available only within a fixed radius of the robot. In this case, it might make sense to partition the world into overlapping regions. Set the regions to be small enough that the robot can reliably sense all of the obstacles in that region. Then, the robot can assess at any point in time which adjacent regions it would be able to navigate to. It picks the most promising one and plots a local path to that region. If that region turns out to be a dead end, then the robot should retreat to the previous region and select the next possible path. In other words, the robot is physically performing a hill-climbing depth-first search. With perfect navigational positioning and a static environment, this algorithm can be complete, but is not optimal.

If the world is dynamically changing, the problem of path finding becomes extremely difficult. Many of the path finding strategies developed here rely on building a world model and using that model for some period of time. However, if the world is dynamic, that model may become inaccurate, and a collision could result. Without some bounds on the rate at which the world can change, we cannot proceed safely (and in fact, something may run into *us*.) If the world is changing at some maximum rate, we will need to be able to rebuild our model at a corresponding rate, and limit our velocity to ensure that we do not collide with anything.

Future Directions

Proofs are needed for some of the arguments in this paper.

There is a tantalizing possibility of performing volume decompositions dynamically, rather than building the whole world model in advance. If the robot is currently in one box, one could grow new boxes from points within that box. This could lead to solutions which are good without the memory requirements of the algorithm presented here.