Multi-Agent Collaboration for Search and Rescue Missions

Abdelrehim Elembaby
Mechatronics
German University of Cairo
Cairo, Egypt
abdelrehim.elembaby7@gmail.com

Hazem Mostafa Abdelaziz
Mechatronics
German University of Cairo
Cairo, Egypt
hazem.anwar@protonmail.com

Aser Sameh Waked
Mechatronics
German University of Cairo
Cairo, Egypt
asser.waked@student.guc.edu.eg

Moustafa Abdellatif Saad
Mechatronics
German University of Cairo
Cairo, Egypt
moustafa.abdellatif@student.guc.edu.eg

Sherif Karim Mounir
Mechatronics
German University of Cairo
Cairo, Egypt
sherif.gindy@student.guc.edu.eg

***Abstract*—Search and rescue (SAR) operations commonly employ multi-agent collaborative systems of unmanned vehicles to search through a disastrous environment as fast as possible, locating possible survivors to send aid in the form of rescue teams. In this paper, we study the effectiveness of different meta heuristic optimization techniques at optimizing the UAV(s) path. The map is discretized, and different priorities are assumed for each cell on the map, which reflect the likelihood of saving more lives if a specific cell is to be searched first.**

***Index Terms*—SAR, search, rescue, metaheuristic, optimization**

I. Introduction

In large disastrous environments, it is unsafe to allocate human search and rescue (SAR) teams to search for survivors, in addition to it being a slower process. Usually, hence, unmanned vehicles of ground or aerial kind are employed to search through the space. Unmanned Aerial Vehicles (UAV) are most commonly utilized in these kind of searches, especially when the map is very large. The identification of survivors utilized any kind of ground vehicle becomes impractical and, in many cases, unfeasible.

II. Literature Review

Amir Shabani, Behrouz Asgarian, Saeed Asil Gharebaghi, Miguel A. Salido, and Adriana Giret proposed a new optimization algorithm based on exploration carried out by humans in search and rescue and called it the search and rescue optimization technique (SAR). SAR is a human-based technique. SAR's performance was evaluated on fifty-five optimization functions, and its results were compared to twelve optimization algorithms, including GA, DE, CMA-ES, and PSO. The results indicated that the SAR algorithm is more competitive than other algorithms and can find solutions with fewer function evaluations compared to other algorithms. [1]

For the classical POMDP formulation, Hend AlTair, Tarek Taha, Jorge Dias, and Mahmoud Al-Qutayri presented a novel rewarding scheme. The proposed approach tries to strengthen objective preference. It assures that high-priority preferences receive high cumulative rewards, that ambiguities are resolved, and that it can be carried out before low-priority preferences. POMDP is an extension of the Markov Decision Process (MDP) to partial observability. The proposed method was evaluated in a Search and Rescue scenario involving a heterogeneous team of a robot and humans with complementary talents. Through a re-definition of the reward function, search and rescue operations incorporate search and rescue elements such as risk, energy, and time into decision-making. Multiple POMDP solvers were used to test its rewarding system in simulated settings. According to the simulated tests, the system can portray an impact on the person (first responder) by enforcing priority of objectives and iteratively optimizing policies to suit better the first responder and the search and rescue mission goals. [2]

Multi-agent systems are not yet suitable for implementation in search and rescue applications, although they progress in several key areas. Eventually, the gaps between our current capabilities and what is required for disaster relief will be bridged. With that being said, some of the most significant technical hurdles to MAS being used in search and rescue can be boiled down to scalability, which has been defined by Daniel S. Drew as "the implication on functionality as agent count increases. A multi-robot system's functionality can be described using three metrics: reliability, autonomy, and mobility. For example, perception and planning require significant autonomy, mobility, and reliability to be effective and as quick as possible at task allocation. Medium autonomy, minor mobility, and significant reliability are needed for robust communication. Significant autonomy, minor mobility, and significant reliability are required for effective one-to-many control, and finally, significant mobility, medium reliability, and medium autonomy for cost/function tradeoff. However, these are not retrospective assessments, and they are qualitative references to discuss the technical challenge areas in MAS and shine the light on new advances in functionality. [3]

Victor San Juan, Matilde Santos, and Jose Manuel Andujar developed a discrete path planner for search and rescue operations (SAR) utilizing unmanned aerial vehicles (UAV) equipped with onboard cameras to quickly find people at risk as possible in a large bounded disastrous environment. Data collected from different sources is used to characterize the surface to be explored and estimate the potential risk and likelihood of people being located in different regions. The path of exploration within the generated risk/occupancy discrete map is planned to utilize different strategies: attraction approach, genetic algorithm (GA), adaptive GA, and particle swarm optimization (PSO). Additionally, the effect of the number of UAVs searching was studied, with both distributed and free formation styles. PSO generated the most optimal paths, with an increasing number of UAVs reducing the time. The different formation styles of multiple UAVs did not show a clear pattern, with one being more efficient than the other. [4]

III. Methodology

A. Problem Formulation

Our problem will be constrained to studying the offline optimization of search and rescue (SAR) operations in large-scale environments, such as cities, towns, and natural environments, affected by an earthquake, fire, or a similarly large-scale disaster. The search and rescue of potential survivors is the most crucial part in these instances, even before trying to lessen the risk (i.e., extinguishing a fire). In such large-scale disastrous environments, to better utilize the resources (i.e., rescue teams) at our disposal, SAR teams, in most cases, are equipped with technology, which today includes Unmanned Aerial Vehicles (UAVs) to locate potential survivors before allocating rescue teams to them.

In this process of searching, it is desirable to utilize all the available data about the environment and the nature of the disaster. This data is used to estimate and define the search area and give probabilistic values indicating the possibility of each region being occupied by people, and in that case, the potential hazard for the life of these people, Pterrain and Pemergency, respectively. These values are based on fuzzy logic outlined in Jan, Santos, Andujar (2018). The search area is divided into cells in discrete path planning, and each cell is assigned probability values. For this discretization of the map to be valid, when the UAV is in the center of the cell, it is expected that it will be able to track the entire cell, a function in the altitude and the opening angle of the lens of the UAV.

Additionally, UAV dynamics (i.e., path following, obstacle avoidance) are neglected, with the UAV running at a constant speed. Since the size of the UAV could be appropriately assumed to be of small size compared to a single cell, any increase in path length of a UAV due to conflicting paths with another UAV, as both detour off the path to avoid each other, is neglected. Hence, paths are considered straight lines from one cell to another.

Given an estimated and discretized search area with differing priority values for each cell, the first objective of a discrete path planner is to minimize the time passed before reaching higher priority value cells. Neglecting the dynamics of the UAV, it is assumed to run at a constant speed. The second objective is to minimize the overall distance traveled by the UAVs. Since this is a SAR operation, with lives at risk, the overall distance traveled will be of much lower weight compared to the main objective. A tertiary objective is to reduce the number of UAVs employed overall during the SAR operation. An increase in the number of UAVs employed may at some point become redundant, which would set an optimum value below the maximum number of UAVs available for deployment.

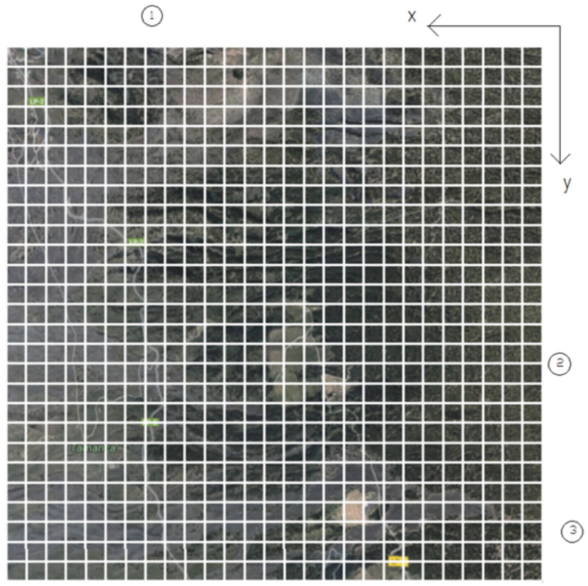

Fig. 1. A discretized geographical map, with surrounding bases.

The maximum number of UAVs available is an initial constraint. The limited battery life of UAVs presents another constraint on the maximum traveling distance before a UAV needs to be recharged at a base station for a fixed period. The quantity and locations of base stations are considered fixed during the SAR operation, and across the border of the search map, with UAVs deployed initially from different base stations. The number of charging ports per base station represents a constraint as well. A UAV must reach a base station with at least a single unoccupied charging port before its maximum traveling distance limit is exceeded.

Figure 1 shows an example of a discretized grid. An example of a discretized search map with three base stations around the borders. A symbolic, mathematical formulation of any solution would be as follows:

$$Sol = \begin{bmatrix} Path_1 \\ Path_2 \\ . \\ . \\ . \end{bmatrix} \tag{1}$$

Where Sol is sol $1xN_{UAV}$ matrix, N_{UAV} is the number of UAVs deployed, $Path_k$ is an array of (x,y) values, locating a base station or a search grid cell, outlining the path for UAV_k. From the problem statement, some solutions will be deemed unfeasible. First of all, any (x,y) of a grid cell in the solution must occur once and only once since each grid cell requires only one-time exploration by any UAV. Secondly, any path of a UAV must begin with the (x,y) of its initial position (at a base station) and end with the (x,y) of any base station. The problem, therefore, can be considered to be a permutation. From the constraints, firstly, N_{UAV} cannot exceed C_{UAV} That is the maximum number of UAVs available for deployment. That is:

$$N_{UAV} \leq C_{UAV} \tag{2}$$

To apply the two remaining constraints, UAV paths are divided into sub-paths, with the condition that each sub-path begins and ends with (i,j) of a base station and any other (i,j) is that of a grid cell. That is:

$$Path_k = \Big[Sub_{k,1} Sub_{k,2} ...\Big] \tag{3}$$

$$Sub_{k,m} = \Big[(x,y)_{k,m,1} ... (x,y)_{k,m,n}\Big] \tag{4}$$

The second constraint limits the landing locations at the end of each subpath to the base locations. The third constraints states that the value of the distance function, $Dist(Sub_{k,m})$, for all sub-paths across the solution, must not exceed C_{dist}, where C_{dist} is the maximum traveling distance before a UAV needs to be recharged at a base station. That is:

$$Dist(Sub_{k,m}) \leq C_{dist} \qquad for\ all\ k,\ m \tag{5}$$

The decision variables can be implied from the solution form and assumed constraints: N_{UAV}, denoting the number of UAVs utilized in the SAR operation N_{Path_k} for all k, denoting the number of sub-paths within each path, $N_{Sub_{k,m}}$ for all k, m, denoting the number of (x, y) within each sub-path, and the order of all cells across all subpaths.

B. Representation of Environment

An external environment is represented as a limited number of finite sets: A set of all (x,y) values of grid cells to be visited by the UAVs, a set of all (x,y) values of charging base locations, and a set of all priority values (between 0 and 1), each corresponding to a unique grid cell of (x,y). Figure 2 shows an example of an environment.

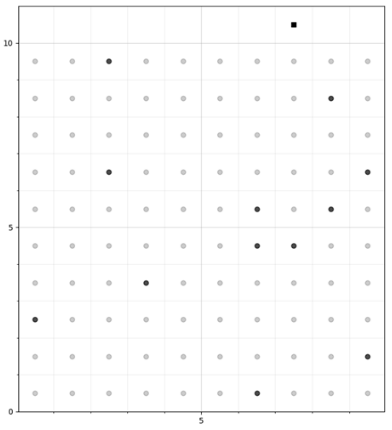

Fig. 2. An environment; squares, grey and black dots, denoting bases, low and high priority cells.

C. Solution Representation

Computationally, the solution is represented as a nested dictionary of keys and values/lists of values. The most important key is path, representing an ordered list of index values, each corresponding to a grid cell. A path must contain all grid cells, once and only once. The second most important key is uav_{count}, represents the number of UAVs utilized to generate this solution. $path_{uavs}$ contains non-overlapping subsets of path, that are approximately equal in distance from one another. Each one belongs to a specific UAV, and is utilized to generate divisions. Divisions is an array of equal length to $path_{uavs}$, each value containing an array of subpaths. These subpaths are generated based on the maximum distance a UAV can travel, while still being able to return to its base. Additionally, the returning base for each subpath is randomly selected, to optimize this aspect of the solution.

D. Code Structure

The language of implementation is Python. Python is an easy-to-use object-oriented, scripting language, oriented towards scientific applications. The code is divided into Python modules. Figure 3 shows an overview pseudocode of the main script. The Environment class is capable of generating random environments, based on initialized constants, for testing. In addition, it receives a number of constants, including speed for UAVs, maximum number of available UAVs and maximum charging time. The Problem class is initialized with an environment, and a number of constants, including switch_per, denoting the number of switching in the path when generating a new solution, per length of path. For example, for a path of length 10, randomized switched would occur 2 times for a switch_per value of 5. It also includes weights for the different cost components, and parameters that govern the randomization of the number of UAVs. A meta-heuristic object is then initialized with a problem object and algorithm-specific constants. After calling the 'run' function in the meta-heuristic object, a dictionary is returned, summing up all the relevant results. The results are ran through visualization functions to analyze and critique.

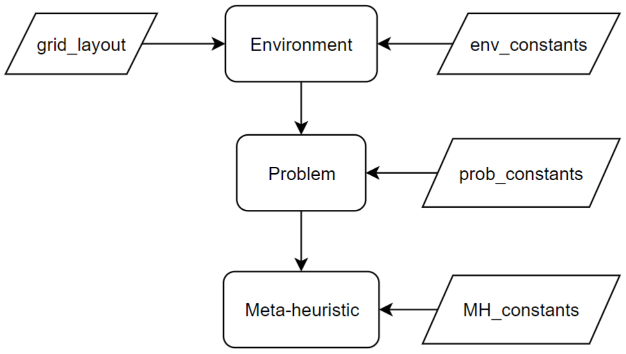

Fig. 3. Flow chart of the object-oriented hierarchal code structure.

E. Simulated Annealing (SA)

Simulated Annealing is one of the most implemented trajectory-based meta-heuristic optimization algorithms. It attempts to simulate the annealing process in metals, and how crystals form in the material. Three main functions in the *Problem* class are utilized by the SA algorithm. Initially, the *sol_init* function randomizes the path array, and the number of variables. Then, it calls the *get_sol_from_path* function. This function calls the *divide_path* function, that divides the path into path$_{uav}$, and further into sub-paths for each UAV. Secondly, in each iteration, *sol_gen* function generates a new solution based on a previous solution. The path is randomized by switching the positions of two random grid cells, multiple times, as well as randomizing the N$_{uav}$ value. The last function called is the *sol_cost*, that calculates and returns the cost of a function.

F. Genetic Algorithm (GA)

The Genetic Algorithm (GA) is one of the most popular population-based algorithms. It mimics biological evolutionary mechanisms and applies it to a population of search agents in the solution space of a problem. The GA algorithm utilizes the same functions as the SA algorithm, in addition to the *sol_cross* function. The *sol_cross* function utilizes David's Order Crossover to generate two children from two parents, by splitting and crossing the two paths of each parent. Additionally, the N$_{uav}$ is combined arithmetically. Parents are selected based on rank selection criteria. Survivors are fitness-based.

G. Ant Colony Optimization (ACO)

The Ant Colony Optimization (ACO) algorithm is inspired by the behavior of ants colonies as they optimize their path to a food source by disposing pheromones in more efficient paths. The ACO algorithm version implemented is the Simple Ant Colony Optimization (SACO), that excludes heuristic information from the transition probability. The algorithm utilizes the *sol_ant* function that accepts a Pheromone object. Based on the Pheromone object, it is able to consecutively construct a full path. This Pheromone object is a symmetrical 2D array. Each (i, j) represents the pheromone level of the path from i to j, or j to i. Additionally, another 1D array represents the pheromone level of the N$_{uav}$ decision variable. The deposition rule implemented is the ant cycle, where the total cost is taken into account, and not just the segment under consideration.

H. Bee Algorithm (BA)

The Bee Algorithm (BA) is inspired by the foraging behavior of bees. Scout bees lead the search by randomly flying to different regions. The scouts with the most profitable food sources recruit the highest number of foragers to search with them. Once a food source has been capitalized upon, the scout randomly explores another spot in the search space. BA utilizes the same functions as the SA algorithm. Scouts are randomly allocated in the search space. Recruits search inside a limited circle surrounding the scout. If a recruit finds a more profitable source, it adopts the scout title. If not, the search circle shrinks onto the local minimum, until it fully converges, allowing the scout to search yet another random location in the solution space. The shrinking local search is implemented by decreasing and increasing the switch_per parameter of the problem. The higher this parameter is, the smaller the search region is.

IV. Results

A. Case Studies

Figure 4 summarizes the variations in the cases we studied, for a total of twelve cases.

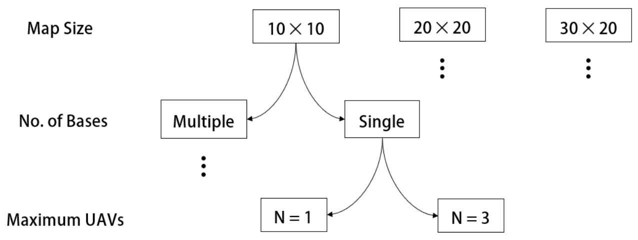

Fig. 4. A summary of our cases studies.

B. Repeatability

Figure 5 shows the standard deviation for each algorithm in each case, for a k of 3 (total of 3 runs per case and algorithm). ACO shows the lowest standard deviation, the highest repeatability, followed by SA.

Size	Base	N	SA	GA	ACO	BA
10x10	Single	1	26.05	38.79	26.24	44.1
		3	19.65	12.96	10.63	65.22
	Multiple	1	33.35	125.53	18.27	17.65
		3	85.64	37.76	29.12	46.93
20x20	Single	1	136.32	624.93	155.51	965.93
		3	343.0	947.61	80.83	719.42
	Multiple	1	19407.86	15753.7	3132.12	6615.87
		3	9784.11	7347.86	1427.42	3823.24
30x20	Single	1	370.47	595.39	59.33	1053.58
		3	681.37	445.65	45.51	82.21
	Multiple	1	872.97	956.82	288.55	601.77
		3	516.31	2026.18	109.94	1412.53

Fig. 5. The standard deviation of each algorithm in each case.

C. Runtime and Optimality

Figure 6 shows the runtime versus cost in the 10 by 10, single base, single UAV case. In this case, SA proved to be the fastest, most optimal. Infact, in all single UAV cases, six in total, SA proved to be fastest and most optimal, showing similar results. Figure 7 shows the runtime versus cost in the 10 by 10, single base, three UAVs case. In this case, ACO became the fastest, most optimal algorithm. This repeats in the multiple bases case. In a larger map, Figure 8 shows ACO to be on the verge of getting stuck, but ultimately reaching the most optimal solution after a long search time. SA, however, reaches a sub-optimal solution very quickly. Also, noticeable it is, how GA and BA perform better in a larger map, in reference to SA, in comparison to smaller maps. Figure 11 shows the initial and best solutions of ACO, 30 by 20, multiple bases, three UAVs case.

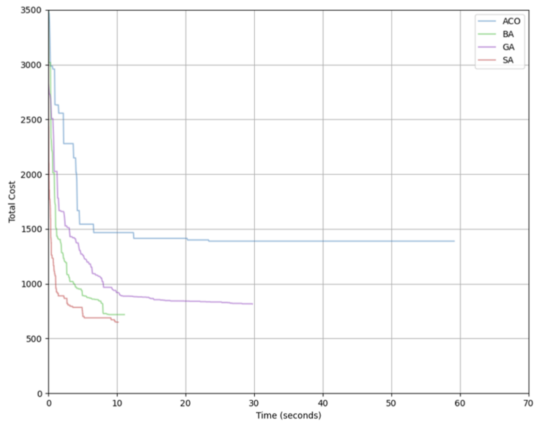

Fig. 6. Runtime versus cost in the 10 by 10, single base, single UAV case.

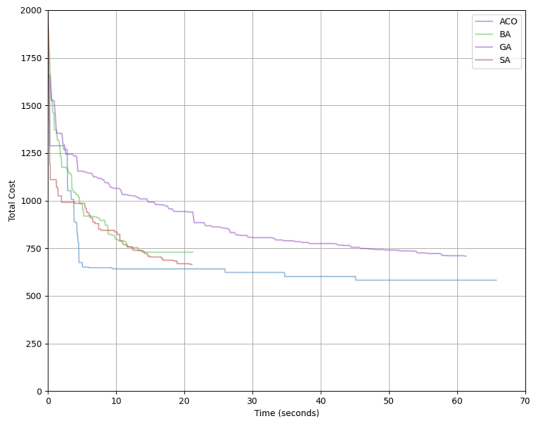

Fig. 7. Runtime versus cost in the 10 by 10, single base, three UAVs case.

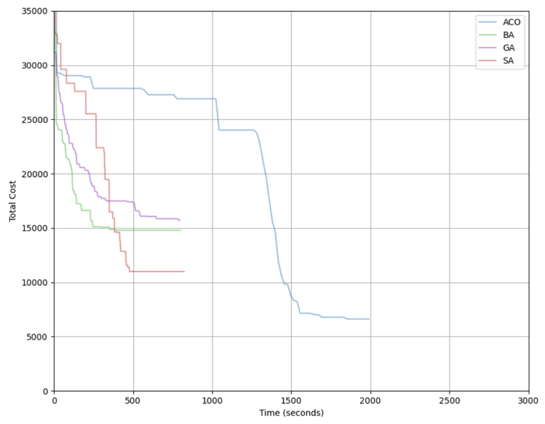

Fig. 8. Runtime versus cost in the 30 by 20, multiple bases, three UAVs case.

V. Discussion

The results show an array of different, peculiar patterns that require analysis. ACO proved to be very prone to getting stuck at low N values. This is inherent to the way we generate our solution. Because we perform our mutations on the path array, that is then further divided into the path_uavs array, the higher N is, the higher the exploration is, irrelevant of the algorithm-specific parameters. Hence, an algorithm that is prone to getting stuck, such as ACO, will be more likely to get stuck at lower values of N.

However, ACO did prove as well to be much more able to deal with path problems than other algorithms. When it did not get stuck, it converged to the most optimal solution. Figure 9 shows the distance component of the cost in ACO and SA. ACO's significantly decreases, while SA's does not. ACO is inspired by a path optimization biological mechanism in ants, and this explains its peak performance. However, due to its heavy computations, requiring more clock cycles to generate a single feasible solution, it takes a lot longer than most other algorithms.

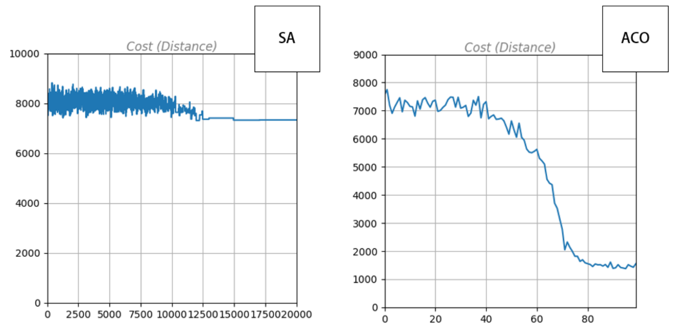

Fig. 9. The distance component of the cost in SA and ACO, in almost all cases.

Another interesting artifact is that GA approaches the performance of SA in larger map sizes. This can be explained through the illustration in Figure 10. As the map grows in size, there becomes more solutions that can satisfy sub-optimally the objectives. In other words, the search space is filled with many more local minima. A trajectory-based algorithm, such as SA, is more likely to get stuck at a local minimum than an algorithm such as GA. However, when there is a single global minimum, SA converges much faster to the optimal solution, compared to GA.

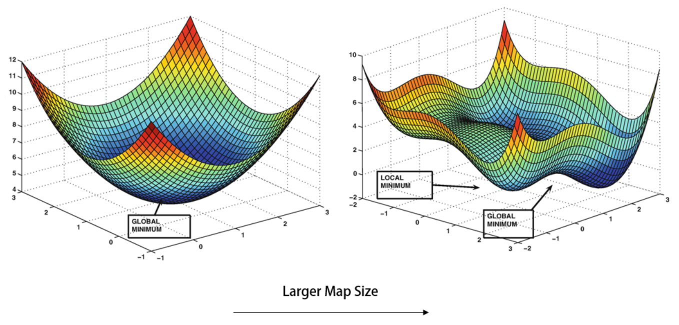

Fig. 10. The search space as the map grows in size.

The BA algorithm was very competitive with the GA algorithm. It showed near but enhanced performance in most cases. There is no obvious reason why this might be the case. Speculating, however, the BA algorithm does not having a convergence mechanism, could be the reason why it performs better. Utilizing rank selection survivor criteria in GA forces the population to mutate and cross from an increasingly converging population base. However, BA explores randomly, and converges locally. Upon local convergence, it again explores another random spot. Hence, it does not suffer from the risk getting stuck at local minima. As long as the algorithm is running, it exploring globally and exploiting locally.

VI. Conclusion

In summary, the performance of our four meta-heuristic optimization algorithms, optimizing the path of multiple UAVs in large SAR operations, was studied. SA proved to be very competent in runtime and solution optimality in small-sized maps. In larger maps, GA and BA proved to be competitive, in possibly more optimal than SA in very large maps. BA performed near but better than GA in most cases. ACO proved to be the most optimal of all, however highly prone to getting stuck, especially when planning the path for a single UAV.

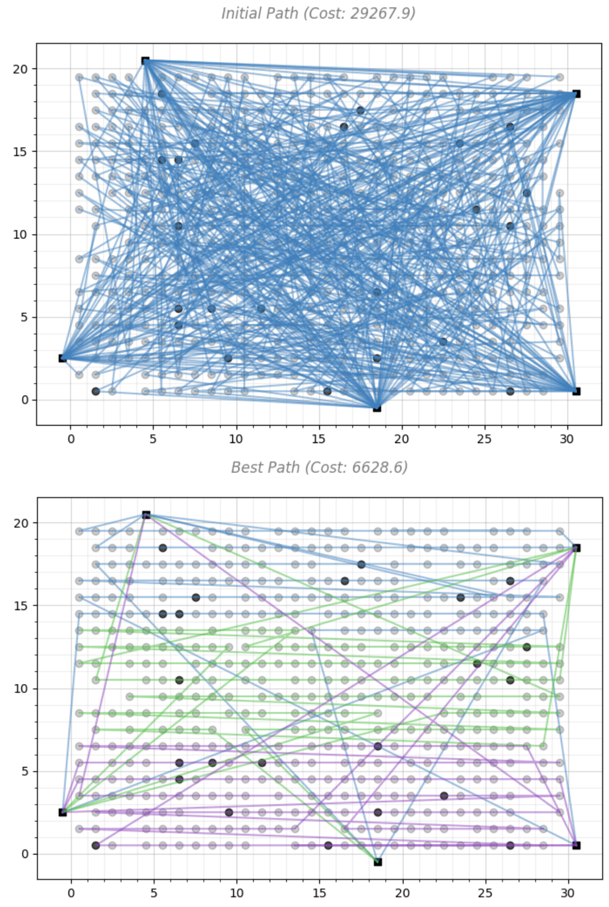

Fig. 11. Initial and best solutions of ACO, 30 by 20, multiple bases, three UAVs case.

VII. Recommendation

For future studies, it is recommended to study the behavior of these algorithms on more realistically sized maps, much larger than studied previously, to further study and verify the behavior of these algorithms.

References

[1] Shabani, Amir and Asgarian, Behrouz and Gharebaghi, Saeed Asil and Salido, Miguel A and Giret, Adriana, "A new optimization algorithm based on search and rescue operations", Mathematical Problems in Engineering, 2019

[2] AlTair, Hend and Taha, Tarek and Dias, Jorge and Al-Qutayri, Mahmoud, "Decision making for multi-objective multi-agent search and rescue missions", International Conference on Electrical and Computing Technologies and Applications (ICECTA), 2017

[3] Drew, Daniel S, "Multi-Agent Systems for Search and Rescue Applications", Current Robotics Reports, 2021

[4] San Juan, Victor and Santos, Matilde and Andujar, "Intelligent UAV map generation and discrete path planning for search and rescue operations", Complexity, 2018