The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

UAV Intelligent Path Planning for Wilderness Search and Rescue

Lanny Lin
Computer Science Department
Brigham Young University
lanny.lin@byu.edu

Michael A. Goodrich
Computer Science Department
Brigham Young University
mike@cs.byu.edu

***Abstract*— In the priority search phase[1] of Wilderness Search and Rescue, a probability distribution map is created. Areas with higher probabilities are searched first in order to find the missing person in the shortest expected time. When using a UAV to support search, the onboard video camera should cover as much of the important areas as possible within a set time. We explore several algorithms (with and without set destination) and describe some novel techniques in solving this problem and compare their performances against typical WiSAR scenarios. This problem is NP-hard, but our algorithms yield high quality solutions that approximate the optimal solution, making efficient use of the limited UAV flying time.**

I. Introduction

The use of mini-UAVs (Unmanned Aerial Vehicles) in Wilderness Search and Rescue (WiSAR) has gained interest for researchers and experienced advancement in recent years due to its low cost, portability, and potential field use [6]. The UAV onboard video camera provides visual support, enables search and rescue workers to systematically survey large areas of importance in real time [6, 17], and increases the workers' awareness of the environment.

For WiSAR, as time progresses, the survivability of the missing person decreases and the effective search radius increases by approximately 3km/hour [19, 22]. Therefore, search efficiency can dramatically affect the outcome of the search and rescue. In the prioritized search phase, the incident commander creates a probability distribution map for finding the missing person based upon terrain features, profile of the missing person, weather conditions, and subjective judgment of expert searchers. Such maps can also be created systematically by utilizing geographical information available to the public via the Internet [4, 13, 21]. UAVs have limited flying time, and in most cases, it is not long enough for the onboard video camera to cover the entire search area. For these reasons, the important question is this: given a probability distribution map, a starting point, an ending point (optional), and specified flying time, what is the best path that enables the UAV onboard video camera to "cover" as much of the probability distribution as possible?

Characteristics such as possibly repeated visits and probability cumulation make this a more challenging problem than standard Orienteering Problem (OP) and coverage problem. Contributions of this paper include novel path planning techniques ("global warming effect", path crossover/mutation), additional specified-destination constraint while accumulating probability, a solid validation of the algorithms' performance, and applying algorithms to a practical, real-world application. Experimental results from this paper are conducted in simulation and not on-board a real UAV.

[1]Four qualitatively different types of search strategies are used in WiSAR: hasty search, constraining search, priority search, and exhaustive search. See [6] for more details.

II. Problem Formulation

We model this problem as a discretized combinatorial optimization problem with respect to probability accumulated in the 2D space for UAVs that use gimbaled cameras. Using Koopman's search metric of the instantaneous probability of detection by one glimpse [10], we assume the observer has a 100% target detection rate. This means that as the UAV camera footprint moves along the probability distribution map, it collects ("zeros out") all the probability along the way and accumulates the probability. A good analogy would be thinking of the UAV as a vacuum cleaner sucking up probabilities with 100% efficiency.

In WiSAR operations, a UAV maintains an altitude of approximately 60m above ground and travels at roughly 12–13m/s [6]. With this height, the onboard camera footprint size comes to about 32m×24m. The batteries on the UAV can keep it airborne for approximately 1–2 hours depending on weather conditions. We assume that the UAV will always maintain the same height of 60m above ground (through Height-Above-Ground automation) and travel at the constant speed of 12m/s, and use 24m×24m as the effective camera footprint size. Given these parameters, a 60×60 probability grid, where each probability node is 24m×24m, represents an area of 2.0736km^2 that will take the UAV 2 hours to cover entirely. In our path planning, we restrict the direction a UAV can travel to only North, South, West and East (making only 90 degree turns), and it takes the UAV 2 seconds (1 time step) to travel from one node to its direct 4-connected neighbor. In real flights, a UAV can approximate a 90 degree turn (covering 3 nodes) in 4 seconds, so this model is close to UAV's capabilities. Also during roll or yaw, the gimbaled camera can rotate to remain aiming straight down, enabling the 90 degree turn of the camera footprint.

Using i for the row number and j for the column number, each probability node (cell in grid) can be written as N_{ij} where $0 \leq i, j < 60$. The value of each N_{ij} is the total volume of probability within the grid cell and thus

$$\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} N_{ij} = 1, \tag{1}$$

978-1-4244-3804-4/09/$25.00 ©2009 IEEE

where n=60. Let T be the total number of time steps allowed for the UAV (specified flying time). Let P be the set of all possible paths for the UAV on the probability grid for T time steps. Each path, $p_k \in P$, can be represented by a sequence of probability nodes $\{N_0, N_1, N_2, ..., N_T\}$ consisting of T+1 nodes. If the UAV is allowed to visit a node more than once, then the same node can be in a different part of the sequence.

If we use a binary variable x_{ij} to represent whether $N_{ij} \in p_k$, x_{ij} becomes a function of path p_k:

$$x_{ij}(p_k) = \begin{cases} 1, & N_{ij} \in p_k \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

The number of unique nodes visited is less than or equal to the length of the path:

$$\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} x_{i,j}(p_k) \leq T + 1, \quad (3)$$

and the total probability accumulated, PC_{p_k}, if the UAV follows path p_k is

$$PC_{p_k} = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} x_{ij}(p_k) N_{ij}. \quad (4)$$

The optimal path $p^* \in P$ is defined such that $\forall p_k \in P, PC_{p^*} \geq PC_{p_k}$, and our goal is to find or approximate the path p^*, which produces the maximum cumulative probabilities within reasonable computation time.

III. Related Work

Many algorithms have been used for UAV path planning such as Voronoi Diagram with Eppstein's k-best paths algorithm [1], A* [17], LRTA* [9], and Probability Roadmaps [16]. These papers focus on obstacle avoidance and sensing multiple targets.

For path planning in searching for a target, some researchers propose to use a probabilistic model and try to maximize accumulated probability along the path. In [8], Hansen et al. propose three search strategies: greedy, contour, and composite search, using a probability grid. In a series of papers (e.g. [2, 3]), Bourgault et al. describe a Bayesian framework for trajectory planning to maximize the chances of finding the target given restricted time using one or multiple UAVs and human systems. However, the solution uses a very simple 1-step lookahead approach which generates paths far from optimal and difficult to improve upon. Both papers do not consider the possible set destination constraint and also lack solid validation of the path efficiency.

If we disallow visiting the same node more than once, this problem falls within a variation of the Traveling Salesman Problem (TSP) called the Orienteering Problem (OP) [18] or the Prize-Collecting Traveling Salesman Problem (PCTSP) [7], both of which are NP-Hard [20]. Many exact solving methods for the OP have been developed ([5, 11, 18]. These exact methods can find optimal solutions to small OP problems, but for large-scale OP problems, approximation heuristic approaches are preferred. Mittenthal and Noon [15] present a heuristic approach that inserts or deletes a city from the subset-tour. Tasgetiren and Smith propose a Genetic Algorithm in [23] that encodes tours using a sequence of points and uses a penalty function to help search infeasible regions. Liang and Smith present an Ant Colony Optimization approach that uses an unusual sequenced local search and a distance-based penalty function in [12]. These algorithms work well with OP problems of small number of nodes (21–100 nodes) but can be slow with large number of nodes. They also don't allow repeated visits.

IV. Path Planning Algorithms

Because none of the path-planning algorithms we discussed above work well under our model of the problem, we developed a set of algorithms based on the following ideas: Local Hill Climbing (LHC), Convolution, and Evolutionary Algorithms (EA). We also verify the paths generated to ensure the UAV is not flying backward or going outside of the allowed search area.

A. Algorithms without a Set Destination

In situations where the operator does not have a preference for where the path should end, the following algorithms were built and evaluated.

1) Complete-coverage Algorithm (CC): The algorithm plans flight paths by following a lawnmower pattern. It first identifies the smallest $m \times n$ bounding rectangle that contains all the non-zero probability nodes. If the starting location is inside the pattern, the algorithm simply generates a path following the pattern. Otherwise, it first plans a shortest path to the edge of the bounding rectangle. When allowed flight time is large enough, this algorithm is guaranteed to collect all the probabilities.

2) Local Hill Climbing Algorithms (LHC): This is a greedy algorithm that always follows the direction with the highest value. A direct implementation of LHC does not work well with a multi-modal probability distribution map because the path generated stays with one mode until it has covered it completely before moving on to another. To address this problem, we use a global warming metaphor where the "ocean surface" represents all the zero-valued nodes and the "islands" represent the probability modes; see Fig. 1. We subtract a constant C from all nodes but keep all node values non-negative, where $C = \max(N_{ij})/l$, and l defines how fine grained the search should be:

$$N'_{ij} \leftarrow \begin{cases} N_{ij} - C, & N_{ij} > C \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

When the ocean surface rises C each time, the volume of islands above water decreases, and if the ocean surface rises l times, all islands will be below water. In our experiments we set l=40 and use the LHC algorithm to generate 40 paths: one before the ocean surface rises and one for each time the ocean surface rises (before water covers everything). We then recompute the probability accumulated for these 40 paths using the original probability grid and return the best path. This global warming technique allows the LHC algorithm to break out of one mode before completely covering that mode and move toward another. In case of a tie as to where to go next, we use two methods as the tie-breaker: LHC-GW-CONV uses a convolution kernel (with small, medium and

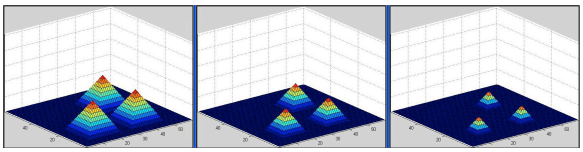

Fig. 1. Global Warming Effect

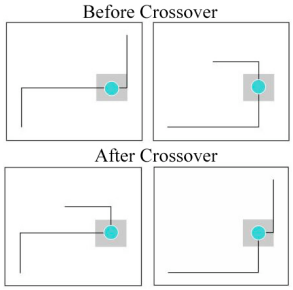

Fig. 2. An example of single-point path crossover (Upper row: the parents. Lower row: the children)

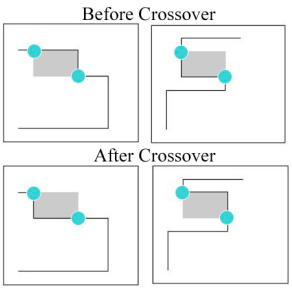

Fig. 3. An example of double-point path crossover (Upper row: the parents. Lower row: the children)

large sizes) to determine which neighbor is more promising, and LHC-GW-PF uses Potential Fields (PF) with various discounting factors to determine where to go next.

3) Evolutionary Algorithms: We developed two Evolutionary Algorithms: EA-Dir and EA-Path. Both use the probability accumulated for each path as the fitness function and employ the proportional selection method [14]. The difference between the two algorithms lies in the path representation during crossover.

With the EA-Dir algorithm, a path is encoded as a string of directions consisting of North, East, South, and West in the crossover phase (e.g. "NNWEE..."). Because the paths generated using single-point crossover [14] have a very high probability of being invalid (flying out of the map), we only use double-point crossover [14] and restrict the mid-section to a fixed 5-direction string.

With the EA-Path algorithm, a path is encoded as a sequence of node positions. If the two parent paths share only one common node, then single-point crossover is used; if they share two common nodes in the same order, then double-point crossover is used; otherwise, the two parent paths are discarded and the process starts over. For the single-point crossover method the two parent paths are crossed at the common node; see Fig. 2. For double-point crossover method, the first common node and the second common node in the parent paths mark the middle sections to be swapped; see Fig. 3. Both techniques could result in one longer path and one shorter path. The longer path is truncated back to the original path length and the shorter path is extended by performing crossover again and then truncating.

Two types of mutation methods [14] are used for flight path evolution; see Fig. 4. They follow a greedy approach with the hope that small positive changes to the path will lead to larger positive changes to the path. First we randomly select a node in the flight path and see if the next two nodes along the path would form an L shape with this node or a straight line (these are the only two possibilities). In the first case, method 1 ("flip") is used and the algorithm replaces the middle node with the node that mirrors the middle node

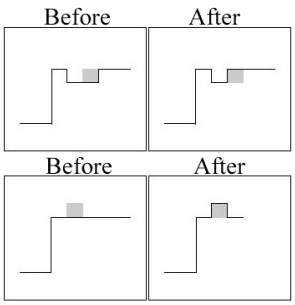

Fig. 4. Examples of mutations in EA-DIR and EA-Path algorithms. (Upper row: method 1. Lower row: method 2)

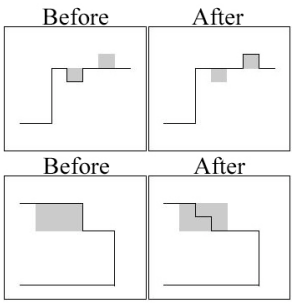

Fig. 5. Examples of mutations in EA-Path_E algorithm. (Upper row: method 2. Lower row: method 3)

if we connect the first node and the third node with a line. This is like flipping a section of the path. In the second case, method 2 ("pull") is used and the algorithm inserts two nodes into the path on one side of the line next to the first and the second nodes. This effectively extends the path by two nodes, so we simply truncate the last two nodes from the path. This is like pulling a string from the middle when the beginning end of the string is fixed. Which side to select for insertion depends on whether the new path is a valid path. If both sides allow valid paths, then the algorithm prefers inserting nodes that are not already in the path. Random selection is the last tie-breaker. If all four nodes on either side of the line are already included in the path, then a new mutation point is randomly selected and the same procedure repeats.

We use an initial population of 100 paths including various paths generated using other algorithms and 95 randomly generated paths. LHC-GW-PF is not used because it is too slow. Other parameters include replacement rate at 30% and mutation rate at 50%. The best three paths are always kept in each iteration. The algorithm runs for at least 500 iterations and stops if either the best path does not improve after 200 iterations or if the algorithm has completed 1000 iterations.

B. Algorithms with a Set Destination

In WiSAR, an operator might prefer the path to end at a specific destination node to support UAV retrieval, persistent visualization of a specific region at a specific time, or planning multiple path segments that make up a longer path. The following algorithms are modified versions from the previous section to handle the additional requirement. We simply add "_E" to the algorithm names to distinguish them.

1) Complete-coverage Algorithm (CC_E): This algorithm is identical to the CC algorithm up to the time when the remaining flight time is just enough to fly the UAV to the end node, then it flies toward the end node using the LHC-GW-CONV_E algorithm (discussed shortly).

2) Local Hill Climbing Algorithms: The LHC-GW-CONV_E and LHC-GW-PF_E algorithms have an additional constraint where nodes that prevent the path from reaching the end node within the remaining time will not be selected.

3) Evolutionary Algorithm: The direction representation of a path does not work with a set destination, so the EA-Path_E algorithm also uses a sequence of node positions to encode the path. Here we increased mutation rate to 90% to force more exploration of the state space. The initial

population of 100 paths includes various paths generated using other algorithms as seeds (both from start node to end node and reversed) and 90 randomly generated paths.

The EA-Path_E algorithm uses both single-point and double-point crossover. The difference is that when the child path is too long, the algorithm truncates the path to the original path length, then backtracks the path until the distance between the end of the child path and the desired end node matches the remaining time. The LHC-GW-CONV_E algorithm is then used to complete the path with the desired end node. If the child path is too short, the LHC-GW-CONV_E algorithm is used to complete the path.

The EA-Path_E algorithm uses three types of mutation methods. First, we randomly select a node in the path and see if the next two nodes along the path would form an L shape with this node or a straight line. In the first case, method 1 ("flip") is used (identical to the one used in the EA-Path algorithm); see Fig. 4. If the nodes form a straight line, then method 2 ("pull") or 3 ("shake") is selected with equal probabilities; see Fig. 5.

Mutation method 2 ("pull") is a modified version from the EA-Path algorithm. This method does not truncate two nodes at the end of the path; instead, it deletes two nodes in the middle of the path. This is like pulling a string from the middle when both ends of the string are fixed.

Mutation method 3 ("shake") works by first marking a small mid-section in the path (to keep it short, we set it to 6 nodes). We first randomly select a node in the path, then traverse the path and find the fifth node down the path. If the path between these two nodes is not a straight line, the method replaces the mid-section with random flying while maintaining the same length for the mid-section. This is similar to shaking a chain where the beginning and ending points remain fixed but the middle section shifts.

V. Experimental Results and Analysis

A. Performance Metrics

We use *Efficiency*, $Efficiency_{LB}$ and Running Time as metrics to measure the performance of the algorithms, where *Efficiency* is calculated if we know what's the best possible and $Efficiency_{LB}$ is used as an estimation when we have no way of calculating the best possible. Sorting all the probability nodes by their values in descending order would generate a list $\{N_1, N_2, N_3, ..., N_{3600}\}$. For the best possible path p^*, the probability accumulated PC_{p^*} is constrained by a theoretical upper bound B:

$$PC_{p^*} \leq \sum_{n=1}^{T+1-d} N_n = B, \tag{6}$$

where d is the distance from the start node to the closest non-zero valued node. Then for any path p_k, we define *Efficiency* and $Efficiency_{LB}$ as the following:

$$Efficiency = \frac{PC_{p_k}}{PC_{p^*}} \tag{7}$$

$$Efficiency_{LB} = \frac{PC_{p_k}}{B} \tag{8}$$

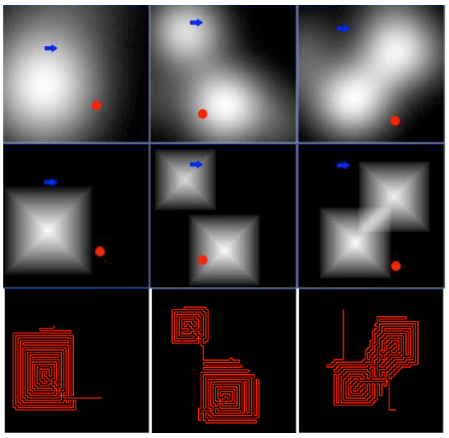

Fig. 6. Top row: 2D representations of unimodal, bimodal, and bimodal with overlap probability distribution maps. Middle row: Simplified versions of the three types of maps. Bottom row: Best paths found for each map.

PC_{p_k} can be calculated using (4). *Efficiency* can be calculated when PC_{p^*} is known and $Efficiency_{LB}$ can be calculated anytime. Clearly, $Efficiency_{LB} \leq Efficiency$.

For example, a path with 95% *Efficiency* means the amount of probability accumulated following this path is 95% of the maximum possible. A path with 85% $Efficiency_{LB}$ means the probability accumulated is 85% of the maximum amount possible if the UAV can teleport from node to node, and the true *Efficiency* could be much higher.

All experiments are run on a Dual-core AMD 3800+ PC with 1GB of memory. For each algorithm, running time is recorded so we can compare algorithm speed.

B. Typical WiSAR Scenarios

In our experiments, we focus on probability distribution maps of three abstract but representative WiSAR scenarios: unimodal, bimodal, and bimodal with overlap. The top row of Fig. 6 shows the 2D representations where each pixel is a probability node; the lighter the pixel, the higher the probability value. The middle row shows three simplified versions of the distributions, which can be used to manually identify the best path possible for each map and compute PC_{p^*}. Then we can measure the true *Efficiency* of paths generated. The blue arrows on the maps mark the starting node (possible location for a WiSAR command center) and the red dots mark the ending node (intentionally selected at a different region from the starting nodes). The bottom row shows the best paths generated for the real maps at T=900.

C. Experimental Results and Analysis

For each distribution type (real and simplified maps) we ran each algorithm (with or without set destination) using T=120, 300, and 900 (4, 10, and 30 minutes). Because of random factors, we ran each experiment 10 times and calculated mean and standard deviation of the results. Due to space limitation, only a subset of the experimental results are presented (e.g. Table I, II and Figure 7–9).

For all the experiments we performed, algorithm running time exhibited the same trend: from the fastest to the slowest we have LHC-GW-CONV(_E), EA(_E) and LHC-GW-PF(_E). For example, with the simplified unimodal map, the LHC-GW-PF algorithm ran for 9.419, 41.952

(%)	Simplified (*Efficiency*)			Real ($Efficiency_{LB}$)		
T	120	300	900	120	300	900
LHC-GW-CONV	88.89	96.80	98.35	81.64	93.97	97.75
LHC-GW-PF	96.63	96.70	96.07	90.28	92.43	96.67
EA-Dir	98.59	97.31	98.80	90.62	94.96	97.96
EA-Path	98.66	98.09	99.07	91.18	95.71	98.02

TABLE I

ALGORITHM EFFICIENCY COMPARISON FOR BIMODAL DISTRIBUTION

(seconds)	Simplified			Real		
T	120	300	900	120	300	900
LHC-GW-CONV	0.90	2.26	7.35	0.52	1.16	5.66
LHC-GW-PF	9.44	29.11	131.35	2.61	8.64	92.38
EA-Dir	9.36	15.56	41.71	10.97	16.69	35.11
EA-Path	10.63	22.89	66.31	12.61	21.20	53.73

TABLE II

ALGORITHM SPEED COMPARISON FOR BIMODAL DISTRIBUTION

and 164.383 seconds for T=120, 300 and 900 respectively. Because the EA(_E) algorithms use the path generated from other algorithms as seeds in the initial population, they are generally slower. However, most of the running time is spent generating the initial population and the evolutionary part of these algorithms only takes a fraction of a second. LHC-GW-PF(_E) algorithms are always the slowest, and that is why we do not include them as seeds in the EA algorithms. For the group of algorithms with set destination, we perform path planning both from the starting node to the ending node and also from the ending node to the starting node (then reverse the path), and then select the better one; we include both runs when we record the algorithm running time. Therefore, the "_E" algorithms always take more time to complete compared to the version before modification.

For the simplified unimodal map, the LHC-GW-CONV(_E) algorithms are the clear winners in each respective group if we consider both the *Efficiency* and the running time. For the group of algorithms without set destination, all algorithms gave above 99.5% *Efficiency*. The LHC-GW-CONV algorithm is always the fastest (e.g. 6.483 seconds for T=900) and achieved 100% *Efficiency* in all cases. The EA-Dir and EA-Path algorithms also achieved 100% *Efficiency*, but at a much slower speed (e.g. 62.236 seconds for T=900 with EA-Path). For the group of algorithms with set destination, the LHC-GW-CONV_E algorithm is also the fastest (e.g. 14.173 seconds for T=900) and achieved 99.955% or higher *Efficiency* in all cases. Although the EA-Path_E algorithm achieved slightly better *Efficiency* (less than 0.1% improvements), it did so at the cost of more running time (e.g. 78.334 seconds for T=900).

For the simplified bimodal map, the LHC-GW-CONV(_E) algorithms did not always perform well because it doesn't handle the space between the two modes very well, especially for very short flight time. Fig. 7 shows the *Efficiency* comparison of the group of algorithms without set destination. The LHC-GW-PF(_E) algorithms still achieved 96% and above *Efficiencies*, but they are also the slowest. The EA(_E) algorithms are more attractive in this case because they achieved the best *Efficiencies* (98.095%+ for EA and 97.857%+ for EA_E) very quickly.

For the simplified bimodal with overlap map, the EA(_E) algorithms achieved the best *Efficiencies* (98.302%+ for EA and 98.653%+ for EA_E), but the LHC-GW-CONV(_E)

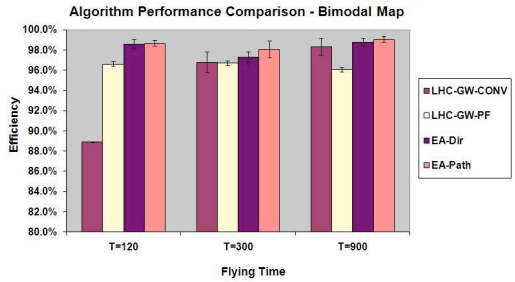

Fig. 7. *Efficiency* comparison for group of algorithms without set destination for simplified bimodal map

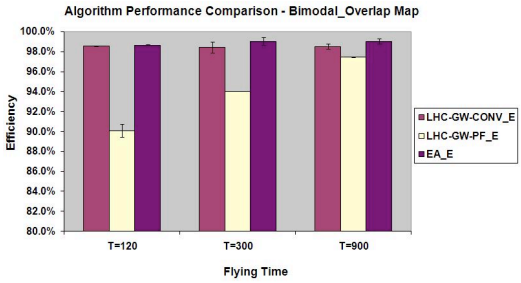

Fig. 8. *Efficiency* comparison for group of algorithms with set destination for simplified bimodal with overlap map

algorithms were able to achieve equivalent or slightly lower Efficiencies (97.391%+ for LHC-GW-CONV and 98.429%+ for LHC-GW-CONV_E) with much less time (8.283 seconds and 16.296 seconds for T=900 respectively). Fig. 8 shows the *Efficiency* comparison of the group of algorithms with set destination.

For each of the three real distribution maps (unimodal, bimodal, and bimodal with overlap), since PC_{p^*} is unknown, we can only calculate $Efficiency_{LB}$. We observed that the $Efficiency_{LB}$ for each real map is very close to the $Efficiency_{LB}$ for each of the counterpart simplified maps, and we hypothesize that the *Efficiency* for each real map should also be close to the *Efficiency* for each of the counterpart simplified maps. Fig. 9 shows an example of the EA-Path algorithm performance for the real and simplified bimodal with overlap map. The columns in the front row are $Efficiency_{LB}$ values and the columns in the back row are *Efficiency* values. Based on this graph, we estimate that the *Efficiency* values for the real map here are above 97% for all T values.

To further evaluate our algorithms, we tested our algorithms on a more complex multimodal distribution map gen-

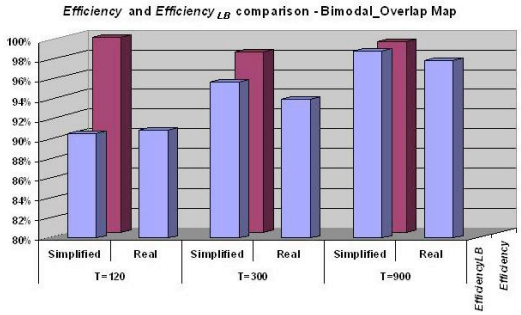

Fig. 9. EA-Path performance for the real and simplified bimodal with overlap map

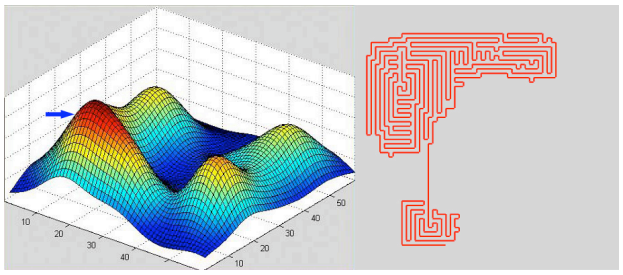

Fig. 10. More complex multimodal probability distribution map

erated by mixing multiple Gaussian distributions with various standard deviations; see Fig. 10. The LHC-GW-CONV algorithm achieved 97.206% $Efficiency_{LB}$ in 5.516 seconds and the EA-Path algorithm achieved 97.609% $Efficiency_{LB}$ in 63.984 seconds. Note here that the $Efficiency$ percentiles can only be better.

In every experiment, the EA(_E) algorithms always achieved the best $Efficiency$ and $Efficiency_{LB}$. Therefore, if the operator has some time for computation, they seem to be attractive candidates. If the operator needs a path generated quickly, the LHC-GW-CONV(_E) algorithms can be used. Although the LHC-GW-PF(_E) algorithms do not work as well with these three distribution maps, initial tests on other distribution types such as sparse map and small-multimodal map suggest that they could perform better than other algorithms.

VI. Conclusion and Future Work

We model the UAV path planning problem in WiSAR as a discretized combinatorial optimization problem and design two groups of algorithms for path planning with or without a set destination using algorithms based on Local Hill Climbing, and Evolutionary Algorithms using novel techniques such as "global warming effect" and path crossover/mutation. We evaluate the performances of these algorithms on six (3 simplified, 3 "real") representations of typical WiSAR probability distribution maps, unimodal, bimodal, and bimodal with overlap, with various flight times and use the simplified maps to validate true efficiencies in real maps. Experimental results show that our algorithms can generate good paths with high $Efficiency$ or estimated $Efficiency$ that approximate the optimal solution within reasonable computation time. Specifically, the LHC-GW-CONV(_E) algorithms should be used for unimodal maps, and if a few minutes computation time is available, because the EA(_E) algorithms always keep the best path found from seed algorithms, they can always find a path with the highest $Efficiency$ compared with other algorithms experimented.

Experimenting with more types of distribution maps, designing a more advanced global warming search model, allowing 8-connected path planning, and dealing with dynamic distribution maps that change over time are all natural extensions for future work. Specifically, the set of algorithms with set destinations enables us to further investigate how the path planning task can be segmented so human operators can plan more strategically while the algorithms plan tactically, and what interface can make this an intuitive, smooth, and effective task for the UAV operator in WiSAR operations.

VII. Acknowledgements

This work was partially supported by the National Science Foundation under grant number 0534736 and by a grant from the Army Research Laboratory. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsoring organizations.

References

[1] R. Beard, D. Kingston, M. Quigley, D. Snyder, R. Christiansen, W. Johnson, T. McLain, and M. A. Goodrich. Autonomous vehicle technologies for small fixed wing UAVs. *Journal of Aerospace Computing, Information, and Communication 2005*, 2(1):92–108, January 2005.

[2] F. Bourgault, A. Chokshi, and M. Compbell. Human-computer augmented nodes for scalable mobile sensor networks. In *Proceedings of the SMC DHMS 2008*, 2008.

[3] F. Bourgault, T. Furukawa, and H. F. Durrant-Whyte. Optimal search for a lost target in a Bayesian world. In S. Yuta, H. Asama, S. Thrun, E. Prassler, and T. Tsubouchi, editors, *FSR*, volume 24 of *Springer Tracts in Advanced Robotics*, pages 209–222. Springer, 2003.

[4] D. Ferguson. GIS for wilderness search and rescue. In *ESRI Federal User Conference*, February 2008.

[5] M. Fischetti. Solving the Orienteering Problem through branch-and-cut. *INFORMS Journal on Computing*, 10(2):133–148, Feb 1998.

[6] M. A. Goodrich, B. S. Morse, D. Gerhardt, J. L. Cooper, M. Quigley, J. A. Adams, and C. Humphrey. Supporting wilderness search and rescue using a camera-equipped mini UAV. *Journal of Field Robotics*, 25(1-2):89–110, January 2008.

[7] G. Gutin and A. P. Punnen. *The Traveling Salesman Problem and Its Variations*. Kluwer Academic Publishiers, 2002.

[8] S. R. Hansen, T. W. McLain, and M. A. Goodrich. Probabilistic searching using a small Unmanned Aerial Vehicle. In *AIAA Infotech@Aerospace*, number AIAA-2007-2740, 2007.

[9] J. K. Howlett, T. W. McLain, and M. A. Goodrich. Learning Real-Time A* path planner for Unmanned Air Vehicle target sensing. *Journal of Aerospace Computing, Information, and Communication*, 3(3):108–122, 2006.

[10] B. O. Koopman. *Search and Screening: General Principles with Historical Applications*. Pergamon Press, 1980.

[11] G. Laporte and S. Martello. The selective Travelling Salesman Problem. *DISCRETE APPL. MATH.*, 26(2-3):193–207, 1990.

[12] Y.-C. Liang and A. E. Smith. An Ant Colony approach to the Orienteering Problem. *Journal of the Chinese Institute of Industrial Engineers*, 23(5):403–414, 2006.

[13] L. Lin and M. A. Goodrich. A Bayesian approach to modeling lost person behaviors based on terrain features in wilderness search and rescue. In *Proceedings of the 18th BRIMS*, Sundance, Utah, March 2009.

[14] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[15] J. Mittenthal and C. E. Noon. An insert/delete heuristic for the Travelling Salesman Subset-Tour problem with one additional constraint. *The Journal of the Operational Research Society*, 43(3):277–283, 1992.

[16] P. Pettersson and P. Doherty. Probabilistic roadmap based path planning for an autonomous unmanned helicopter. *Journal of Intelligent and Fuzzy Systems*, 17(4):395–405, Sep 2006.

[17] M. Quigley, B. Barber, S. Griffiths, and M. A. Goodrich. Towards real-world searching with fixed-wing mini-UAVs. In *Proceedings of IROS*, 2005.

[18] R. Ramesh, Y.-S. Yoon, and M. H. Karwan. An optimal algorithm for the Orienteering Tour Problem. *ORSA Journal on Computing*, 4(2), Spring 1992.

[19] T. J. Setnicka and K. Andrasko. *Wilderness Search and Rescue*. Appalachian Mountain Club, 1980.

[20] P. R. Sokkappa. *The Cost-Constrained Traveling Salesman Problem*. Doctoral dissertation, University of California, October 1990.

[21] E. Soylemez and N. Usul. Utility of GIS in search and rescue operations. In *ESRI Users Group Conference*, September 2006.

[22] W. G. Syrotuck. *An Introduction to Land Search Probabilities and Calculations*. Barkleigh Productions, Mechanicsburg, PA, 2000.

[23] M. F. Tasgetiren and A. E. Smith. A Genetic Algorithm for the Orienteering Problem. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 2, pages 910–915, 2000.