Robust Algorithm for Real-Time Route Planning

ROBERT J. SZCZERBA
PEGGY GALKOWSKI
IRA S. GLICKSTEIN
NOAH TERNULLO
Lockheed Martin Federal Systems

Route planning for intelligent guidance and navigation systems is an extremely complex problem with both military and commercial applications. Standard route planning algorithms usually generate a minimum cost route based on a predetermined cost function. Unfortunately, such a solution may not represent a desirable route for various mission scenarios. We present a novel route planning approach to generate mission-adaptable routes in an accurate and efficient manner. The routes are computed in real-time and are able to take into account various mission constraints including: minimum route leg length, maximum turning angle, route distance constraint, and fixed approach vector to goal position.

Manuscript received September 18, 1998; revised October 31, 1999; released for publication February 2, 2000.

IEEE Log No. T-AES/36/3/07805.

Refereeing of this contribution was handled by T. Froome.

Authors' address: Lockheed Martin Federal Systems, Mail Drop 0210, 1801 State Route 17C, Owego, NY 13827, E-mail: (robert.j.szczerba@lmco.com).

I. INTRODUCTION

Route planning is an important problem for a number of diverse applications including intelligent transportation systems, space applications, autonomous robotics, and military guidance and navigation systems [1, 4, 5, 9–11]. We address the routing problem in the context of route planning for aircraft (rotorcraft or fixed-wing, manned or unmanned), though our approach can be easily extended to the other applications outlined above.

Standard route (in this paper the terms "route" and "path" are used interchangeably) planning algorithms usually generate a minimum cost solution based on a predetermined cost function (relating factors such as terrain features, threat locations, mission requirements, etc.). Unfortunately, such a solution may not represent a "desirable" route for many mission scenarios. A desirable route can be considered a route that does not: 1) exceed the physical limitations of an aircraft, 2) exceed the threshold comfort level and/or workload of a pilot, or 3) violate mission scenario parameters. In particular, many missions require several constraints on the resultant path, such as the following.

1) *Minimum Route Leg Length*: This constrains the route to be straight for a predetermined minimum distance before initiating a turn. Aircraft traveling long distances generally do not want to weave and turn constantly because this adds to pilot fatigue and increases navigational errors.

2) *Maximum Turning Angle*: This constrains the generated route to only allow turns less than or equal to a predetermined maximum turning angle. Such a constraint may be aircraft or mission dependent. For example, aircraft flying in tight formation cannot make severe turns without a greater risk of collision.

3) *Route Distance Constraint*: This constrains the length of the route to be less than or equal to a preset maximum distance. This corresponds to a finite fuel supply or a fixed time at which the goal must be reached.

4) *Fixed Approach Vector to Goal Position*: Constrains the route to approach the goal position from a predetermined approach angle. This could correspond to the approach vector for a runway or for a specific mission objective.

Furthermore, these constraints need not be fixed, and may vary during the course of the mission (i.e., a shorter leg length may be needed at the end of a mission than at the start). An "adaptable" route planner is needed to generate routes based on aircraft limitations and/or mission parameters. The problem of creating such a route planner is quite difficult. In fact, an optimal solution to the general case of this problem is considered to be NP-complete in nature (corresponding to a particular class of

problems in which no polynomial time solution is known to exist) [2]. Even if such an optimal solution could be found (which is highly unlikely), the time to converge to a solution and the memory required would be completely unrealistic for real-time applications.

We present an algorithm which allows for the generation of desirable (as good or better than a pilot could generate) solutions in real-time (much faster than a pilot could do), using a finite amount of memory, while allowing the variation of the four mission parameters outlined above. This algorithm was originally developed for the *Rotorcraft Pilot's Associate* (RPA) Program, under contract to Boeing and U.S. Army AATD. The RPA system is designed architecturally as an intelligent *Cognitive Decision Aiding System* (CDAS) that sits on top of an existing avionics package. CDAS allows pilots to operate attack, scout, and special operations aircraft more efficiently than could previously be achieved. The route planning algorithm discussed here is a key component of the RPA system. More details about the complete RPA program can be found in [1]. An extended abstract of this route planning approach can be found in [12].

Since most of the DoD-related operational route planners are either classified and/or proprietary, we cannot directly compare our planner to these [6–8]. However, after discussions with our DoD customers, we are confident that no existing route planner has the capability to provide solutions in the order of 30 seconds with the constraints discussed above. One operational route planner, The Common Low Observable Auto-Router (CLOAR) is used on aircraft which are considered "low observable" or stealth, such as the F-117A, the B-2, and the Joint Air-Surface Stand Off Missile (JASSM). CLOAR is a classified product, but is purported to determine figure of merit costs as it routes and determine least threat cost route based on inputs. CLOAR is deployed as a module within the Air Force Mission Support System, which is used for *premission* planning on the aforementioned weapons systems. CLOAR route planning times are on the order of hours [7, 8]. Thus, it cannot be used as a real-time in-flight mission planner. Another route planner, the Automated Routing and Maintenance System (ARMS), computes sortie effectiveness as part of the preplanning process, and analyzes data after a sortie has been flown. Like CLOAR, ARMS does not have real-time, in-flight capabilities and cannot address the problem of real-time in-flight constraint-based route planning [6].

An outline of the rest of this paper is as follows. Section II provides background information necessary for understanding our approach. The details of our route planner, called Sparse A* Search (SAS), are provided in Section III, with implementation results given in Section IV. A summary and discussion of future work is presented in Section V with acknowledgments given in Section VI.

II. PRELIMINARIES

Most geometrical route planning algorithms can be considered either grid-based or graphic-based in nature, each with their own associated advantages and disadvantages. Graph-based approaches are usually very accurate but may suffer from long convergence times (possibly even exponential in nature). Grid-based approaches generally are able to converge in real-time, but have difficulty when combining metrics and/or route constraints (such as maximum route distance, maximum turning angle, etc.). Surveys of different route planning approaches can be found in [4, 5] and are not discussed here. Our approach is a novel combination of these two techniques, allowing the generation of mission-adaptable routes in both an accurate and efficient manner. The uniqueness of our approach is the combination of functionality and efficiency, which sets it apart from other route planners. This result is extremely important in the area of intelligent guidance/navigation systems, especially for military applications. Furthermore, our approach is very robust in nature, which allows it to be easily modified and expanded for use in a number of different mission scenarios.

A digitized grid consisting of square cells of equal size represents the environment in which the route planning is performed. The size of the environment (or path planning workspace) is $m \times n$ grid cells. Most miliary applications use Digital Terrain Elevation Data (DTED) as input to their route planners. DTED is grid-based in nature, and hence most military route planners are also grid-based. A route is planned from a given "start" grid cell location to a "goal" grid cell location. Each grid cell corresponds to a particular location in the environment. A *cost estimation* step establishes a "cost" value for a particular cell, corresponding to the cost incurred by traveling through that particular region. This cost value can be used to represent various terrain elevations, cultural features (road, towns, etc.), threat exposure, weather conditions, or any number of other factors. This set of cost values is known as the *map cost* (MC) array and is also of size $m \times n$ [1, 9, 12].

A *cost minimization* step takes the MC array as input and generates a *best cost* (BC) array as output. Each cell in the BC array contains the cost of the cheapest (minimum cost) path to reach the goal from that particular cell. Movement in the BC array is allowed to any of the neighboring eight cells. By continually moving to the adjacent cell in the BC array with the lowest cost, a minimum cost path to the goal from a particular cell can be computed. The BC value of the grid cell of the start position

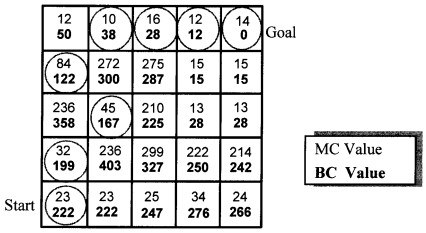


Fig. 1. MC and BC values in grid-based environment.

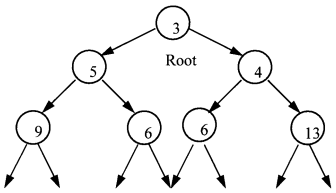


Fig. 2. Example of min-heap with 7 elements.

contains the total cost of the *walking path* from the start to the goal. The walking path is defined as the minimum cost path to the goal if one ere moving over the ground and were not constrained by a maximum turn angle, distance limitations, etc. A simple example of generating a walking path from the BC array is shown in Fig. 1. The upper value of each grid cell represents the MC value from the MC array. The BC values are shown as the lower value in each grid cell. The walking path, represented by the circled entries, corresponds to continually moving to the cell with the lowest BC value until the goal position is reached. There are a number of different ways to efficiently generate the BC array, any of which can be used [5, 9]. Note that for the simple example shown in Fig. 1, we are not considering the different costs associated with movement to horizontally/vertically adjacent cells versus movement to diagonally adjacent cells (i.e., a cost weighting factor of 1.0 versus 1.4, respectively).

To efficiently explore the search space, a data structure known as a *min-heap* is used in our approach to store the cost of various route segments. A *min-heap* is a binary tree whose keys (value stored at a node) satisfy the following heap property: "the key of every node is less than or equal to the key of any of its children." By the transitivity law, the min-heap property implies that the key of every node is less than or equal to the key of all the descendants of that node [2]. An example of a simple min-heap is found in Fig. 2. The root, corresponding to the minimum element in the heap, is indicated. Min-heap data structures are useful for implementing the following two operations.

Insert(x): Inserting a node x into a heap with n elements in $O(n \log n)$ time.

*Remove*_Min(x): Removing the minimum element from a heap (found at the root) and reestablishing the heap property for the remaining elements in $O(n \log n)$ time.

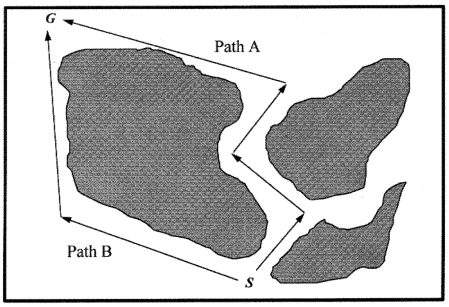


Fig. 3. Approaches based on turn point removal techniques may not be able to produce low-cost flyable route.

One common approach for incorporating additional route constraints (e.g., minimum leg length, maximum route length, etc.) into a search process involves "straightening out" the generated "walking path" until the additional constraints are met. Unfortunately, such an approach will not necessarily lead to a good solution. With such an approach, the generated path (from the BC array) is traversed and a check is made at each turnpoint to see if the turnpoint can be removed (joining the previous and next turnpoints with a straight line) without increasing the overall cost of the path by more than a preset limit (e.g., 10%). The idea is that by removing turn points, the path becomes straighter and thus may be able to meet the imposed route constraints. This approach has many limitations. The primary drawback is that there may be very few possible turn locations that can be removed which do not increase the path cost by a significant amount. Furthermore, with such an approach, there is no way to guarantee a minimum leg length or to impose a maximum turning angle on the generated path without possibly traversing high cost threat areas. An example of this drawback can be seen in Fig. 3. Assume that Path **A** has a slightly lower total cost than that of Path **B**. Thus, a turn point removal algorithm would seek to remove turn points from Path **A** to create a more "flyable" path. Unfortunately, Path **A** is a winding path and the removal of any turn points would result in traversing a high threat area (indicated by the shaded regions). Path **B** would be the logical choice since it has fewer turns and a longer leg length than Path **A**. Unfortunately, this type of approach would never find Path **B** since it was originally considered to be of slightly higher cost than Path **A** and discarded during the search process.

Approaches based on generating a poorer quality route and then attempting to improve it to meet route constraints are not effective for most applications. An approach is needed to keep track of different paths based on a combined metric (turns and leg length) as well as the total cost of the path *during* the search process. This is a very difficult problem to solve in both an accurate and efficient manner.

III. SPARSE A* SEARCH (SAS) APPROACH

For the following approach, the BC array is used as an input to our main route-planning algorithm, which generates the final, flyable route. As discussed previously, a route is desired which is of minimum total MC that has the following characteristics: 1) minimum route leg length, 2) maximum turning angle, 3) route distance constraint, and 4) fixed approach vector to goal position.

We propose a new approach to handle the previously described constraints called Sparse A* Search (SAS). The SAS technique is a novel variation of the standard heuristic searching algorithm A* (pronounced "A-star") which is used quite extensively in route planning and graph searching applications. Before our SAS approach is described, a brief overview of A* in general is presented as comparison.

A* is an optimal, best-first search heuristic that computes a cost function for various locations in an environment [3]. A* explores the environment by computing a cost function for each possible "next" position to search, and then selects the lowest cost position to add to the search space. The addition of this new location to the search space is then used to generate more path possibilities. All paths in the search space are explicitly represented using pointers from each position back to the previous position from which it was derived. Equation (1) is the cost function that is minimized at each step of the A* propagation

$$f(x) = ag(x) + bh(x). \tag{1}$$

In (1), the term $g(x)$ is the *actual* cost from the start position to the intermediate position x. The term $h(x)$ is the *estimated* cost from an intermediate position x to the goal position. a and b are parameters used to weight the actual and estimated costs (usually, both are set to 1). At each step in the A* propagation, the lowest $f(x)$ value is selected and inserted into a sorted list of possible paths. It has been proven that if the actual cost from x to the goal is greater than or equal to the estimate ($h(x)$) of this cost, then the solution produced by A* is guaranteed to be a minimum cost solution [3].

The problem with the A* approach is that, depending on the weighting (cost) factors in the environment, the algorithm may take a very long time (exponential in nature), and use an unbounded amount of memory to converge to an optimal solution. This is especially true for turn angle constraints in a grid-based environment. For such a case, a single grid cell could represent an almost infinite number of nodes in the A* search space, since each cell could theoretically be entered and exited by a proposed path from a different angle. For cases such as these, a true A* approach will not work due to the time constraint imposed for real-time planning systems. To overcome these difficulties, we introduce the SAS approach. The SAS approach accurately and efficiently "prunes" the search space to allow the generation of an acceptable solution that converges in real-time. The next several subsections outline how various route constraints can be incorporated into the SAS approach.

A. Minimum Route Leg Length and Maximum Turn Angle

Assume that initially we are given a start location, goal location, an initial trajectory, minimum route leg length, and a maximum turning angle of the desired route. Thus, there are a finite number of cells that can be reached from the start location with a maximum turning angle and the minimum leg length (assuming an initial heading). These cell positions are divided into sectors and the minimum cost vector (to each cell on the arc bounding a particular sector) from each of these sectors, based on their $f(x)$ value (see (1)), is stored in a min-heap data structure. The number of sectors used is proportional to the memory and accuracy constraints for the specific application. The greater the number of sectors, the more accurate the resulting solution as well as the greater amount of memory required. In order to reduce the memory requirements as well as the amount of time necessary to converge to a solution, only the minimum cost cell in each sector is stored in the min-heap.

The BC array, generated in the cost minimization step (described earlier), provides the values for $h(x)$. The BC array is a novel choice for $h(x)$ because it represents a lower bound on the cost of the path from an intermediate node x to the goal position. Note that we have already shown that the BC array contains the minimum cost walking paths to reach the goal position from any location in the environment. Since we need to introduce turn and distance constraints to the computations, this can only further increase the cost of the generated path. Thus, the BC value *must* be a lower bound for our particular search. It is also a more accurate measure of $h(x)$ than just using the straight line distance to the goal that is more commonly used in various A* applications [4, 5]. Our choice of using the BC array is a significant one because the BC values force the graph search to concentrate on areas with a higher probability of containing the minimum cost route, within the given constraints. The MC array provides the values for $g(x)$ by computing the sum of the MC values along each vector.

At each iteration of the search propagation, the minimum cost node from the min-heap is removed and expanded. This process continues until the goal position is reached. A tree-like data structure (*Sparse A* Search Tree* (SAST)) is used to store the nodes which were "popped" off (or removed from) the min-heap. The final route can be generated by tracing back up the SAST once the goal has been reached.

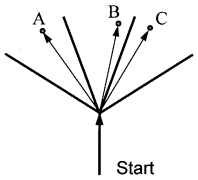


Fig. 4. Minimum cost vector from each of 3 sectors.

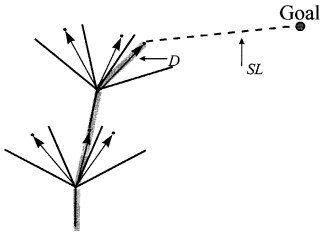


Fig. 5. Node is expanded only if $D + SL <= d\text{-}max$.

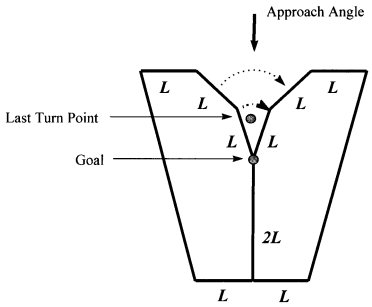


Fig. 6. Bucket-shaped high-cost region in MC array can be used to force predetermined approach angle to goal.

Further details of this technique can be found in subsection IIID.

Fig. 4 shows an example of the initial fan-tail from the start position (the arrow entering the start corresponds to the desired initial trajectory of the aircraft). The fan-tail is divided into 3 sectors and the minimum cost vector (indicated by the arrows) from each of these sectors is inserted into the min-heap H for later propagation. Note that if no initial heading is given, the initial fan-tail can be for a 360 deg circle around the start location. For this case, the entire 360 deg around the start would be divided into sectors and the minimum cost vector for each of these sectors would be stored in the min-heap H.

Also note that, for some applications, the minimum leg length and the maximum turning angle can vary during the route (i.e., a shorter leg length may be allowed at the end of a mission than at the start). Our approach allows for this consideration since each fan-tail may be of a unique size and shape during the search process.

B. Route Distance Constraint

The route distance constraint corresponds to the maximum allowable length of the computed route. This could represent a finite amount of fuel for a particular mission or a constraint on arrival time. No paths of length greater than this distance (referred to as *d-max*) should be considered as possible paths during the search process. As discussed previously, the SAST keeps track of which locations have been searched (corresponding to nodes popped off the SAS min-heap). The approach can accommodate the distance constraint by limiting which nodes are included in the SAS min-heap. For a given node x, we only add x to the min-heap if $D(x) + SL(x) <= d\text{-}max$, where $D(x)$ is the actual distance of the calculated path to reach x from the start position and $SL(x)$ is the straight-line distance from x to the goal. The straight-line distance is added to the current length of the path since the straight-line distance is an intuitive lower bound on the path length from node x to the goal. An example of this is shown in Fig. 5. The introduction of *d-max* has other benefits as well. Even when there are no fuel or time constraints on the route, *d-max* can be used to effectively generate straighter (and possibly more desirable) routes, since *d-max* limits the amount of turning possible in the resultant path. Furthermore, a lower value of *d-max* also significantly speeds up the search time since fewer nodes have to be considered during the search process.

C. Fixed Approach Vector to Goal Position

We can solve the problem of the fixed approach vector (goal approach angle) by adding a high cost offset to the MC array (relative to the rest of the environment) in a "bucket-shaped" area around the goal position. This technique can be effectively used to force the SAS into approaching the goal (or any intermediate point) from a desired direction. For example, assume that the goal location for a particular mission must be approached from a cone about a specified approach angle. For such a case, the shape of the bucket is as indicated in Fig. 6. Since we want the algorithm to terminate when the aircraft is within one leg length (L) of the goal, the bucket must be shaped so that the path cannot "force" its way through the bucket to reach the goal. In Fig. 6, each L refers to the minimum leg length. The two angles at the entry to the bucket (corresponding to the dotted lines) force the path to approach the goal from the desired angle. One could vary the costs of the "bucket" according to the specific mission scenario (relating the cost of the bucket to the costs of different threats along the route).

D. Algorithm Summary

This section summarizes the details of the SAS algorithm as well as discussing modifications for memory limitations and faster performance.

Algorithm 1: Sparse A* Search( )
Input: MC and BC arrays, each of size $m \times n$. Start position, Goal position. Route constraints: max route length (*d-max*), approach to goal vector, minimum leg length (L), maximum turning angle.
Output: List of grid cells corresponding to the final flyable route.

1) Create a fan-tail of 2 times the maximum turning angle from the start position (an aircraft can turn right or left not more than the maximum turning angle). The length of the fan-tail is of minimum leg length.

2) Divide the fan-tail into S sectors. The larger the value of S, the greater the probability of finding a desirable solution (the memory required and the time to converge also increases proportionally). For most applications, a value of 3, 4, or 5 for S is sufficient (note: the initial fan-tail, from the start location, may be larger and divided into more sectors to take into account any preferred initial trajectories).

3) Compute the cost of every vector (of length L) to cells at a distance L from the origin of the fan-tail in each of the S sectors. Recall that the angular resolution of the sectors can be freely chosen in accordance with the computational resources available. Select the minimum cost vector (based on $f(x)$) from each sector.

4) For each of the minimum cost vectors computed in Step 3, insert these vectors into a min-heap H if they don't violate the *d-max* constraint (i.e., $D(x) + SL(x) <= d\text{-}max$, as described in subsection IIIB).

5) Remove the minimum cost (root) element from the min-heap H and expand the search space to that position. Expanding the search space involves repeating Steps 1–4 from this new position.

6) Once a node is removed from the min-heap H, it is expanded and simultaneously stored in the SAST structure to keep track of which nodes were expanded and the direction from which the route entered that particular node.

7) Repeat this procedure until the node removed from the heap is within one leg length L from the goal. When this occurs, a separate "end game" process is used which is arbitrary and may be as simple as "connecting the dots" from the last computed position to the goal position. Additional algorithms may be needed to guide the aircraft (reducing speed, altitude, etc.) to the goal position.

8) If the min-heap H ever becomes empty before the goal position (within one leg length L) is reached, the algorithm terminates since a path to the goal cannot be found within the given problem constraints and parameters. If such a case occurs, the parameters can be changed and the route planning algorithm rerun. Note that rerunning the algorithm with new parameters is only for ground-based premission planners when there is ample time to rerun test cases to determine the best parameter values for different mission scenarios. For in-flight planners, the algorithm parameters are predetermined to ensure that a solution will be found within the required time frame.

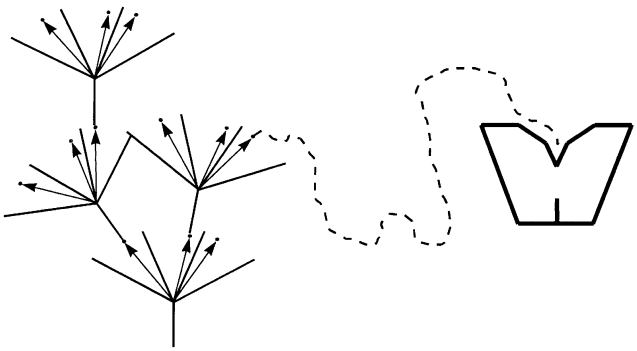


Fig. 7. Snapshot of SAS with fixed approach angle.

9) Trace back the path up the SAST from the last computed position until the starting position is reached. This results in a minimum cost path (based on the pruned search space, environmental resolution, etc.) from the start to the goal. Note that a globally optimal path is not generated.

10) Once the route is computed, if a particular exact estimated time of arrival (or time on target) is desired, the speed of the vehicle can simply be adjusted along the route to arrive at a given time.

Fig. 7 shows a "snapshot" of the overall SAS algorithm. The fan-tails keep expanding nodes until the goal is reached (indicated by the bucket shape). The dotted line represents the walking path ($h(x)$ value) used in computing the minimum cost vectors to place into the min-heap.

To accommodate practical limitations on memory requirements and to obtain real-time performance it should be recalled that the nodes of the SAS search space are stored in a min-heap data structure. If the memory allocated to the min-heap is unbounded, the algorithm will eventually terminate and generate a minimum cost route to the goal (within the given resolution of the search space and the imposed constraints). Unfortunately, for very large environments, the time to converge to a solution, even with such an efficient approach, may be too long for real-time applications. Because of this, we can limit the size of the min-heap and force less desirable solutions to be pruned from consideration. While this approach allows the generation of faster solutions and requires less memory, the main drawback is that by pruning the heap (regardless of the method used), there exists the possibility of potentially good routes being removed from consideration. Thus, if the maximum heap size is set to too small a value, then

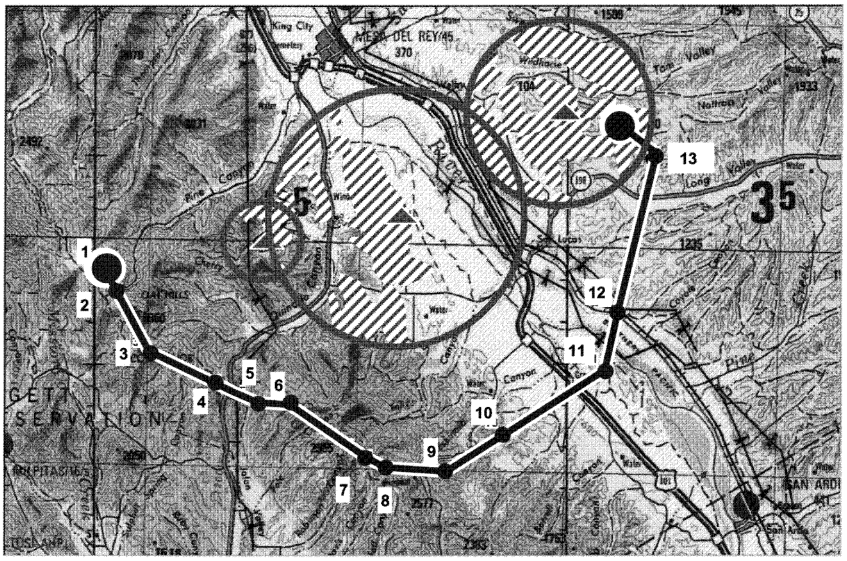


Fig. 8. Route planning SAS algorithm. Large *d-max*.

possible paths leading to desirable solutions may be inadvertently discarded. The larger the memory allocated for the min-heap, the better the chances are for obtaining desirable routes. The method of heap pruning used for this implementation is as follows. When a new element is added to the full min-heap (after the min-heap has already reached its maximum allotted size), random locations at the base of the min-heap are selected and the element to be inserted is compared with the current elements at those positions. If the cost of the new element is smaller than the element to which it is compared, the new node replaces that element and the min-heap is reorganized to maintain the min-heap property discussed earlier.

The SAS route planner may also be used for planning routes with multiple segments (i.e., **A** to **B** to **C** to **D**). Such a situation may occur through the introduction of "hard-points" into the mission. A hard-point is a location that the route *must* pass through, such as a critical target or a rendezvous point. Thus, the computed route may go from starting point **A** to hard-point **B** to hard-point **C** to endpoint **D**. The entire route from **A** to **D** may not represent the lowest cost, but the route from **A** to **D**, that passes *through* hard-points **B** and **C** will be the route that is of lowest cost.

The SAS route planner also works very well in environments with dynamic threat locations. In such a situation, when a new threat appears (or an existing threat location changes) the MC array is updated in real-time to reflect this new information. This can be as simple as increasing the cost value (MC value) of certain cells where a threat is expected (and subsequently reducing the cost value of cells where the threat has left), or as complicated as a stochastic function associated with each of the grid cells.

IV. IMPLEMENTATION RESULTS

The SAS algorithm was implemented and three examples of the routes produced are shown in Figs. 8, 9, and 10. Real world elevation and feature data were used from Northern California (around Fort Hunter Liggett Military Base). Each of these routes consists of a start and goal location, in an area populated by high-lethality threats. The resultant flyable route is indicated by the dark path. The start and goal locations are represented by the larger filled circles at the endpoints of the routes. The three lighter, large circles represent areas of perceived threats. The areas which are shaded within the threat circles are areas where the threat has intervisibility to an aircraft flying at the height specified for this segment of the route, indicating a much more dangerous area. Each of the smaller numbered circles indicates turn point locations in the route where a pilot would have to adjust course. In Fig. 8, the SAS algorithm planned the route with a relatively large *d-max* value (2.5 times the length of the straight-line distance between the start and goal locations), essentially allowing the route planner to plan a safe, low cost path without too much concern for overall route distance. Note that the route is far outside the threat circle in an area of low terrain exposure. In Fig. 9, the SAS algorithm was only allowed to plan a path with *d-max* equal to 1.3 times the straight-line distance between the start and goal, severely restricting how far away from the threat areas the route could go. Note that the paths may cut through the threat circle, but still remain in areas of low intervisibility (since they are not within the shaded areas within the threat circles). In Fig. 10, the pilot requested a route with an approach angle into the goal of 135° (coming into the goal heading north corresponds to an approach angle of 0°, hading east is an approach angle of 90°, etc.), still keeping *d-max* at

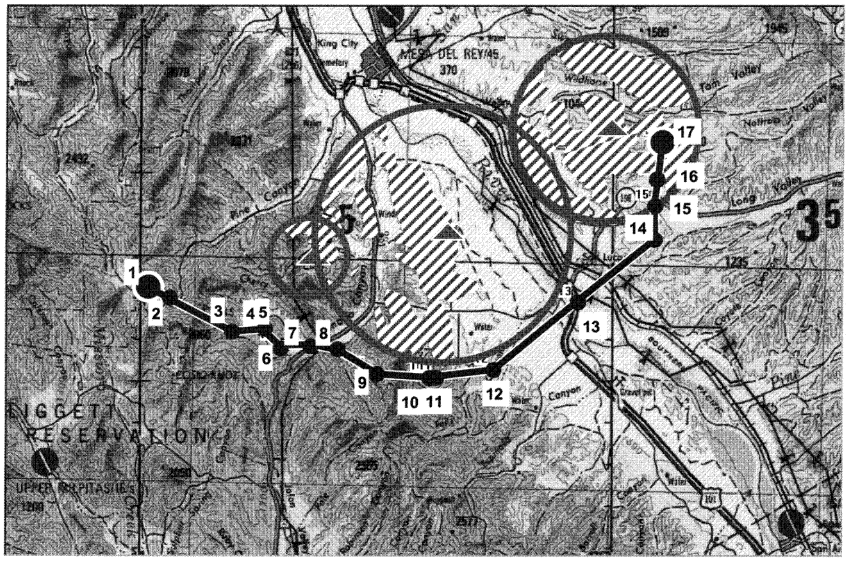


Fig. 9. Route planning SAS algorithm—*d-max* at 1.3 times segment length.

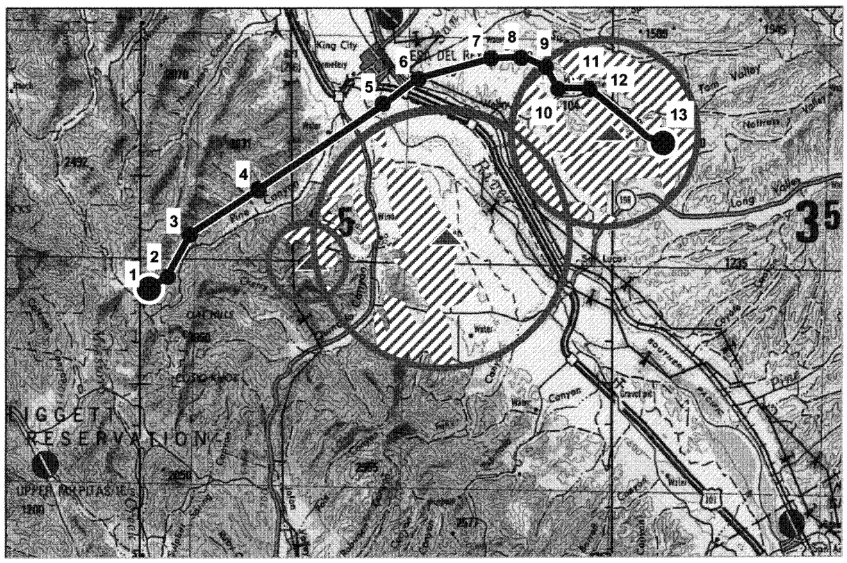


Fig. 10. Route planning SAS algorithm with approach vector at 135 deg.

1.3. Note that the route planner found a route through the cracks in the threat intervisibility to approach the goal from 135° (within predetermined error bounds). The computation time for each of the routes was well under a minute with a resolution of 100 m $\times$ 100 m for each grid cell. The actual computation times and more details about the implementation are considered proprietary information and unfortunately are not able to be included in this paper. More detailed information can be obtained directly through the authors.

V. SUMMARY AND FUTURE WORK

Our SAS route planner is an accurate, efficient, and robust algorithm that advances the state of the art for real-time route planning applications. The ability to introduce various route constraints during the planning process, as well as varying these parameters over the duration of a mission, makes the algorithm valuable for almost all types of intelligent guidance/navigation systems, including, but certainly not limited to air, land, and sea military craft. The uniqueness of our route planner is the combination of functionality and efficiency that it affords. To achieve this desired combination, we introduce novel techniques to prune the search space, while constraining the resultant path to meet the mission parameters.

Since the algorithm is based on the input of an integer cost map, we can apply the problem to a number of other domains by simply varying what is used to compute the cost values in the map. The current version of the SAS route planner constrains the route distance, leg length, approach angle, and

turn angle in a 2-D environment, but other search criteria (such as maximum climb angle, radar cross-section, threat netting, etc.) can also be included. Furthermore, the SAS route planner can be extended to route planning in higher dimensional environments as well as to environments with multiple, dynamic goal locations. Researchers at Lockheed Martin Federal Systems are currently addressing these additional features.

ACKNOWLEDGMENTS

The following people have contributed in various ways to the development and testing of the SAS route planner (in alphabetical order): Kathy Ballester, Chris Bodenhorn, Dave Card, Garf Cooper, Heidi Ng, Steve Rostedt, Diana Shalkey-Federowicz, and Peter Stiles.

REFERENCES

[1] Bodenhorn, C., Galkowski, P., Stiles, P., Szczerba, R., and Glickstein, I. (1997)
Personalizing onboard route re-planning for recon, attack, and special operations missions.
Presented at the 1997 American Helicopter Society Conference (Avionics and Crew Systems Technical Specialists Conference), Sept. 1997.

[2] Cormen, T., Leiserson, C., and Rivest, R. (1990)
Introduction to Algorithms.
New York: McGraw Hill, 1990.

[3] Hart, P., Nilsson, N., and Raphael, B. (1968)
A formal basis for the heuristic determination of minimum cost paths.
IEEE Transactions of Systems Science and Cybernetics, **4**, 2 (July 1968), 100–107.

[4] Hwang, Y., and Ahuja, N. (1992)
Gross motion planning—A survey.
ACM Computing Surveys, **24**, 3 (Sept. 1992), 219–291.

[5] Latombe, J. (1991)
Robot Motion Planning.
Boston, MA: Kluwer, 1991.

[6] Mitchell, F. (1996)
Use of preprocessing cruise missile data for strategic planning.
NTIS: AD-A318 875/2, Nov. 1996.

[7] Rodriques, L. (1999)
Defense acquisitions—Achieving B-2A bomber operational requirements.
GAO/NSIAD-99-97 Report, July 1999.

[8] Rodrigues, L. (1997)
Real-time replanning for stealthy missions.
International Defense Digest, **30** (1997).

[9] Stiles, P., and Glickstein, I. (1994)
Highly parallelizable route planner based on cellular automata algorithms.
IBM Journal of Research and Development, **38**, 2 (Mar. 1994), 167–181.

[10] Szczerba, R. (1996)
New cell decomposition techniques for planning optimal paths.
Doctoral dissertation, University of Notre Dame, Notre Dame, IN, 1996.

[11] Szczerba, R., Chen, D., Uhran, I., Jr. (1997)
A framed-quadtree approach for determining Euclidean shortest paths in a 2-D environment.
IEEE Transactions on Robotics and Automation, **13**, 5 (Oct. 1997), 668–681.

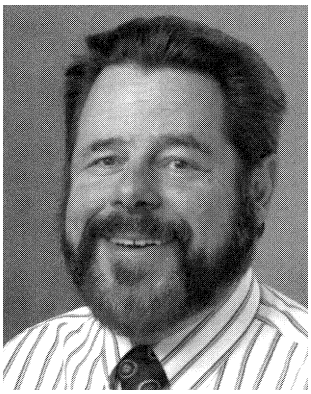
[12] Szczerba, R., Galkowski, P., and Glickstein, I. (1998)
A mission adaptable route planner for intelligence guidance/navigation systems.
Presented at the AIAA Conference (Aerospace Sciences Meeting and Exhibit), Jan. 1998.

Robert J. Szczerba received his B.S. and M.S. degrees in electrical engineering and a Ph.D. in computer science and engineering from the University of Notre Dame, Notre Dame, IN, in 1990, 1993, and 1996, respectively. He did post-doctoral research with Notre Dame and NASA's Jet Propulsion Laboratory, focusing on developing massively parallel algorithms for planetary rovers.

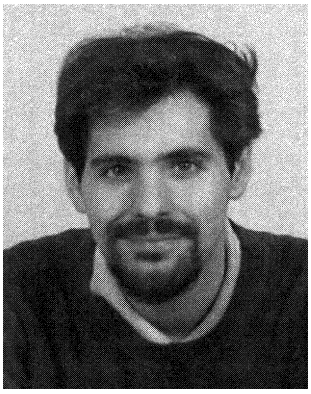
He is currently a Research Scientist at Lockheed Martin Federal Systems in Owego, NY in the Advanced Technology Department. His main research interests include artificial intelligence, autonomous systems, parallel computing, and intelligent decision support systems.

Peggy J. Galkowski is a Senior Engineer at Lockheed Martin Federal Systems in Owego, NY. She received her B.S. and M.S. degrees in electrical engineering from M.I.T. in 1975.

In 1975, she joined Lockheed Martin Federal Systems (formerly IBM). During her career, Ms. Galkowski has developed algorithms for route planning systems, guidance and navigation systems and mission management systems.



Ira S. Glickstein received his Bachelors degree in electrical engineering from City College of New York and his Masters and Ph.D. in systems science from the Watson School of Engineering, Binghamton University in 1961, 1989, and 1996, respectively. He is a Senior Systems Engineer at Lockheed Martin Federal Systems (formerly IBM) in Owego, NY, and an Adjunct Instructor in Computer Science and Systems Engineering at Binghamton University and in Software Engineering at the University of Maryland University College. His main research interests include artificial intelligence, advanced visionics, avionics architecture and hierarchy theory.



Noah J. Ternullo received his B.S. in electrical engineering from Binghamton University in 1997, where he is also currently completing his M.S. in computer science.

Mr. Ternullo is a Senior Associate Systems Engineer at Lockheed Martin Federal Systems in Owego in the Avionics Department. His current primary research interests are in the area of mobile computing.