Communication Relay for Unmanned Aerial Vehicles in Autonomous Search and Rescue Missions*

Dongyu Zheng[1], Ian Yuan[2], Howard McLauchlan[3], and Jae Han Chung[4]

Abstract— **The application of swarming without position alleviates the need for sensors which are dependent on the environment and generally expensive in terms of energy, cost, size, and weight. This principle can be applied to search and rescue situations where target locations are unknown, in which autonomous unmanned aerial vehicles can form ad-hoc wireless communication networks to facilitate contact between ground users in lieu of damaged, nonexistent, or congested networks.**

To approach the difficult task of efficiently deploying drones for search and rescue applications, a variety of optimization techniques are explored and evaluated on a simplified, but sufficient formulation of the problem. These techniques include iterative deepening search, tabu search, simulated annealing, simple genetic algorithm, and ant colony optimization. It is noted that as input (maps modeling a variety of situations) increases in size and complexity, techniques such as simple genetic algorithm and ant colony optimization which have comparatively good asymptotic runtimes significantly outperform other approaches.

I. INTRODUCTION

In search and rescue situations where target locations are unknown, autonomous unmanned aerial vehicles (UAVs) can form ad-hoc wireless communication networks to facilitate contact between ground users. This approach enables the supplementation and replacement of damaged, nonexistent, or congested networks, and can contribute substantially to disaster mitigation efforts. The aerial nature of this application also facilitates more energy-efficient line of sight transmissions and is unrestricted by difficult terrain. Furthermore, the application of swarming without position allows the aerial vehicles to function adequately with only proprioceptive sensors, greatly reducing cost and increasing practicability.

The objective of this project is to effectively simulate the aforementioned application of UAVs to search and rescue scenarios. Specifically, we first transform the problem into a simplified, but sufficient formulation, after which we explore the utility of several classes of optimization techniques with respect to the problem. These techniques include iterative deepening search, tabu search, simulated annealing, simple genetic algorithm, and ant colony optimization. In testing, the most promising algorithms proved to be simple genetic algorithm and ant colony optimization. Since these two techniques have several tunable hyperparameters, simple genetic algorithm was applied to optimize these hyperparameters, further improving the quality of our solutions. It is concluded that, out of the explored methods, simple genetic algorithm and ant colony optimization yield the best results for this problem.

II. RELATED WORK

Presently, there exists no deterministic methodology to design the local interactions necessary in order to manifest the desired swarm behavior. Current state-of-the-art literature [1] suggests an approach inspired by army ant colonies, which utilize pheromones to direct other ants to food sources. This is analogously applied to the deployment of aerial vehicles in creating and maintaining communication networks. Similar works based on the trophallactic behavior of honeybees [2], [3], [4], [5] and foraging of ant colonies [6] have demonstrated substantial utility in ground robot foraging.

In the approach inspired by army ants, the micro aerial vehicles (MAVs, for brevity) are equipped with only proprioceptive sensors and do not rely on global positioning data. Here, [1] makes a distinction between Node MAVs and Ant MAVs: the former serves to guide Ant MAVs by storing relevant pheromone-direction data, while the latter traverses the already explored space (occupied by a network of Node MAVs) and expands the frontiers of that space. Ant MAVs then become Node MAVs, extending the communication network and allowing subsequent Ant MAVs to explore further. The aforementioned virtual pheromones play a key role in directing Ant MAVs down heuristically favorable paths. These pheromones are adjusted by various actors to maintain the utility of the heuristic.

Our solution extends the army ants approach, leveraging the analogous structures of ant pheromone networks and drone networks. We do not make the distinction between Node and Ant MAVs, opting instead to treat them all homogeneously to allow greater flexibility in solution generation.

III. PROBLEM FORMULATION

The problem begins with a predefined physical space where UAVs will be deployed in order to conduct search and rescue. Defining the physical space in terms of three-dimensional coordinates, down the granularity of some unit of length, the space can then be constructed by a number of unit cubes, shown in Fig. 1. We reduce the problem to two dimensions (Fig. 2). For this problem, an algorithmic

*This paper is submitted as part of the University of Waterloo's Cooperative and Adaptive Algorithms (ECE457A) course project, taught by Dr. Allaa Hilal `ahilal at uwaterloo.ca`

[1]Dongyu Zheng is a 4A Software Engineering student at the University of Waterloo `d28zheng at edu.uwaterloo.ca`

[2]Ian Yuan is a 4A Computer Engineering student at the University of Waterloo `iyyuan at edu.uwaterloo.ca`

[3]Howard McLauchlan is a 4A Software Engineering student at the University of Waterloo `h96wu at edu.uwaterloo.ca`

[4]Jae Han Chung is a 4A Computer Engineering student at the University of Waterloo `jhj2chun at edu.uwaterloo.ca`

approach for 2D is easily extensible to 3D with additional compute. Table I shows how a state in the search space is encoded.

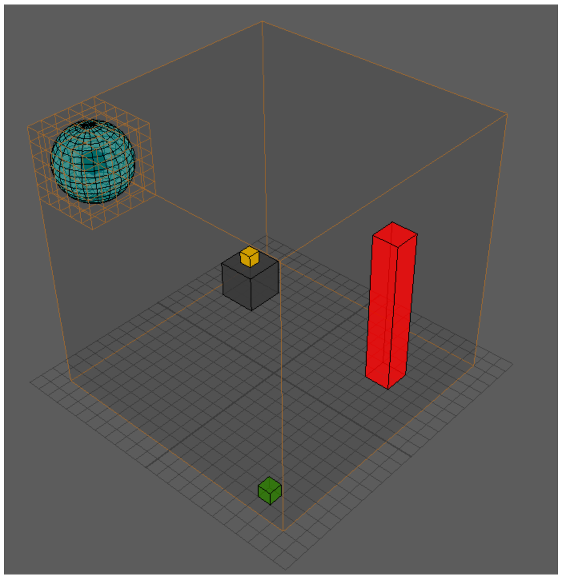

Fig. 1. Physical space representation in 3D: teal is a drone communication radius, gold is a signal relay, grey is a building, red is an obstacle, green is a target.

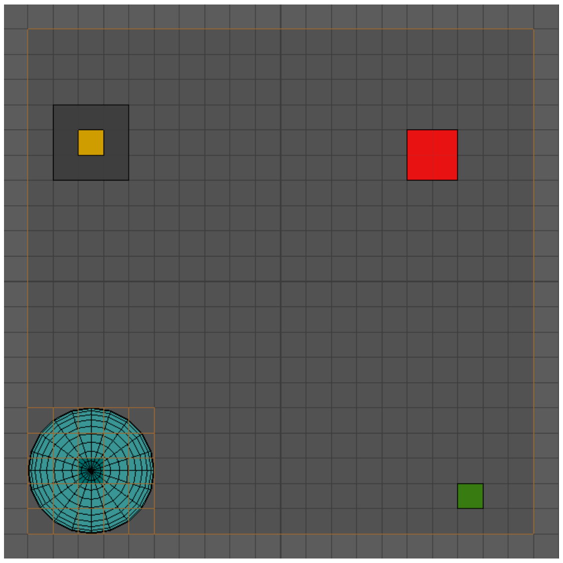

Fig. 2. Physical space representation in 2D.

TABLE I
INFORMATION ENCODED BY A STATE

Information Encoded	Data Type	Initial Value
Number of drones deployed	Integer	D = 0
Number of targets remaining	Integer	T >= 1
Location of drones, relays, targets, and unnavigable terrain	2D Array	Grid filled with relays, targets, and unnavigable terrain

We make the following base assumptions:

- A drone can hover in a relatively fixed location.
- A drone can communicate with nearby drones and relays in a communication radius of C units.
- A drone can detect a target in a radius of S units.
- Drones do not suddenly fail.
- The number of targets, T, is known.

For this problem, a goal state is reached when all targets have been found, i.e., when the number of targets remaining reaches zero.

In each step/move, one of several actions may occur:

- Add a drone to a *legal coordinate* space.
- Move a drone to a *legal coordinate* space, if it is a *valid operation*.
- Remove a drone, if it is a *valid operation*.

A *legal coordinate* is defined as a navigable space within communication range of at least one other drone or base relay. A *valid operation* is defined as the movement or removal of a drone such that it does not create orphaned drone(s) with have no neighboring drones or relays within communication radius. Thus, the neighborhood operator is defined to be the addition, move, or removal of a drone from a state, generating the next state.

With this information, a graph can be used to represent the communication network between relays, drones, and targets, where edges are the communication links formed between nodes, and weights are the Euclidean distance between nodes.

The cost of a state is then determined by the function

$$C(E, D, T) = \alpha E + \beta D + \gamma T \tag{1}$$

where E is the *sum of edges*, D is the number of drones deployed, T is the number of targets remaining, and α, β, γ are weights (lower C is better). The *sum of edges* is defined as the total sum of the edge weights in the network.

An algorithm itself is evaluated using

$$E(t, D, T) = \zeta t + \eta D + \theta T \tag{2}$$

where t is the number of iterations before algorithm termination, D is the maximum number of drones deployed, T is the number of targets remaining, and ζ, η, θ are weights (lower E is better).

IV. Proposed Solutions

We investigated the following techniques:

- Iterative deepening search
- Tabu search
- Simulated annealing
- Simple genetic algorithm
- Ant colony optimization

A. *Iterative Deepening Search*

Iterative deepening search (IDS) was chosen over breadth-first-search and depth-first-search since it performs better than both in the average case. IDS also avoids the immense memory requirements of breadth-first-search as the search space increases.

However, the problem with an uninformed IDS is that it still performs poorly in a large search space. We attempted to resolve this problem by introducing a simple heuristic that encourages drone placement at the frontier of exploration—that is to say, to explore areas that have not been previously explored. A maximum depth limit d is kept constant. Additionally, a constant branching factor β was introduced to help control memory usage. While IDS is complete and optimal in a search space where moves are of uniform cost, this is not the case with the introduction of d and β:

- The algorithm is incomplete due to an imposed max depth limitation, d, and branching factor, β.
- The algorithm is suboptimal due to the introduction of branching factor, β.

The algorithm terminates when all targets have been found or when the max depth limit is reached.

B. *Tabu Search*

Tabu search (TS) is a trajectory method which leverages memory structures to encourage a balance of intensification and diversification in an attempt to converge on a global optima [7]. A solution in TS is equivalent to some state in the search space such that all targets have been found.

The memory structures used in TS are:

- Short-term memory—used to keep track of the tenure on each position in the grid. This is used to prevent repetitive removals or movements of a drone from/to a position on the grid.
- Long-term memory—used to keep track of the number of iterations since finding a state better than the best state so far. Upon exceeding some predefined number of iterations, we perform *restart diversification* by jumping back to the initial state.

The algorithm begins with a random feasible solution state. Using the neighborhood operator defined in Section III, the algorithm generates a set of N neighboring states. This set of states is sorted based on their cost using (1). The algorithm proceeds to iterate through this set of neighboring states, where:

- If the changes from the current state to this neighboring state is not in the tabu list or it meets the *aspiration criteria*, visit this new state and add the relevant changes in drone positions to short-term memory with a random tenure, $0.75\sqrt{n} \leq tenure \leq 1.25\sqrt{n}$, where n is proportional to the problem size; update best state so far if applicable.
- Otherwise, we consider the next state after this one.

The *aspiration criteria* is met when the cost of a neighboring state is lower than the cost of the best state so far. This is referred to as *default aspiration*.

Termination occurs once the algorithm has found there to be no improvement in state cost after a constant number of iterations, or after a fixed number of iterations in total. Upon termination, the algorithm returns the best state found throughout its execution as the solution.

C. *Simulated Annealing*

Similar to TS, simulated annealing (SA) is a trajectory-based method which uses randomness to escape local optima [8]. SA keeps track of the best solution state so far throughout its execution and returns it when the termination condition is met. A solution in SA is equivalent to some state in the search space such that all targets have been found.

The current temperature, T, is initially set to some fixed constant, $T_{initial} > 0$. An initial state is first randomly generated and subsequently evaluated, yielding a cost of C using (1). Using the neighborhood operator defined in Section III, the algorithm iterates through a set of generated neighboring states, where the following occurs:

- If a neighboring state has a lower cost than the best state so far, the algorithm will visit this neighbor and set it as the best state so far.
- Otherwise, the algorithm *may* visit this neighbor based on a probability function, defined in (3), setting the best state so far if applicable.

$$p_{acceptance} = e^{-\Delta C/T} \tag{3}$$

where ΔC is the change in cost between the new state and the current best state.

This probability function allows for more frequent acceptances of worse solutions at higher temperatures, thus promoting diversification earlier in the algorithm's execution. Conversely, this function tends to reject more worse solutions at lower temperatures. After X number of iterations, the current temperature decreases according to the cooling schedule, which uses a geometric temperature decrement rule (4).

$$T = T \times \alpha \tag{4}$$

where $0 < \alpha < 1$.

The algorithm terminates once the current temperature is lower than some defined threshold, $0 \leq T_{final} < T_{initial}$, and returns the best state found throughout its execution as the solution.

D. *Simple Genetic Algorithm*

Simple genetic algorithm (SGA) can be applied to the problem by iteratively improving upon the performance of the solution through fitness selection on a multi-generational basis [9]. SGA keeps track of the best solution throughout the course of multiple generations and returns the best solution found once termination conditions are met.

An initial population of N candidate solutions is first randomly generated, where N denotes the population size for each generation of solutions. A candidate solution is encoded as a list of paths from each relay, $r_1, r_2, ..., r_a$, to each target, $t_1, t_2, ..., t_b$, where a, b denote the number of relays and targets, respectively. The current population pool is comprised of the parent solutions that are used to produce offspring solutions in the next generation. The population is then randomly shuffled. Each pair of consecutive solutions will undergo crossover at a fixed probability p_c producing two new children, c_1 and c_2 respectively. If crossover does not occur, the children are direct copies of the parents. These children, c_1 and c_2, will then undergo mutation, where each path in the offspring solution may be replaced with a new randomly generated path at a probability p_m, and are added to the offspring pool. *Elitism* is also applied, where the best unmodified parent, in terms of *fitness*, is directly added to the offspring pool.

Once crossover and mutation have been performed, each offspring is evaluated based on its *fitness*. A solution's *fitness* is defined as follows:

- Partition the paths in the solution into $P_1, P_2, ..., P_b$ such that P_i is a set of paths from any relay to target t_i
- Let S be $P_1 \times P_2 \times ... \times P_b$, where $\times$ denotes the Cartesian product
- Then $s_i \in S$ is a set of paths that finds each target exactly once
- The fitness is then the minimum cardinality of the set union of the nodes traversed over $s_i \in S$
- The $s_i \in S$ producing the minimum cardinality is the actual solution of this candidate solution, where drones are placed

Only the top N offspring are selected as the next generation of solutions and the rest are discarded. Every subsequent generation will also undergo crossover and mutation, breeding new generations of solutions until termination conditions are met. The termination of the algorithm occurs when the maximum number of generations is reached or there is no improvement in the best solution after a fixed number of generations. The final solution, then, consists of the minimal distance paths from each relay to each target, sharing the maximum number of nodes traversed. For visualization purposes, drones are then greedily placed on the best solution.

E. Ant Colony Optimization

Ant colony optimization (ACO) is well suited for our problem since a drone has several fundamental similarities with an ant in the context of pathfinding. Artificial pheromones serve as the collective memory of drones, allowing each generation to benefit from all previous generations' findings [10]. Thus, an optimal drone placement can be reached by utilizing the pheromones left behind by several generations of ants. Similar to SGA, a solution is defined as a list of paths from each relay, $r_1, r_2, ..., r_a$, to each target, $t_1, t_2, ..., t_b$, where a, b denote the number of relays and targets respectively. The transition rule is defined as follows: at node i, an ant takes the edge ij with probability p_{ij}, where p_{ij} is defined as

$$p_{ij} = \begin{cases} \frac{\tau_{ij}^{\alpha}/d_{ij}^{\beta}}{\sum_{n \in N_i} \tau_{in}^{\alpha}/d_{in}^{\beta}} & if \quad j \in N_i \\ 0 & if \quad j \notin N_i \end{cases} \tag{5}$$

where τ_{ij} denotes the amount of pheromone on edge ij, α, β are values chosen to balance the local vs. global search ability of an ant, and d denotes the edge weight [10].

However, within the scope of this problem, all edges are equally weighted. Setting edge weights to $d = 1$, α, β become irrelevant and can be set to $\alpha = \beta = 1$, simplifying (5) to

$$p_{ij} = \begin{cases} \frac{\tau_{ij}}{\sum_{n \in N_i} \tau_{in}} & if \quad j \in N_i \\ 0 & if \quad j \notin N_i \end{cases} \tag{6}$$

The pheromone evaporation and update rule is defined as follows: all pheromones decay by a factor of $1 - \rho$ per iteration, and only pheromones from the best solution is deposited (*elitism*). The equation is shown in (7):

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \rho \Delta \tau_{ij}^{best}(t) \tag{7}$$

where $\tau_{ij}(t)$ is the pheromone amount on edge ij at iteration t, and $\Delta \tau_{ij}^{best}(t)$ is defined as

$$\Delta \tau_{ij}^{best}(t) = \begin{cases} Q/L^{best} & if \quad edge_{ij} \in best\ solution \\ 0 & otherwise \end{cases} \tag{8}$$

where Q is some constant and L^{best} is the cardinality of the set of nodes traversed in the best solution [10].

The quality of a solution is determined using the same fitness function described under SGA. The termination of the algorithm occurs once the max number of iterations has been reached or when there has been no improvement over the best solution so far for a fixed number of iterations. Once the termination condition has been met, the resulting best final solution consists of the minimal distance paths from each relay to each target, sharing the maximum number of nodes traversed. For visualization purposes, drones are then greedily placed on the best solution.

V. Performance Evaluation

Table II summarizes the run-time complexities for all algorithms discussed in this report. Though we report the space complexity for each individual algorithm for completeness, we disregard these values when evaluating performance since memory was not a limiting factor in our experiments even when handling fairly large inputs.

Qualitatively, we observe that our SGA and ACO implementations have a superior asymptotic runtime of $O(N)$ compared to the other approaches. Of the remaining techniques, IDS has the worst asymptotic behavior with a runtime complexity of $O(N^d)$, whereas TS and SA both operate in $O(N^2)$. It is clear from asymptotic analysis that SGA and ACO are theoretically the fastest as input sizes increase, as no other technique can approach a linear runtime. Since our objective was formulated around the application of drones to search and rescue situations where time is critical, it is evident that SGA and ACO are the best fit with project objectives from a qualitative perspective.

TABLE II

TIME AND SPACE COMPLEXITIES OF ALGORITHMS

Algorithm	Time Complexity	Space Complexity
IDS	O(N^d)	**O(d)**
TS	O(N^2)	O(N^2)
SA	O(N^2)	O(N^2)
SGA	**O(N)**	O(N)
ACO	**O(N)**	O(N)

Table III compares the performance of the algorithms discussed in this report. Performance is measured using (2), where lower scores are better. ∞ means the algorithm did not terminate within a reasonable time. The focus of Table III will be on SGA and ACO, as their asymptotic runtimes are linear and will likely perform better than the other algorithms.

Quantitatively, we observe that SGA performed the best out of all the algorithms on map 0, 1, and 2 producing costs of 4010, 13310, and 10800 respectively. This beats IDS, TS, and SA by a significant margin and is only slightly better than the costs produced by ACO.

We also observe that ACO performed the best out of all the algorithms on map 3 producing a cost of 22330. Again, ACO significantly outperforms IDS, TS, and SA by a significant margin and is only slightly better than the cost produced by SGA.

TABLE III

PERFORMANCE COMPARISON

Map	IDS	TS	SA	SGA	ACO
0 (very simple)	5005	5181	5950	**4010**	4870
1 (simple)	19019	22500	21030	**13310**	15230
2 (simple)	19019	32402	18070	**10800**	11720
3 (moderate)	∞	47844	49320	25230	**22330**

Table IV summarizes the pros and cons of each algorithm discussed. We can see that IDS is simple to implement but its runtime complexity is poor. TS and SA both offer a good balance of intensification and diversification, but both also yield suboptimal performance as problems become more complex. SGA and ACO scale extremely well with input size as a result of their asymptotically linear runtimes, but are both much more difficult to implement.

TABLE IV

SUMMARY OF ALGORITHM EVALUATION

Algorithm	Pros	Cons
IDS	• simple to implement • efficient in space complexity	• blind search without heuristic • runtime complexity scales poorly
TS	• good balance of intensification and diversification	• suboptimal performance as maps become more complex
SA	• good balance of intensification and diversification	• suboptimal performance as maps become more complex
SGA	• high efficiency and scalability in time and space	• implementation complexity
ACO	• high efficiency and scalability in time and space	• implementation complexity

From the results in Table II and Table III, it is clear that SGA and ACO had the best performance of all the techniques explored. Thus, for hyperparameter tuning, we chose to focus on tuning SGA and ACO.

A. Hyperparameter Tuning for SGA and ACO

Simple genetic algorithm was applied to tune the hyperparameters of SGA and ACO. The highlighted rows in Table V and Table VI contain the hyperparameters that, on average, outperformed the rest on maps 0 to 3 for SGA and ACO, respectively. These hyperparameters were later used for SGA and ACO on maps 4 to 6 in the next subsection.

It is observed that for SGA, the lowest average cost across maps 0 to 3 was achieved with hyperparameters $p_c = 0.81$ and $p_m = 0.08$, despite the fact that this specific combination did not result in any lowest individual cost for maps 0 to 3.

Furthermore, it is observed that for ACO, the lowest average cost across maps 0 to 3 was achieved with hyperparameters $\tau_{initial} = 0.8$, $\rho = 0.09$, and $Q = 3.7$. This particular set of parameters yielded not only the lowest average cost across maps 0 to 3, but also yielded the lowest costs for maps 2 and 3.

TABLE V

TUNING SGA WITH SGA

P_c	P_m	Map0	Map1	Map2	Map3	Avg. Cost
0.65	0.07	4090	15350	11280	25010	13932
0.65	0.13	**4010**	14760	12100	24930	13950
0.65	0.13	4170	14030	12690	**20800**	12922
0.65	0.13	4220	14550	**10010**	23770	13137
0.65	0.13	4330	13180	11780	22410	12925
0.72	0.08	4790	15100	11210	25520	14155
0.72	0.13	**4010**	14180	10140	23720	13012
0.72	0.13	**4010**	13270	10910	24890	13270
0.72	0.13	4320	14730	11530	24490	13767
0.81	0.07	**4010**	14940	11060	26370	14095
0.81	0.07	4090	14390	11710	25460	13912
0.81	0.08	4080	14190	10830	21090	**12547**
0.81	0.13	**4010**	15650	10590	27240	14372
0.93	0.07	4260	14650	12300	28350	14890
0.93	0.08	4820	**12630**	11210	24420	13270
0.93	0.08	**4010**	14140	10200	22400	12687
0.93	0.08	**4010**	14180	11430	24190	13452
0.93	0.12	4920	13440	10590	24190	13285
0.93	0.13	4030	13900	11920	24100	13487

TABLE VI
TUNING ACO WITH SGA

# Ants	τ_0	ρ	Q	Map0	Map1	Map2	Map3	Avg.
6	0.8	0.09	2.6	4560	14130	10250	25820	13690
13	0.9	0.09	2.6	4110	14390	10910	24240	13412
13	0.9	0.09	2.6	4070	14510	12030	25550	14040
13	0.9	0.09	3.7	4960	15020	12020	24070	14017
13	0.9	0.09	9.4	4050	14300	10260	24990	13400
13	1.3	0.09	2.6	4270	13310	11150	25100	13457
14	0.8	0.09	3.7	4140	14060	**10040**	**22100**	**12585**
14	0.8	0.1	8.7	4670	13540	11200	25080	13622
14	1	0.1	3.7	**4020**	13400	11060	25650	13532
14	1	0.1	8.7	4270	14060	10280	23090	12925
15	0.8	0.09	3.7	4280	**12020**	11620	25140	13265
15	0.8	0.09	8.7	4150	13160	11010	25250	13392
15	1	0.09	3.7	5160	13750	10850	25650	13852
15	1	0.15	5	4730	15090	10850	24760	13857
15	1	0.1	8.7	5030	15940	11150	24360	14120
18	1.2	0.09	9.4	5070	15140	11620	25690	14380
18	1.3	0.09	2.6	4040	13030	10870	24170	13027
18	1.3	0.09	9.4	4050	12860	11250	25080	13310
18	1.3	0.09	9.4	4060	13470	12760	25370	13915

B. *Comparison of Tuned SGA & ACO*

After producing a set of tuned parameters for SGA and ACO that resulted in the lowest average cost across maps 0 to 3, we then applied these parameters and evaluated the performance of SGA and ACO on the more complex maps 4 to 6. Table VII summarizes the performance of all algorithms on maps 0 to 6. We note that for every technique except SGA and ACO, maps 5 and 6 took an unreasonable amount of time to terminate.

Quantitatively, we observe that SGA outperformed ACO in maps 4 and 6. SGA produced costs of 21490 and 16120 respectively, whereas ACO produced 25060 and 17640, respectively. However, we observe that ACO outperformed SGA on the largest map, map 5. ACO produced a cost of 96100 whereas SGA produced a cost of 105020. These results suggest that SGA will typically outperform ACO on smaller maps but that ACO performs better than SGA on bigger maps.

From Table VII, the lowest cost solutions for maps 0 to 6 are shown in the Appendix, in Fig. 3 to 9. Maps 0, 1, 2, 4, and 6 (Fig. 3, 4, 5, 7, and 9, respectively) were completed using SGA. Maps 3 and 5 (Fig. 6 and 8, respectively) were completed using ACO.

TABLE VII
PERFORMANCE COMPARISON

Map	IDS	TS	SA	SGA	ACO
0 (very simple)	5005	5181	5950	**4010**	4870
1 (simple)	19019	22500	21030	**13310**	15230
2 (simple)	19019	32402	18070	**10800**	11720
3 (moderate)	∞	47844	49320	25230	**22330**
4 (moderate)	70720	94770	53690	**21490**	25060
5 (difficult)	∞	∞	∞	105020	**96100**
6 (very difficult)	∞	∞	∞	**16120**	17640

VI. CONCLUSIONS & RECOMMENDATIONS

This report provides insight into the design of drone deployment techniques using minimal sensing for search and rescue applications. A number of optimization techniques (including IDS, TS, SA, SGA, and ACO) were considered, with SGA and ACO demonstrating the greatest utility on large, complex inputs (maps). Specifically, the linear runtime of SGA and ACO when applied to the chosen problem formulation yields a significant advantage over IDS, TS, and SA, resulting in faster termination and lower costs over all inputs of non-trivial complexity.

Future developments will be concerned with extending the problem formulation to three dimensions and assessing the impact of an additional dimension on qualitative and quantitative performance across techniques.

APPENDIX

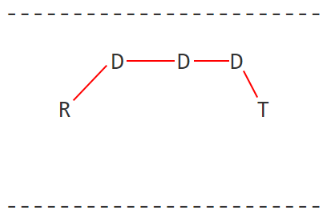

Fig. 3. Simple genetic algorithm solution for map 0.

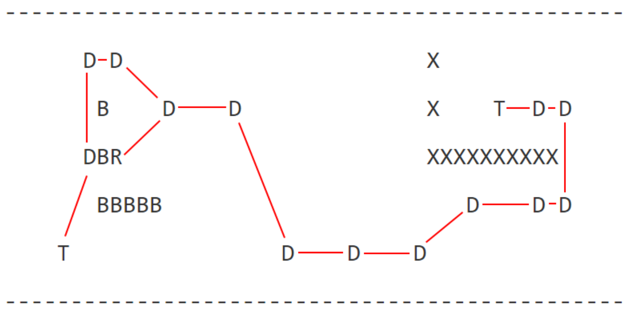

Fig. 4. Simple genetic algorithm solution for map 1.

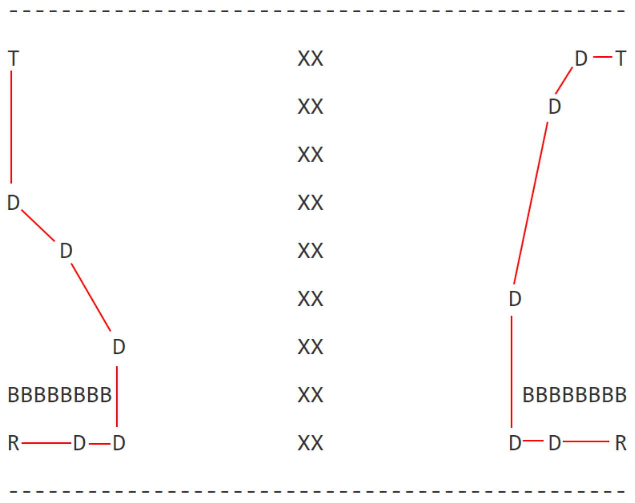

Fig. 5. Simple genetic algorithm solution for map 2.

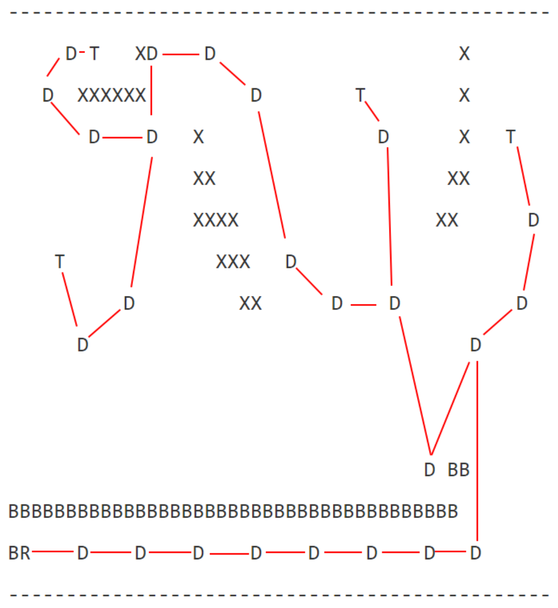

Fig. 6. Ant colony optimization solution for map 3.

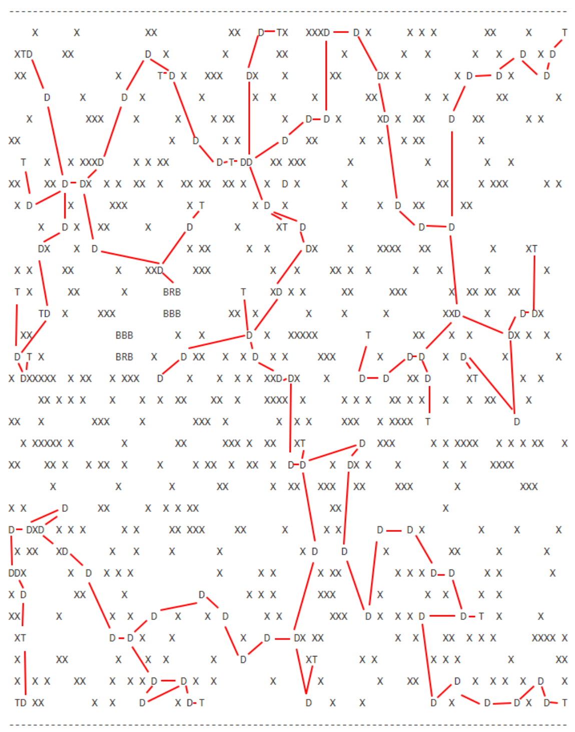

Fig. 8. Ant colony optimization solution for map 5.

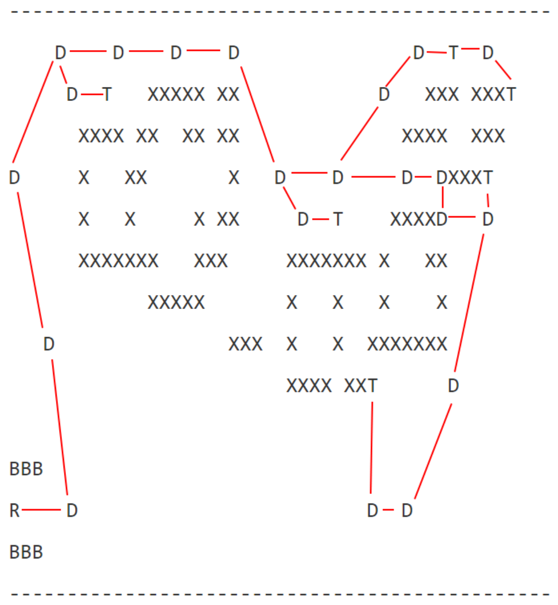

Fig. 7. Simple genetic algorithm solution for map 4.

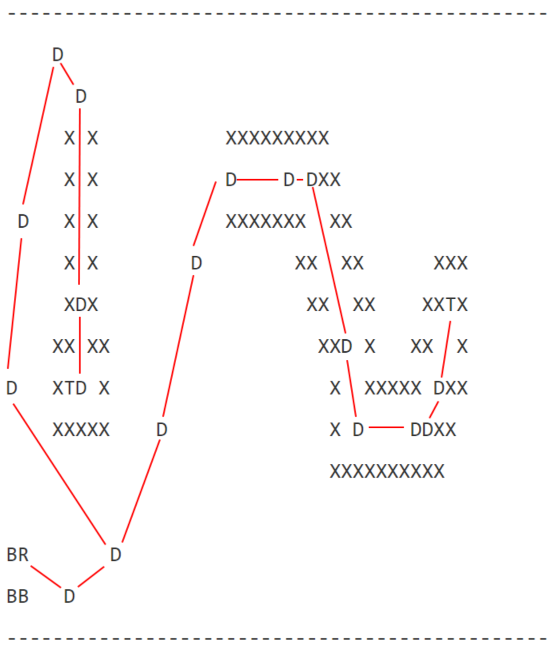

Fig. 9. Simple genetic algorithm solution for map 6.

ACKNOWLEDGMENT

The authors would like to thank Dr. Allaa Hilal for teaching the course, and teaching assistants Pouya Mehrannia and Hossein Soleimani for offering their guidance and support for the project.

REFERENCES

[1] S. Hauert, L. Winkler, J.-C. Zufferey and D. Floreano, ”Ant-based swarming with positionless micro air vehicles for communication relay,” Swarm Intelligence, vol. 2, no. 2-4, pp. 167-188, 2008.
[2] S. Camazine, K. Crailsheim, N. Hrassnigg, G. E. Robinson, B. Leonhard and H. Kropiunigg, ”Protein trophallaxis and the regulation of pollen foraging by honey bees (Apis mellifera L.),” Apidologie, vol. 29, no. 1-2, pp. 113-126, 1998.
[3] K. Crailsheim, ”Trophallactic interactions in the adult honeybee (Apis mellifera L.),” Apidologie, vol. 29, no. 1-2, pp. 97-112, 1998.
[4] T. Schmickl and K. Crailsheim, ”Trophallaxis among swarm-robots: A biologically inspired strategy for swarm robotics,” in The First IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics, 2006. BioRob 2006., Pisa, Italy, 2006.
[5] T. Schmickl and K. Crailsheim, ”Trophallaxis within a robotic swarm: bio-inspired communication among robots in a swarm,” Autonomous Robots, vol. 25, no. 1-2, pp. 171-188, 2008.
[6] S. Nouyan, A. Campo and M. Dorigo, ”Path formation in a robot swarm: self-organized strategies to find your way home,” Swarm Intelligence, vol. 2, no. 1, pp. 1-23, 2008.
[7] F. Glover, ”Tabu Search—Part I,” OSRA Journal on Computing, vol. 1, no. 3, pp. 190-206, 1989.
[8] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, ”Optimization by Simulated Annealing,” Science, vol. 220, no. 4598, pp. 671-680, 1983.
[9] J. McCall, ”Genetic algorithms for modelling and optimisation,” Journal of Computational and Applied Mathematics, vol. 184, no. 1, pp. 205-222, 2005.
[10] K. H. Hingrajiya, R. K. Gupta and G. S. Chandel, ”An Ant Colony Optimization Algorithm for Solving Travelling Salesman Problem,” International Journal of Scientific and Research Publications, vol. 2, no. 8, pp. 1-6, 2012.