In ICAPS-09 Workshop on Bridging the Gap Between Task and Motion Planning (BTAMP)

A Comparison of Risk Sensitive Path Planning Methods for Aircraft Emergency Landing

Nicolas Meuleau[*] and **Christian Plaunt** and **David E. Smith** and **Tristan Smith**[†]
Intelligent Systems Division
NASA Ames Research Center
Moffet Field, California 94035-1000
{nicolas.f.meuleau, christian.j.plaunt, david.smith, tristan.b.smith}@nasa.gov

Abstract

Determining the best site to land a damaged aircraft presents some interesting challenges for standard path planning techniques. There are multiple possible locations to consider, the space is 3-dimensional with dynamics, the criteria for a good path is determined by overall *risk* rather than distance or time, and optimization really matters, since an improved path corresponds to greater expected survival rate. We have investigated a number of different path planning methods for solving this problem, including cell decomposition, visibility graphs, probabilistic road maps (PRMs), and local search techniques. In their pure form, none of these techniques have proven to be entirely satisfactory – some are too slow or unpredictable, some produce highly non-optimal paths or do not find certain types of paths, and some do not cope well with the dynamic constraints when controllability is limited. In the end, we are converging towards a hybrid technique that involves seeding a roadmap with a *layered visibility graph*, using PRM to extend that roadmap, and using local search to further optimize the resulting paths. We describe the techniques we have investigated, report on our experiments with these techniques, and discuss when and why various techniques were unsatisfactory.

1. Introduction

In two previous papers, we described a planning system designed to aid the pilot of a damaged aircraft choose an appropriate emergency landing site (Meuleau et al. 2008; 2009b). The system assumes that an onboard flight management system and diagnostic system provide information about:

- current position, heading, altitude and airspeed
- relevant weather information
- the current flight envelope of the aircraft (bank, climb/descent rate, and speed limitations)

The objective is to quickly produce a ranked list of emergency landing sites ordered by *risk*. Relevant factors in computing risk include:

- en route weather conditions such as thunderstorms, turbulence, and icing
- ceiling, visibility and wind conditions at the landing site
- runway length, width and surface condition[1]
- population density along the final approach path
- emergency facilities at the landing site (fire, rescue, and medical personnel and equipment)

The controllability of the aircraft along the different axes influences the importance of these different factors. For example, if an aircraft has limited ability to bank to the right, turbulence becomes increasingly dangerous, and the aircraft may not be able to cope with a right crosswind on approach and landing. This in turn influences the choice of runway and the path to get there.

This problem can be mapped into the following search/planning problem: for each landing site within range, find a minimal risk path to the landing site, and evaluate the risk of the path and landing site combination. The results are then ordered according to total risk. There are various heuristics that can be used to improve the efficiency of this search, such as ordering the candidate landing sites according to distance and landing risk. However, the central step in this process is finding a minimal risk path to each candidate landing site. Fundamentally, this is a 3D path planning problem. There has been extensive previous work on path planning and obstacle avoidance to address similar problems. See (Choset et al. 2004) for a survey of this field. However, a characteristic of the emergency landing planning (ELP) problem is that it gathers several difficulties that are rarely encountered simultaneously:

- There are multiple possible goals (landing sites) to consider
- As human lives are at stake, we are not interested in just any feasible path, but in trajectories that are as close as possible to optimal
- Weather obstacles are traversible, but flying through them incurs a higher risk. Consequently, the cost of a path cannot be summarized by its length or duration – the nature of the “terrain” traversed must also be taken into account

[*] Carnegie Mellon University
[†] Mission Critical Technologies

[1] We consider only runways here, but the same criteria apply to possible off airport landing sites such as roads, fields, beaches and bodies of water.

- The aircraft dynamic constraints play a very important role in determining whether a path is flyable or not. This contrasts strongly with many robotic path planning problems where every robot move is reversible. Because damaged aircraft may have very reduced controllability, dynamic constraints are particularly important
- Optimization is constrained: the aircraft may have a limited range or limited time and must land before exhausting either of these non-replenishable resources

The problem of optimal path planning among a set of soft polygonal (or polyhedral) obstacles with different costs per step is known as the *weighted region* problem. It has been studied extensively since the end of the 80's (see for instance (Mitchell and Papadimitriou 1991) and (Rowe and Alexander 2000)). However, this work is largely theoretical and, to our knowledge, has not lead to a practical solution for 3D Euclidean spaces of the size of our problem instances.

We have considered and investigated a number of more practical path planning methods for solving this problem, including cell decomposition, visibility graphs, probabilistic road maps (PRMs), and local search techniques. In their pure form, none of these techniques have proven to be entirely satisfactory – some are too slow or unpredictable, some produce highly non-optimal paths or do not find certain types of paths, and some do not cope well with the dynamic constraints when controllability is limited. In the end, we are converging towards a hybrid technique that involves seeding a roadmap with a *layered visibility graph*, using PRM techniques to extend that roadmap, and using local search to further optimize the resulting paths. In this paper, we describe the techniques and combinations of techniques we have investigated, report on some experiments with these techniques, and discuss when and why various techniques were unsatisfactory.

2. Path Planning

We distinguish between two different kinds of algorithms: *path planners* (in the strict sense) generate candidate solutions (paths) based on problem data (aircraft state, target and obstacles). *Local search* algorithms use the path produced by another component (path planner or local search) as a seed to try to produce an improved path.

Another important distinction is between algorithms that take into account the aircraft dynamics and those that do not. The later neglect variables that are important in determining the flyability of a path (aircraft heading and speed).[2] They work in 3D Euclidean space: aircraft states are represented as triples (latitude, longitude, altitude), and paths between states are straight line segments. As a result, the paths produced by these algorithms are not necessarily flyable, and need to be post-processed by a path planner that accounts for the aircraft dynamics and turn radius. The other class of algorithms works in 4D (Euclidian space plus heading) and attempts to produce paths that do respect the dynamic constraints, and can therefore be used without post-processing.[3]

Figure 1 shows the way in which various possible algorithms can be combined for this problem. Problem data can be fed into a path planner with or without dynamics, and the resulting path can be fed through local search algorithms with or without dynamics. The Trajectory Planner [4] shown in Fig. 1 takes as input a raw 3D path that does not account for aircraft dynamics, and outputs a flyable 4D path that goes through the same sequence of waypoints, but respects heading and speed constraints. It does so by essentially constructing Dubins paths between the waypoints it is given. The Trajectory Planner is for our purposes a black box that is essentially part of the aircraft flight management system. It has a richer model of aircraft control, and is tailored to the way in which flight plans are represented and interpreted by pilots. Note that the path produced by the Trajectory Planner is always at least as long as the input path without dynamics, but could be considerably longer, so it may exhaust a resource even if the seed path does not.

2.1 Path Planners

All of our path planners construct and search over a *roadmap*. A roadmap is a topological representation of the environment that captures the connectivity of the free space. Formally, it is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where vertices $v \in \mathcal{V}$ represent specific locations in the environment, including the starting aircraft location (v_0) and the targeted destination (v_g), and edges $e \in \mathcal{E}$ represent possible transition between neighboring locations. Path planners typically work by first building a roadmap, and then finding the shortest path in the roadmap between the aircraft and the target.

In all our algorithms, search is performed by the A* algorithm. This guarantees that optimal solutions are found within each given roadmap. The heuristic value of a state is obtained by computing the Euclidean distance to the target and assuming this distance has to be flown in clear weather. It constitutes an admissible heuristic with a sufficient informativeness to prune a consequent portion of the search space. If there is a maximum distance or time before landing, A* search nodes must be augmented with a state variable representing the remaining level of each limited resource. That is, two search nodes are considered equal if they represent the same state of the aircraft (Euclidean or configuration state), and if they have the same level of remaining resources (distance and time). This is necessary to guarantee that the search graph below a node depends only on the description of this node. It impacts the complexity of the algorithm both negatively (by increasing the number of search states) and positively (by limiting the search to trajectories within the resource bounds).

Our path planners differ on the nature of the roadmap they

[2]One way to compensate partially for ignoring dynamics is to expand each obstacle by an amount determined by the aircraft controllability. This allows sufficient space for the aircraft to be able to turn around the obstacle.

[3]We currently treat airspeed as constant. Including airspeed makes the space 5D.

[4]In the path planning literature "trajectory" is commonly used to refer to a time-paramaterized version of a path. We do not imply that here, although the two are equivalent given aircraft speed, since the initial time is fixed.

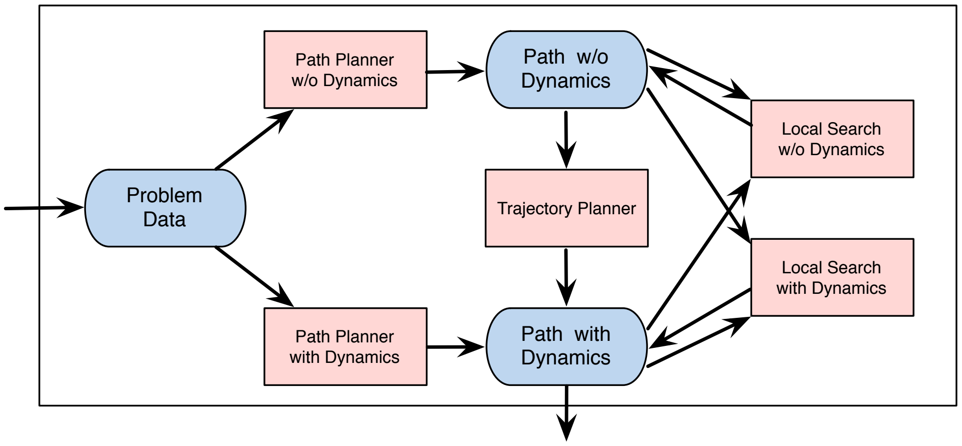
Path Planner w/o Dynamics
Path w/o Dynamics
Local Search w/o Dynamics
Problem Data
Trajectory Planner
Local Search with Dynamics
Path Planner with Dynamics
Path with Dynamics

Figure 1: Combining path planning algorithms.

use. A selection of approaches that we have investigated is presented below. We first present the algorithms that ignore the aircrafts dynamics, then we show how some of them can be augmented to respect the aircrafts dynamic constraints

Visibility Graph: One of the earliest and most common roadmaps is called the *visibility graph* (VG). This graph is defined in two-dimensional space. For our domain, obstacles are naturally columnar in nature (thunderstorms, icing regions, etc.) and are therefore represented as 2D polygons with an associated floor, ceiling and risk. It therefore makes sense to consider a 2D projection of the space. For the purpose of building the visibility graph, we consider all obstacles to be untraversable (risk = 1), we ignore the floors and ceilings of the obstacle and use only their two-dimensional polygonal representation. The nodes $\mathcal{V}$ of the visibility graph include: the start location v_0, the possible destinations v_g, and all the obstacle vertices (corners between two edges of the polygons). The edges $\mathcal{E}$ are straight lines between vertices that do not traverse any obstacle. In 2D, the visibility graph is guaranteed to contain the shortest path from the start to the goal. Unfortunately, this property does not hold if the same approach is applied in higher dimensions, or if some of the obstacles are traversable.

The *reduced visibility graph* or *tangent graph* is a subgraph of the visibility graph that is also guaranteed to contain the shortest 2D path. Because it contains fewer edges, it is easier to solve. It is based on the observation that the shortest path in the standard visibility graph traverses only edges that are tangent to obstacles. Therefore, non-tangent edges can be safely removed from $\mathcal{E}$ (see Fig. 2). Determining the set of tangent edges can be a difficult problem (Liu and Arimoto 2004). Instead of computing this set exactly, we eliminate edges whose extremities are not “locally tangent” to an obstacle. Consider an edge e incident to vertex v, between two sides s and s' of the polygonal obstacle. Then, e is locally tangent in v if s and s' fall on the same side on the straight line passing through e. A tangent edge may not contain an extremity that is not locally tangent (but the converse is not true). Therefore, we can safely eliminate edges with an extremity that is not locally tangent. This eliminates fewer edges than the real tangent graph; however, since testing for local tangency is very cheap, it is a good overall compromise (Cormen et al. 2001).

When a 2D solution is found, altitude is linearly interpolated along this path (from the current altitude to the target altitude). The resulting path does not account for the aircrafts dynamic constraints, and so it must be transformed by the path planner or a local search algorithm. This path is often of poor quality, because the possibility to fly above, below or through an obstacle is omitted. However, local search algorithms may sometimes transform it into an interesting path, allowing the traversal of soft obstacles. Using the visibility graph in our simulation is a way to measure how much work the local search algorithms can do when they are provided a low quality initial path.

Layered Visibility Graph: So far, we have limited the discussion to a 2D framework. In our emergency landing domain, obstacles have a floor and a ceiling, and it is sometimes possible to fly above, through or below some of them. The *layered visibility graph* (LVG) is an extension of the visibility graph to account for some opportunities of 3D movements.

An LVG waypoint is a 3D Euclidean waypoint. From each two-dimensional vertex in the visibility graph, we create a set of vertices in the LVG by varying the altitude.We create vertices at the ceiling and floor of the obstacle, at the aircraft’s starting altitude (if it is between the the obstacle’s ceiling and floor) and at each increment of 10000 feet between the ceiling and floor of the obstacle.

We add an edge to our LVG between any two vertices for which the corresponding vertices in the 2D visibility graph were connected. In addition, we also add: (i) an edge between the start and goal; (ii) extra edges so that the start and end vertices are connected to their 100 nearest neighbors. Notice that these edges may go above, through or below an obstacle.

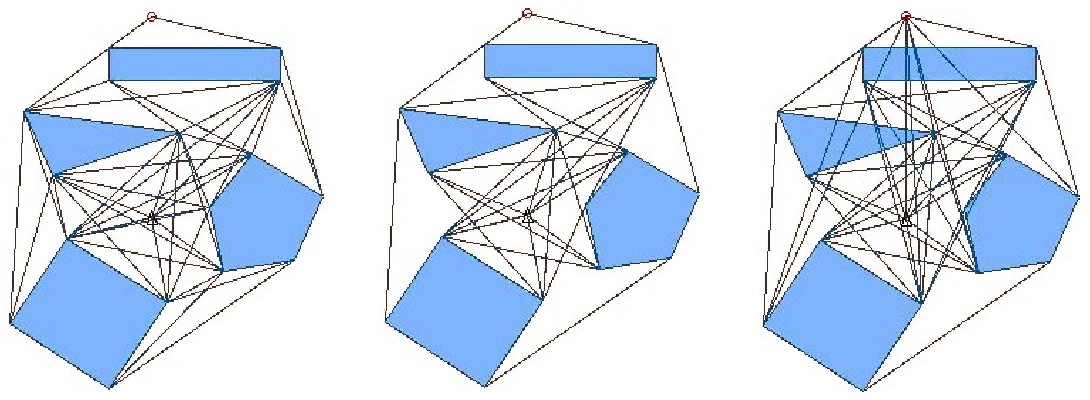

Figure 2: Three types of roadmaps (from left to right): visibility graph (58 edges), tangent graph (45 edges), and hybrid roadmap (69 edges). The aircraft is represented by the small triangle in the center of the figure, and the targeted landing site is the small circle at the top of the figure.

The LVG inherits some of its properties from the visibility graph. Because the roadmap contains edges along the sides of obstacles, the search algorithm is able to find near-optimal ways to go around obstacles. The roadmap also allows some movements above, through or below obstacles. However: (i) this is limited to a restricted set of line segments (the edges of the LVG); (ii) there is no attempt to optimize the vertical dimension along these edges. For instance, if the line segment from the aircraft to the target intersects an untraversable obstacle, the algorithm will make no attempt at passing above or below this obstacle. In other words, the algorithm does not try to minimize the set of obstacle traversed by varying the aircraft altitude. Nevertheless, this limited set of edges is sometimes sufficient to exhibit qualitatively interesting trajectories that local search algorithms can optimize.

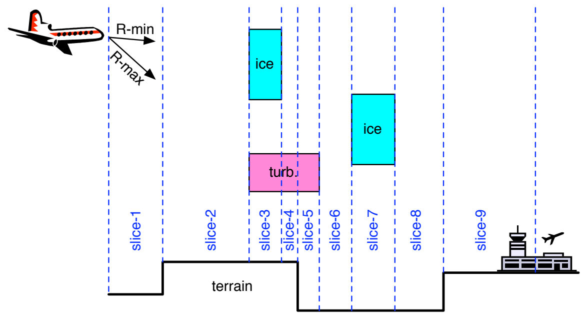

Figure 3: Cut of the 3D space along the segment from the aircraft to the targeted landing site. The cut is divided into 9 slices where ground elevation and obstacles are constant.

Hybrid Roadmap: The *hybrid roadmap* (HRM) is a more sophisticated attempt at deriving a 3D path planning algorithm from the 2D visibility graph. It exploits the fact that obstacles are columnar in nature and represented as 2D polygons with a floor and a ceiling.

The HRM is called “hybrid” because it contains states $s = (v, h)$, where v is a vertex of the graph and $h \in \mathbb{R}$ is real-valued altitude. Being in state (v, h) represents being at location v and altitude h. To connect two vertices of the HRM, we enumerate all obstacles that intersect the straight line segment between these vertices and all ground-level variations along this segment. As shown in Fig. 3, this path is divided into a series of *slices* inside of which the ground level is constant and the same set of obstacles is traversed. An edge in the HRM is a complex data structure that represents a cut through the 3D space as such a sequence of slices. There is one such edge connecting:

- The start and goal locations;
- The start or goal location and any corner of an obstacle, if the segment does not pass through this obstacle and does not have a non-locally tangent extremity;
- Two corners of different obstacles, if the segment does not traverse either of the two obstacles and does not have any non-locally tangent extremity.

The resulting graph contains more edges than the tangent graph, and often contains more edges than the visibility graph. Although it is not guaranteed to contain the shortest path in 3D space, it allows for some possibilities of movement such as going above, below, or through an obstacle.

The HRM is exploited by a 3D path planning algorithm called Hybrid A* (HA*). HA* is an extension of the A* algorithm that can handle a form of continuous state variables. It can also be seen as a deterministic special case of the HAO* algorithm (Meuleau et al. 2009a).

HA* associates with each vertex v of the HRM a finite set of intervals $\mathcal{R}_v = \{(l_i, u_i), i = 1, 2, \ldots, k_v\}$ representing the altitudes at which v can be reached from the current position of the aircraft. The sets $\mathcal{R}_v$, $v \in \mathcal{V}$ can be are computed incrementally, by propagating altitude intervals along the edges of the HRM. An ϵ-length interval is created to represent the initial position of the aircraft. This seed is then *pushed through* every edge starting in v_0, which creates new altitude intervals that are propagated through the graph in turn.

Given:(i) the data structure representing a vertical cut of the terrain as in Fig. 3; (ii) an initial altitude interval representing the set of altitudes reachable at the start of this edge; (iii) the aircraft minimum and maximum descent rate; it is possible to compute the set of altitudes reachable at the end of the edge as a finite set of intervals. Moreover, if we assume that:

1. The cost of traversing an obstacle depends only on the nature of this obstacle and the 2D-distance travelled in the obstacle (that is, we neglect altitude variations when computing the distance travelled)[5];
2. The cost of traversing a slice of the terrain cut depends only on the most expensive obstacle traversed (that is, this cost is computed by assuming that all the slice length is travelled inside of the most expensive obstacle traversed);

then the basic node evaluation function of the A* algorithm (traditionally denoted as g, h and $f = g + h$) are constant over each altitude interval of the arrival node.

This property is the basis of the HA* algorithm. Based on it, the algorithm computes finite partitions of the (infinite) hybrid-state space, such that the basic A* functions are constant over each partition. Then it performs standard A* search in the space of partitions. The partitions are built on the fly, as the search progresses. A detailed presentation of the algorithm can be found in previous versions of this paper (Meuleau et al. 2008; 2009b).

As with the VG and LVG, the HRM is able to find optimal ways to go around obstacles by following the sides of obstacles. As with the LVG, it is capable of a limited number of traversals above, through or below obstacles. In contrast with LVG, it finds near-optimal ways to traverse a set of obstacles by varying the aircraft altitude along a given direction. It also suffers from the restricted set of edges that are considered: in many cases, a path that traverses some obstacles can be improved by shortening the distance travelled in the most costly obstacles, which requires breaking an HRM edge is several sub-edges that are not co-linear. However, local search algorithms can compensate for this drawback. HA* is often able to find a solution that is qualitatively interesting and that local search can optimize.

An important drawback of the HRM approach is its sensitivity to the number of obstacle vertices (corners). This is also true for the VG and LVG, but the HRM is more sensitive to the number of waypoints, because HA* is much more costly than standard A* search per edge calculation. At the other end of the problem spectrum, the HRM may also fail to find obvious solutions in situations when there are few obstacles, because the roadmap is too sparse to contain interesting paths. This drawback is also shared with VG and LVG. One way to remove it is to add random waypoints. This observation inspired the VPRM approach presented below.

[5]The horizontal speed of an aircraft is so much greater than the vertical speed that the difference between 2D distance and 3D distance is usually negligible.

Probabilistic Roadmaps: Another approach we have adapted to our domain is the *probabilistic roadmap* (PRM).[6] A standard motion planning technique in robotics, this algorithm has two phases. The first builds the graph; V vertices are randomly generated in the reachable state space and edges are considered between each vertex and its n nearest neighbors. If an edge is traversable, it is added to the graph. In the second phase, the start and end vertices are connected to the graph (again, by connecting each to nearest neighbors), and a graph search algorithm looks for a path between them.

We adapt this approach to the ELP problem in the following way:

- Our roadmaps contain 2000 waypoints generated within the range of the aircraft. Each waypoint is connected to its 200 nearest neighbors, resulting in a graph with around 400,000 edges.
- Edges are directed and can go through traversable obstacles. In our domain, very few edges pass through untraversable obstacles, but many can be expensive;
- As explained before, our graph search algorithm is A* with an informed heuristic and search nodes possibly augmented to account for remaining resources;
- Because the expensive part of the algorithm is the edge computations, we use a lazy approach that does not compute the traversibility and cost of an edge until that edge is reached during A* search. Once computed, results are stored for the rest of the search.[7]

The advantage of the PRM approach is its ability to build and consider non-trivial paths through traversable obstacles, and insensitivity to the complexity of obstacles (number of corners). Because of this it represents an attractive alternative to the systematic approaches presented above. Its main drawbacks are (i) the complexity of nearest neighbors computation, and (ii) the inability of the algorithm to find locally optimal solutions. For Example, in the presence of untraversable obstacles, VG-based roadmaps find near-optimal trajectories that cut as close as possible to the obstacles. In contrast, PRM may find a path that wanders far from the obstacles, and so is judged infeasible because of limited resources. If the PRM path is not discarded as too costly, local search can compensate for this drawback and move it closer to the obstacles.

VPRMs: We developed the VPRM in an attempt to combine the advantages of the PRM and systematic approaches derived from the visibility graph. The idea is to seed the graph of a PRM using some or all of the layered-visibility

[6]Our code extends the OOPSMP motion planning library, http://www.kavrakilab.org/OOPSMP/index.html.

[7]Note that this is different from the classical lazy-PRM approach (LPRM). In LPRM, the edge costs do not account for possible collision with obstacles until an optimal solution is found. Collision checking is performed only along the edges traversed by the optimal solution. If the path collides, the cost of these edges is updated and search is restarted.

graph vertices and edges. Since they can be ideal for skirting obstacles efficiently, the PRM finds shorter, smoother paths around obstacles. Conversely, random waypoints and the additional edges introduced by the PRM help the LVG to find non-trivial paths through, above, and below obstacles.

Some previous work used notions of visibility inside of a PRM approach. For instance, Siméon et al. (2000) use visibility to reduce the number of vertices in a PRM. The idea is that both building the roadmap and extracting a solution are faster if there are fewer waypoints. Our motivations and our approach are different here. Because we want *optimal* paths to the goal (as opposed to *any* solution), we are not interested in drastically reducing the number of nodes in the roadmap. Instead, we want to cover the reachable space as completely and uniformly as possible, so that a near optimal solution has a good chance of being included in the graph (at least a solution that is qualitatively optimal and that local search can turn into a true optimal solution). By adding some of the visibility graph edges to the PRM, we actually increase the size of the roadmap. However, we hope to include edges that have a good chance to be in optimal solution, and that would probably not be discovered just by random generation of waypoints.

Path Planners with Dynamics: The LVG, PRM and VPRM path planners can be augmented with aircraft dynamics to output trajectories that are directly flyable. We refer to these planners as DLVG, DPRM and DVPRM respectively. To achieve this, we replace the 3D Euclidean state space by a 4D *configuration space* that includes aircraft heading.[8] Then the Trajectory Planner can be used to find flyable paths between neighboring states, and the cost of such a path can be computed given the risk of the regions traversed by the resulting path.

DPRM waypoints are generated by sampling the 4 dimensions of the space. DLVG waypoints are generated in the following way: for each LVG waypoint we now add two vertices, one for each of the headings tangent to the corner of the obstacle.[9] Planning is performed as in the Euclidean space, except that edge costs reflect the aircrafts dynamic constraints, and so the resulting path is guaranteed to be flyable.

Taking into account the dynamics while generating the initial path can be a considerable advantage. There may be a locally optimal path that attracts all algorithms without dynamics but is not feasible. For instance, suppose that all algorithms without dynamics try to pass to the North of an obstacle to avoid turbulence to the South. However, this solution may not by flyable because of limited aircraft maneuverability. Only PRM with dynamics can identify this fact and look for a path to the South and through the region of turbulence. In this case, this is a crucial advantage because even local search algorithms can not make a qualitative jump such as changing the side that a path goes around an obstacle. However, taking into account dynamics has a cost in terms of complexity. First, the computation of the cost of an edge is more expensive. More importantly, the dimension of the search space is augmented when we move from Euclidean to configuration states, and so more waypoints are needed to cover the space adequately.

[8]The experiments reported in this paper assume a constant speed, and so the search space is only 4D rather than 5D.

[9]We only connect two vertices inherited from the visibility graph if the difference in heading between them is less than 90 degrees. This avoids, for example, connecting two vertices on adjacent corners of an obstacles that are heading in opposite directions.

2.2 Local Search

Local search algorithms are responsible for trying to improve a pre-existing path to find a better solution. We report on two such algorithms.

Euclidean Local Search (ELS): As its name indicates, this algorithms works in the 3D Euclidean space, and so it outputs trajectories that cannot be flown directly because they do not respect the aircrafts dynamic constraints. The basic idea is to try to improve the given path by skipping some of its waypoints. It was later augmented to allow the creation of new waypoints.

ELS works in the following way:

1. A new graph is created by taking the transitive closure of the initial path. That is, each waypoint in the path is connected to all of its descendants.[10] The resulting graph is searched and its optimal solution is passed to the next stage. It is the best path that can be built from the set of waypoints in the initial path, and so it is at least as good as the initial path.
2. The path output at stage 1 is *densified*. That is, waypoints are added at regular intervals along each leg of the path (every two nautical miles). Waypoints are then connected as in the first step, and the new graph is searched for an optimal solution. Note that the new set of waypoints subsumes the waypoints in the initial path, and so this stage is also guaranteed to improve the path.
3. Stage 2 is repeated as long as the path improves by more than a certain percentage (1%).

Graph search at stage 1 and 2 is performed using the A* algorithm with the same heuristic as for path planners.

This simple mechanism proved very efficient in practice. It is capable of important quantitative improvements such as:

- Bringing closer to an obstacle a path that wanders far away from it (e.g., a PRM path);
- Conversely, moving a path that uselessly follows the edges of obstacles (e.g., an HRM path) away from those obstacles;
- Finding optimal (shortest) cuts through the most costly obstacles.

Of course, there is no guarantee of such improvements, as the performances of ELS depends on the seed path provided. For example, local search will not be able to improve a straight line path, even though it may be very costly.

[10]A descendant of a node is either an immediate successor, or a descendant of an immediate successor.

Another drawback of this technique is that its worst-case complexity is exponential in the length of the path: in the worst-case, A* enumerates all trajectories that can be built from a set of waypoints. If there are n waypoints, then there are 2^n such trajectories. Moreover, the number of waypoints n in the densified path is proportional to the path length. To limit this effect, we modified stage 2 so that the search skips all trajectories that have more than two consecutive waypoints taken from the same leg of the initial path. It avoids considering trajectories with three aligned waypoints, which are as well represented by discarding the middle waypoint. As a result, the worst-case complexity of the algorithm becomes exponential in the number of legs in the initial path, which is always much lower.

Dynamics Local Search (DLS): As explained above, the Trajectory Planner takes a 3D Euclidean path and outputs a 4D path that flies through the same set of locations and respects the aircraft dynamics. This translation is a crucial step in our system as the trajectories output by all 3D Euclidean path planners (a majority of our path planners) need to go through it to be usable by the pilot. However, it is hard to predict the results of this process, particularly if the aircraft has low controllability and the path is long. Moreover, the path planner ignores obstacles, and so it sometimes makes non-optimal choices. For instance, it might make a left turn and hit an obstacle while a right turn, although slightly longer, would avoid the obstacles.

For these reasons, we tried to develop a local search algorithm that can perform the same translation while optimizing the output path. The DLS is an attempt at generalizing the ELS second stage in that way. The idea is to use the Trajectory Planner to evaluate the same set of Euclidean paths as generated by ELS, and to return the best 4D path obtained.

The major problem with this simple schema is the following: the cost of an edge between two states depends on the four variables of states. In other words, there is no well defined cost between two 3D Euclidean states that can account for aircraft dynamics. As a consequence, the A* algorithm cannot be used to find the optimal path. Therefore, DLS generates all dynamics-free paths in a systematic depth-first search, evaluates each of them globally using the Trajectory Planner, and then returns the highest valued. It is still an optimal algorithm, but it does not benefit from all computational advantages of A*. As a result, the worst-case 2^n complexity is realized at each run. However, it is reduced to being exponential in the number of legs using the same technique as in ELS.

As the set of 3D trajectories explored by ELS and DLS contains the seed path, DLS is guaranteed to produce a path that is at least as good as that produced by the Trajectory Planner.

3. Results

3.1 Experimental Setup

We have run our algorithms on a wide variety of data. In many situations, solutions are obvious and all algorithms find them quickly. For this paper, we have selected four different emergencies, which include a variety of interesting obstacles that serve to differentiate our approaches (even for these, many airports are either impossible or trivial to reach):

1. An aircraft near Flagstaff, AZ with large thunderstorms to the north and west, and a small gap between them. There are some minor terrain obstacles at lower altitudes and relatively few airports within range.
2. A second version of the Flagstaff emergency where the large thunderstorms are merged.
3. An aircraft near Washington DC, with some nearby regions of turbulence, icing and rain, as well as some special-use (restricted) areas that should be avoided if possible. Terrain is not relevant, and there are many airports nearby.
4. An aircraft near Dubuque, IA with some large weather cells to the north. Terrain is not relevant and there are an average number of airports nearby.

We vary each emergency in three ways. First, we use either 90 or 150 nautical miles as the aircraft's maximum range[11]. Second, we vary the aircraft's ability to turn, using bank angles that are either normal (45 degrees) or severly limited (5 degrees); the latter results in wide turns. Finally, we vary the aircraft's ability to ascend and descend, either leaving it effectively unconstrained, or severly limited.

This gives us 8 different versions of each emergency. For each version, we consider the top 100 candidate runways (if there are that many), based on a heuristic that measures the obstacle-free distance and the quality of the runway itself, resulting in a total of 616 problem instances.

For each runway in each version of each emergency, we run each of these algorithms (note that we do not include results for the visibility graph since it is strictly dominated by LVG):

1. Straight: This computes a straight-line edge between the start and goal, to provide a baseline comparison. In many cases, even in the presence of obstacles, this is the best solution.
2. HRM: The hybrid roadmap.
3. LVG, DLVG: The layered visibility graph, both with and without dynamics
4. PRM, DPRM: The probabilistic roadmap, both with and without dynamics.
5. VPRM, DVPRM: The probabilistic roadmap seeded with visibility graph vertices and edges, both with and without dynamics.

Finally, we can measure the quality of the path planner by itself (PP), or when combined with Euclidean local search (PP+ELS), local search with dynamics (PP+DLS), or both (PP+ELS+DLS). Each individual algorithm is given a 30 second time limit.

[11]For DC we only use a 90 mile range.

3.2 Results

Table 1 summarizes our results. For each combination of algorithm and local search, we provide the average probability of success and the number of failures out of the 2464 runs. We also show the average and standard deviation of runtimes. These are cumulative; for example, the entry for PP+ELS includes the time to run PP.

First, consider the algorithms without local search (PP). While all algorithms outpeform Straight, as expected, the differences vary significantly. While HRM and LVG are only 7% better than Straight, on average, DVPRM is 14% better. The best approaches are the ones that incorporate dynamics. This makes sense. Planners that ignore dynamics tend to produce solutions that turn out to be illegal when converted to a flyable path, usually because they end up exceeding the aircraft's range. This is especially true with limited bank angles; for example, the difference between LVG and DLVG is 0% with normal bank angles, but 2.5% for runs with limited bank angles.

While limiting the bank angles tends to hurt planners that do not include dynamics, limiting the ability to descend is different because the approaches that do not use dynamics still incorporate the descent limits into their search. HRM handles descent limits well, due to the hybrid nature of its search. We feared that PRM approaches would have more difficulty, because more edges in the original roadmap are untraversable, but this does not appear to be a significant issue.

When local search is included, things get interesting. As expected, the approaches that did not include dynamics are improved quite a bit, especially by local search with dynamics (preceding DLS with ELS does not seem to help). For example HRM jumps from .35 to .38 and VPRM jumps from .37 to .40. These improvements can mostly be attributed to cases where local search produces a feasible path out of one that is not flyable. For example, VPRM has 1333 failures before DLS but only 1078 afterwards.

The majority of the local search improvements are due to scenarios where the aircraft has limited controllability. For example, local search only improves HRM from 44.1% to 45.6% when the aircraft has full controllability but improves it from 24.7% to 29.2% with limited bank angles. Again, this is due mostly to the fact that local search reduces the number of failures by converting solutions that are unflyable (usually because the aircraft's range is exceeded) into flyable trajectories.

Notice that for approaches that did use dynamics, local search makes no improvement. For these approaches, if the original path planner failed, there is no path that local search can attempt to improve. Notice that local search never reduces the number of failures for these planners.

One advantage of local search does not show up in these numbers. When using visual inspection, local search trajectories tend to look cleaner and simpler, and sometimes avoid extraneous turns or loops. Due to how probabilities are calculated, the cost of traveling extra distance in free space is negligible compared to the cost of traversing obstacles, therefore these better-looking trajectories hardly vary the overall probability of success. Therefore, even if local search does not help rank the runways appropriately, it may be important for producing the final flight plan for a pilot.

While Euclidean local search (ELS) does improve results before dynamics are considered, and results in modest improvements after dynamics are considered, it is outperformed by DLS. For example, ELS improves HRM success from 35% to 36% in 2.1 seconds, on average, but DLS can make a bigger improvement (35% to 38%) in less time (1.6 seconds).

Now consider running times of the path planners. At a high level, PRM approaches are slowest, taking at least 5 seconds per airport. Also, it takes between 25% and 100% more time for any approach to use dynamics, due to the extra cost of each edge evaluation.

HRM, as well as all PRM-based approaches have high standard deviations, due the fact that they have large search graphs to explore in worst-case scenarios. Out of 2464 runs, HRM reaches the cutoff (30 seconds) 130 times, PRM reaches it 48 times, and VPRM hits it 135 times.

As expected, local search is fast, with very small increases in total runtime for ELS and negligible increases for DLS. In fact, our results show no change in DLS; we would need another decimal place for the differences to be seen.

Finally, note that for these experiments we recreate and reevaluate the roadmap for every runway. Because all of our planners could reuse that roadmap for multiple runways, we do not expect it to take much longer to evaluate all nearby runways than to evaluate a single one; the times reported here can be amortized across all runways. While we cannot do the same for the local search algorithms, they appear fast enough to run separately for each individual runway.

We conclude with a couple of examples that show why the randomly generated points created by the PRM approaches can help.

3.3 Example: KCMR 18

This runway in the Flagstaff emergency lies to the west of the aircraft, and one of the thunderstorms is between the aircraft and the runway (Figure 4) Because the shortest distance around the thunderstorm (the tall rectangle) slightly exceeds the 90 mile aircraft range, non-PRM approaches all end up with a straight-line solution through the thunderstorm. PRM, however, is able to find a solution that mostly goes around the storm but cuts through a corner of it to meet the 90 mile cutoff. This results in a solution that scores .71 instead of .32.

3.4 Example: KLFI 08

This runway in the DC emergency lies to the south of the aircraft in a large area of rain (Figure 5). Non-PRM path planners fly directly to the goal because no other waypoints (obstacle corners) are helpful. PRM uses a randomly generated waypoint to come in from a different angle which allows the aircraft to spend less time in the area of rain; this improves the probability of success from negligible to .05.

4. Conclusions and Future Work

We have tried a wide variety of algorithms, and each has strengths and weaknesses. Ordinary visibility graphs are fast

	PP				PP+ELS				PP+DLS				PP+ELS+DLS			
	success		sec.		success		sec.		success		sec.		success		sec.	
Algorithm	avg.	#fail	avg.	σ	avg.	#fail	avg.	σ	avg.	#fail	avg.	σ	avg.	#fail	avg.	σ
Straight	.28	1152	0.0	0.0	.28	1152	0.1	0.1	.28	1152	0.0	0.0	.28	1152	0.1	0.1
HRM	.35	1327	1.6	6.8	.36	1255	2.1	6.7	.38	1172	1.6	6.8	.38	1150	2.1	6.7
LVG	.35	1202	0.1	0.2	.36	1161	0.5	0.9	.37	1058	0.2	0.2	.37	1058	0.5	0.9
DLVG	.37	1085	0.2	0.2	.37	1086	0.4	0.7	.37	1085	0.2	0.2	.36	1090	0.4	0.7
PRM	.37	1304	5.3	5.7	.39	1228	5.9	5.9	.40	1062	5.3	5.7	.41	1010	5.9	5.9
DPRM	.41	991	8.2	6.0	.41	993	8.6	6.1	.41	992	8.2	6.0	.39	1006	8.6	6.1
VPRM	.37	1333	6.8	7.8	.38	1264	7.4	7.9	.40	1078	6.8	7.8	.40	1060	7.4	7.9
DVPRM	.42	996	8.6	6.4	.42	997	9.0	6.5	.42	996	8.6	6.4	.41	1006	9.0	6.5g

Table 1: Average performance comparison of the eight planning algorithms with and without local search processing.

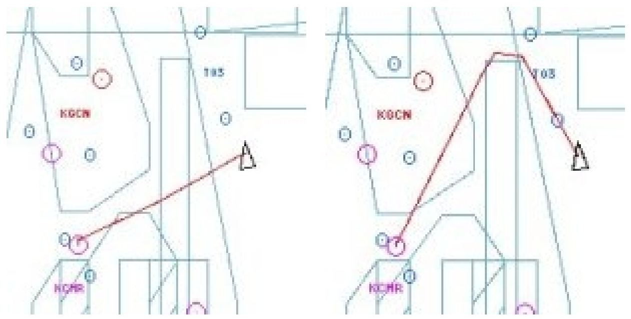

Figure 4: HRM and PRM solutions to runway KCMR 18. PRM is able to mostly avoid the thunderstorm because it can cut the corner, improving the probability of success from .32 to .71.

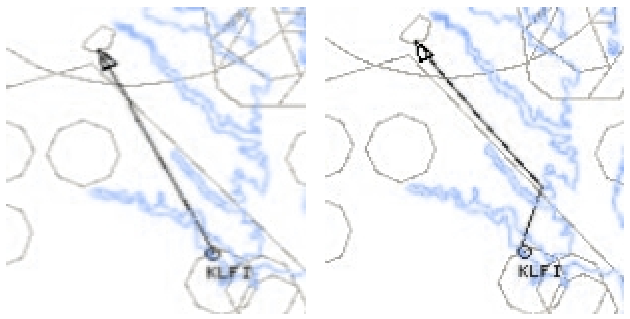

Figure 5: HRM and PRM solutions to runway KLFI 18. PRM find a path that travels along the outside of the obstacle before cutting in as late as possible, improving the probability of success from negligible to .05.

and work well when the space can reasonably be projected into two dimensions (as with columnar obstacles). If obstacles are reasonably sparse, dynamics can be accommodated to a large extent by placing buffers around the obstacles. This approach can even be extended to three dimensions by taking a series of slices through the obstacles at different altitudes and adding edges connecting the points in the different altitude layers. This is essentially what we do in LVG and to seed VPRM. Where the visibility graph approach really breaks down is when it is possible to traverse obstacles, but with increased risk. The HRM technique was one attempt to extend the visibility graph with additional edges that allow traversing obstacles. However, with this approach only a small subset of the possible ways of traversing an obstacle are considered, and the resulting path may be far from optimal. In addition, the HRM technique seems to suffer from the resulting explosion in the number of nodes in the visibility graph.

In contrast, PRMs can be adapted to permit paths both around and through obstacles. In theory, they should always be able to find a high quality path. However, unless one generates a very large number of points (a big roadmap), the resulting paths can be highly non-optimal, and sometimes quite bizarre. Seeding the PRM with a visibility graph helps, because it permits intuitive, optimal paths around obstacles when that turns out to be the best route. The PRM is then relegated to the task of adding additional edges that go through the soft obstacles. A second problem with PRMs is that the resulting paths can be quite jagged. Adding local search as a post-processing step helps to cut corners, eliminate waypoints, and generally smooth out the path. As a result, we are finding that this combination approach produces more natural higher quality paths.

There are several ways the algorithms we have described here can potentially be improved. We have not spent time trying to improve the cases where individual algorithms run slowly. For example, we expect to be able to modify HRM to avoid the cases where it times out, and to modify the local search algorithms to be fast even if the input path contains many waypoints. We should also be able to correct for the case where PRM techniques fail only because the final path is evaluated at a finer granularity than that used in search.

Acknowledgments

This work was supported by the Intelligent Resilient Aircraft Control program of the NASA Aeronautics Research Mission Directorate. We thank John Kaneshige and Stefan Campbell for help with the FLTZ Trajectory Planner, and the full motion simulator. We thank United Captain Mietek Steglinski for discussion on the factors most relevant to deciding between alternative emergency landing sites. We thank the anonymous reviewers for pointers to relevant literature and for their comments on terminology and alternative approaches.

References

Choset, H.; Lynch, K.; Hutchinson, S.; Kantor, G.; Burgard, W.; Kavraki, L.; and Thrun, S. 2004. *Principles of Robotic Motion: Theory, Algorithms, and Implementation*. Cambridge, MA: MIT Press.

Cormen, T.; Leiserson, C.; Rivest, R.; and Stein, C. 2001. *Introduction to Algorithms, Second Edition*. Cambridge, MA: MIT Press. Chap. 33.

Liu, Y., and Arimoto, S. 2004. Computation of the tangent graph of polygonal obstacles by moving-line processing. *IEEE Trans. on Robotics and Automation* 823–830.

Meuleau, N.; Plaunt, C.; Smith, D.; and Smith, T. 2008. Emergency landing planning for damaged aircraft. In *ICAPS-08: Proceedings of the Scheduling and Planning Applications Workshop*.

Meuleau, N.; Benazera, E.; Brafman, R.; Hansen, E.; and Mausam. 2009a. A heuristic approach to planning with continuous resources in stochastic domains. *JAIR* 34:27–59.

Meuleau, N.; Plaunt, C.; Smith, D.; and Smith, T. 2009b. An emergency landing planner for damaged aircraft. In *Proceedings of the Twenty First Innovative Applications of Artificial Intelligence Conference*. To appear.

Mitchell, J., and Papadimitriou, C. 1991. The weighted region problem: finding shortest paths through a weighted planar subdivision. *Journal of the ACM* 38(1):18–73.

Rowe, N., and Alexander, R. 2000. Finding optimal-path maps for path planning across weighted regions. *International Journal of Robotics Research* 19(2):83–95.

Siméon, T.; Laumond, J.; and Nissoux, C. 2000. Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics* 14(6):477–493.