Artificial Locust Swarm Routing Algorithm:

An Approach to Consider both Routing via FRA and Applying RAD

Vladyslav Alieksieiev
Department of Applied Mathematics
Lviv Polytechnic National University
Lviv, Ukraine
vladyslav.i.alieksieiev@lpnu.ua,
https://orcid.org/0000-0003-0712-0120

***Abstract*— A problem of routing in airspace looks both similar and different to the problem of routing on the ground. There are many peculiarities in air routing. However, due to the complexity of the problem and its similarity to 2D routing, there are many applications of a good known routing techniques from graph theory, like Dijkstra, A-Star etc. Nevertheless, these algorithms remain nearly useless or reveal poor efficiency at least in case of implementing specificity of a free routing airspace (FRA) and/or applying some kind of restrictions derived from route availability document (RAD). The results of research presented in current paper give the approach to solve routing problem in airspace with an ability to involve both understanding of routing via FRA and necessity of applying some known restrictions from RAD. The key idea of the developed algorithm is to combine routing in both regular and free airspaces and to apply some permanent requirements from route availability documents within a single route construction procedure. The solution was found in area of so named nature-inspired algorithms. The routing algorithm was considered similar to behavior of a locust swarm according to a few basic principles of movement. Applying some abstraction of a single locust behavior within a swarm allows considering the artificial locust swarm to be a good approach in search of route between departure and destination points for the routing problem in airspace. A couple of algorithm modifications and its areas of applicability were discussed also. Artificial locust swarm routing (ALSR) algorithm can be used as a platform solution for routes construction in airspace and is expected to allow further implementation of some particular routes optimization techniques, like genetic algorithms (GA) and/or other.**

Keywords- routing; path planning; FRA; RAD; artificial locust swarm; metaheuristic approach, nature inspired approach

I. INTRODUCTION

Current efforts in air routing had faced recent years many problems yielded from specificity of routing in airspace. If one compares a routing problem in airspace to a classic routing problem, there are many similarities. However, the differences are significant to make it rather difficult or even almost impossible finding the route in airspace. Sure, it is not a question to find some route itself, but the question is for what price it is done. If having a predefined network topology as a graph, then some of the good known algorithms, like Dijkstra [1], A* [2] or B* [3] algorithm, still could be used to build a route. However, computational efforts to build some valid routes may require using supercomputers or huge clusters to find a viable solution. Or it may take hours and days to get the only route, making the search process unsuitably time consuming.

First, remember, that an airspace routing has to consider three dimensions (unlike to on-ground routing problem solved typically in two dimensions). This means, if consider routing network, that each waypoint has projections to each flight level. And this fact can yield a multiplicative growth of possible edges (see Fig. 1): an edge connecting two waypoints means at least an existence of this edge in each flight level and, if applicable, edges to change flight level higher (climb) and/or lower (descend) from each flight level. This may result a combinatorial burst and critically raise the complexity for the route search if one would add all these edges to a routing network topology. Assessment for a maximum possible number of edges (the real number of edges) based on the number of edges on a plane is

$$N_{AllEdges} \leq N_{Edges} \times N_{FL} \times 3, \quad (1)$$

where N_{Edges} is the number of edges on a plane, N_{FL} is the number of flight levels, and number 3 stands for ability to fly higher, fly lower or to stay in a current flight level (FL).

Second, there is a set of rules and restrictions that are forcing, forbidding or allowing to fly between some pair of points. This may also look similar to an on-ground routing, like when there are road works and a road is closed. But in the sky it all may look imaginary. For example, on-ground routing is a somewhat simple when someone needs to cross the country border – this could be made only by crossing a border control points from one state to another and there are directly connected waypoints in both neighboring states. There are no visible country borders in airspace, so a pilot can violate country borders unintentionally. Moreover, there could be no defined direct connection between couple of waypoints in neighboring airspaces or this can be even marked as a single waypoint in both airspaces. So, the pilot may choose to fly between airspaces along the border and it violates the rules, but there is no road and fence in the sky to make someone to follow the rules. Another option is that the pilot can change flight level to avoid an obstacle (i.e., an airspace closed for military flights etc.). This means the same “road” could be in use, but it will require only to go above or below the closed levels. On a horizontal plane the route may look quite the same like a route violating the restriction and ignoring the obstacle presence.

This research is sponsored by RocketRoute Ltd. (London, UK).

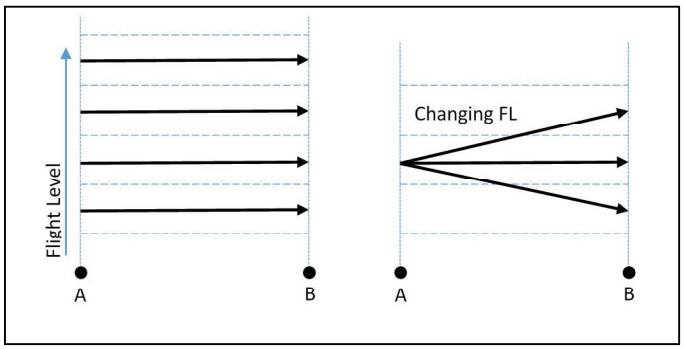

Figure 1. Multiplicative growth of actual number of edges

Third, there is a special kind of routing network part named FRA – free routing airspace. This means there are no "roads" there, but only a set of waypoints and/or some parts of routing network (i.e., SID/STAR routes close to airport) "hanging" alone in the sky. Actually, the case of routing through the FRA may become a "hell network" for the routing engine based on algorithms working with predefined topology. The reason is the existence of each-to-each connection between all waypoints (see a simplified view of FRA at Fig. 2 with only entry and exit points). And don't forget about the first concern of multiplicative growth of edges (1). The assessment for a number of maximum possible edges within FRA is

$$N_{FRAEdges} \le N_{Points} \times (N_{Points} - 1) \times N_{FL} \times 3, \qquad (2)$$

where N_{Points} is the number of FRA waypoints on a plane, N_{FL} is the number of flight levels, and number 3 stands for ability to fly higher, fly lower or to stay in a current flight level (FL).

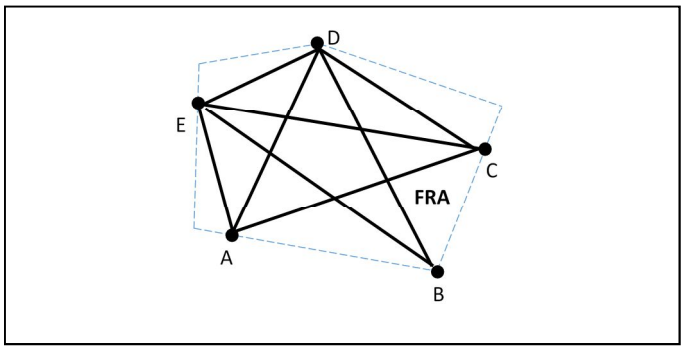

Figure 2. A simplified FRA view with only entry and exit points

Finally, the problem looks like more related to finding of a less cost path, rather than only the shortest path. Sure, there are many cases, when the shortest route has the less cost if to consider all the possible options. In general, when a route for an aircraft is needed, both distance, time and fuel consumption, route smoothness, flight safety, weather forecast and pilot preferences should be considered. This means, the problem requires a multifactor analysis approach to find some viable route for the flight. The complexity of the problem to find a route in airspace in compare to an on-ground routing problem is clear.

This paper is focused on discussion for development of a new approach to search for routes in airspace, including necessity to fly through FRA and ability to consider restrictions during the search process. The algorithm is focused on 3D routing as in airspace. Further development should involve understanding of weather conditions and allow improvements to a 4D routing.

II. Problem Statement

Initially, the problem is to find a route between departure and destination points to be able to perform a flight. Typically, this problem can be reduced to a 2D routing with predefined topology as a graph and an algorithm to find a path between two vertices on a graph is applicable. Some key aspects and factors for the analysis of the problem were discussed previously [4]. Current approach to solve routing problem is devoted to following specifics:

- Build a path between departure and destination points – a core problem.
- Consider existing ATS route network (as a topology presented by a directed graph with a set of vertices and a set of directed edges).
- Consider FRA (as an extension of topology presented by a subset of vertices with attributes of affiliation to FRA and role within this FRA).
- Consider basic restrictions for the flight (RAD, avoid areas, etc.).
- Get the result as a set of possible routes suitable for further optimization.

III. Approaches to Solve the Problem

A. Core problem of path finding

As mentioned above, the core of the problem to find a path can be solved using known approaches to build a path over the network presented as a graph. Many researchers apply this approach using Dijkstra or A* or similar methods [5–9] to find shortest path or exploiting shortest path search to build routes. Most researchers prefer using Dijkstra algorithm due to it is finite and it finds the optimal solution (unlike to possible sub-optimal solutions in A*). However, the A* often gives fast and robust solutions and its use looks preferable for directed search.

B. Approaches considering FRA

One approach to path finding in FRA [9] uses also a discretization to obtain a graph representation and relies on further application of Dijkstra and A* algorithms directly. Another approach to routing within FRA [10] relies on analysis of congestions in separate flight levels in FRA and shortest direct flights between entry and exit points. In general, this means direct flights within FRA can be considered as a shortest straight line between entry and exit points.

C. Nature-inspirde and other approaches

There is a number of researchers using a dynamic programing method to solve different problems related to path planning and path optimization [8]. The dynamic programming is mentioned often when some weather conditions are involved for routing [11, 12].

978-1-7281-1401-9/19/$31.00 ©2019 IEEE

An approach for route optimization using cellular automata was used to offer air network optimization in China [13]. This research is particularly interesting for discussing routing problems because of consideration of restricted (forbidden) areas in China and building routes avoiding these areas.

A weighted Euclidean shortest path was used within an approach of a framed-subspaces [14]. The idea of implementation of weighted region problem looks very simple: the whole space split into weighted regions with some weights (special cell decomposition technique was offered) and then a weighted Euclidean shortest path is used. This approach was offered to be used both in 2D and 3D environments. However, there are many other researches considering airspace splitting, like sectorization or segmentation or discretizing into cells.

One may also meet an approach called constrained shortest-path [15]. It was offered to be used in military aviation and path planning for strike aircraft, unmanned aerial vehicles (UAVs) or cruising missiles.

It is very important to mention here, that all previously described approaches consider the specificity of airspace routing problem and in particular regarding presence of obstacles and areas with different characteristics of applicability to fly through. This is the reason for search not only the shortest path, but an optimal path. As one can see, many solutions start with solving the shortest path problem with further implementation of some optimization technique. Some other solutions are the attempts to build an optimal solution combining path search with optimization efforts.

The most interesting subset of approaches to solve routing problem is a group of so named "nature-inspired" approaches. One of the latest approaches for 4D flight trajectories optimization was inspired by an artificial bee colony (ABC) [16]. The ABC algorithm allows both to search for path and build an optimized route between departure and destination airports, involving weather information and considering fuel consumption. It is worth mentioning, that the idea of swarm intelligence methods to search for path on a graph was also offered in an ant colony optimization (ACO) algorithm [17].

For example, among other recent nature inspired approaches there are BNMR (blind, naked mole-rat) algorithm [18] and grasshopper optimization algorithm (GOA) [19, 20]. These two approaches were used in numerical functions optimization and structural design problems, but not yet implemented to air routing problem. It is also interesting that the grasshopper optimization algorithm has a reference to a model of rolling swarms of locusts [21] offered for modeling a physical process.

IV. Artificial locust swarm routing (ALSR)

A. General analogy for locust swarm and aircraft flights

First of all, it is important to understand key features making it possible to apply an analogy from real-life to a technical system. Among these features of locust swarm are the following:

- *Model for 3D flights*: ability of locusts to fly allows applicability in 3D environments (like airspace).
- *Non-returning route*: effective devastation of fields and gardens allows to perform a movement always forward and not to return back, where all the food was eaten.
- *Stay within boundaries*: the swarm moves forward as a whole and each locust tends to keep together within the swarm, which means low probability of scattering.
- *Wind optimization*: the locust swarm is wind vulnerable and considering wind directions is similar to an aircraft flight.

Next, there would be a set of basic ideas in a model to make it easy to understand connection between nature processes and technical analogue. One can consider a network as a set of waypoints filled with some quantity of food. The locust will stand for an aircraft performing the flight across one possible route. The swarm stands for a set of all possible (or found) routes (not only to destination point).

B. Basic ideas

The artificial locust swarm will act as a whole and through each particular locust. These acts would be made with respect to the following ideas or rules:

- Swarm is a set of locusts and each locust aims to "eat" more "food".
- The food is distributed over the network, some quantity of "food" (≥ 0) at each waypoint.
- Locust moves over the network (by default, using known topology edges connecting waypoints) to eat more and a "hunger" makes locust to move due to exhaustion of previously visited waypoints.
- A track of each locust is stored and is added to a set of routes found, once the target was reached by the locust (and locust stops moving).
- Locust can "choose" whether to stay at a waypoint or to "jump" to a neighboring waypoint according to food quantity.
- If a waypoint is exhausted (zero food quantity) locusts can "fly" over the waypoint to its next neighboring waypoint.
- If there is no food in either current, any neighboring or next neighboring waypoint, then a locust stops and "dies".
- If a dead-end is found, then all edges leading to the waypoint are marked as "closed".

The most common schema of the algorithm is shown at Fig. 3.

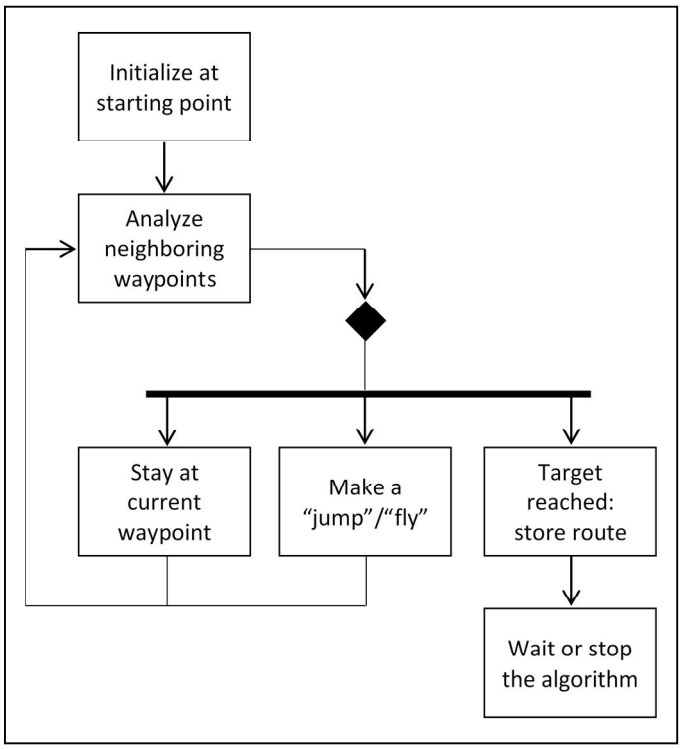

Figure 3. Common schema of ALSR algorithm

C. Additional assumptions

There is also a set of additional assumptions about locust swarm and each locust behavior (implementation of these ideas and rules can be optional):

- Locusts may avoid non-vacant waypoints (locust can "see" if the waypoint is vacant, when choosing a waypoint for the jump/fly).
- Locusts may avoid already used edges (edges can be marked as used and other locust should avoid these edges).
- Locust may be considered as died if there is no more food and no chance to reach target (the object for the locust can be deleted).
- Locust may have an intention to keep close to a swarm (not to be left far from the swarm).
- A long jump closer to the target is considered better than a short one.
- The wind could be considered to choose better edge to jump.
- Jumps between two waypoints without a connecting edge could be allowed (like in FRA).
- Food distribution could be a significant factor, but may relate actually to a maximum number of locusts allowed to pass through.

D. Locust movement

The movement of each locust is the core action allowing the route search. Performing a random movement may cause swarm scattering and overall algorithm failure. Due to importance of correct movement there is a set of strategies to be implemented:

- Next waypoint selection strategy – a set of rules to choose where to move.
- Time distribution strategy – a set of rules to make algorithm pick a locust to perform a move.
- Best route part strategy – a set of rules to use within a swarm the best route parts for the locusts which had routes crossings.

All three of these strategies relies, in some way, on a distance metric for the routing network. When selecting a next waypoint or picking a locust to move, an assessment of distance to the target can be performed and this requires some distance calculation. When comparing route parts, a calculation of distance covered should be made. So, anyway the question of distance calculation for any kind of locust movements is important. The most popular distance metrics are Euclidean distance, Manhattan distance and Chebyshev distance. Actually one can replace any of these distance metrics with a Minkowski distance:

$$D(\boldsymbol{x}, \boldsymbol{y}) = \left(\Sigma_{i=1..n} |x_i - y_i|^p \right)^{1/p}. \quad (3)$$

Here $\boldsymbol{x} \in R^n$, $\boldsymbol{y} \in R^n$ and n is the number of dimensions (n=2 or n=3 for 2D and 3D respectively), $p \geq 1$ is the distance order (p=1 and p=2 for Manhattan and Euclidean distance respectively, and $p \to \infty$ for Chebyshev distance). This kind of distance metric is not enough accurate for distance measuring on geoid. Nevertheless, this is quite enough to use as an initial approach to measure distance on a plane (or in Cartesian coordinate system).

Now let's discuss **next waypoint selection strategy**. General schema of next waypoint candidates are shown at Fig. 4. Here are some thoughts about neighboring waypoints to perform selection of a next waypoint on the route:

- Current waypoint is the last waypoint added to a track.
- Previous waypoints from the track should remain "blocked" to move to.
- Some edges connected to current waypoint give only a backward (reverse) direction way.
- Other edges connected to current waypoint give a forward direction (also a bi-directional edges) way.
- Waypoints from the other side of forward direction edges can be chosen as a candidate to become a next waypoint in the track.

978-1-7281-1401-9/19/$31.00 ©2019 IEEE

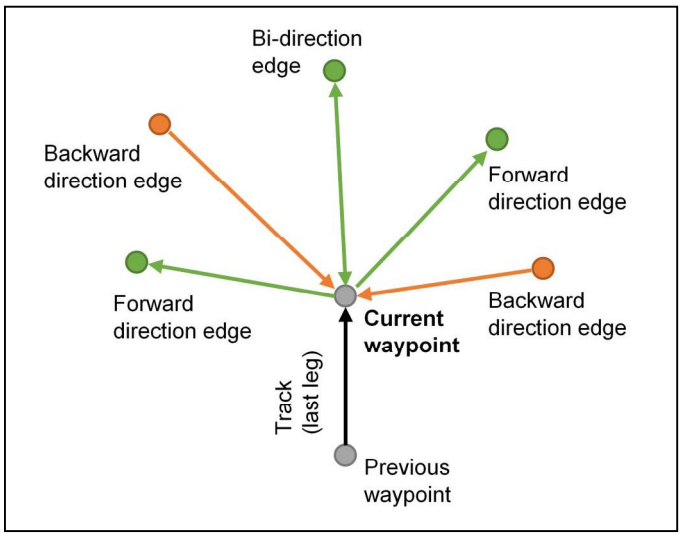

Figure 4. Next waypoint candidates from ATS route network

Besides the ATS route network with a predefined topology and edges connecting different waypoints, the movements through a FRA should be considered also if the current waypoint is an entry point to some FRA. Next waypoint candidates within a FRA are shown at Fig. 5.

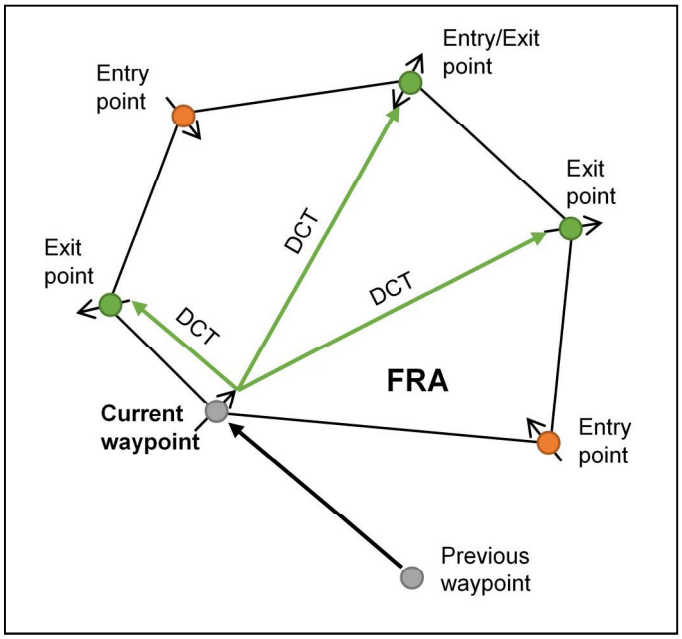

Figure 5. Next waypoint candidates from FRA

According to possible selection of next waypoint candidates both from ATS route network and from FRA previous thoughts should be extended with the following ones:

- Current waypoint is the last waypoint added to a track and located on the FRA border as an entry (or both entry/exit) point.
- Some waypoints on the FRA border are marked as entry points only.
- Other waypoints on the FRA border are marked as exit (or both entry/exit) points.
- A DCT between current waypoint and FRA exit point can be generated and added to a track.
- Further fitting of DCT should be made to meet all requirements to fly through the FRA.

Now, combining selection of next waypoint candidates from both ATS route network and FRA allows to define a set of waypoints for possible locust movement. Meanwhile, there can be a set of restrictions to use some edge or to build a DCT. These restrictions can be given by a route availability document (also known as RAD). Considering such restrictions to select next waypoint candidates is mandatory for the successful route search.

However, it is not yet enough to perform the best move. There are two sub-strategies involved to influence the best move selection: swarm size strategy and "food" initialization strategy. These two sub-strategies can also dynamically change both the whole state of a routing network system and the whole algorithm performance.

The ***swarm size*** sub-strategy can be one of the following:

- *Fixed-size swarm*. The swarm size can be a predefined value. The number of locusts in the swarm can be found according to number of waypoints in ATS routing network. The number of locusts in the swarm should be enough to find a route. Nevertheless, it is possible that all locusts in the swarm can get lost on a specified network. Moreover, there is no guarantee that the swarm size would be enough to find the optimal (the best) route. Each locust will use a special procedure to select the best next waypoint to move forward.
- *Growing swarm*. The swarm size can grow up according to the need to cover all possible routes from current waypoint to each next possible waypoint. This means, that a locust at a current waypoint will produce N–1 clones (descendants) to move to all N neighboring waypoints. This strategy makes the algorithm similar to Dijkstra and A-Star algorithms.

The ***food initialization*** sub-strategy can be one of the following:

- *Food "distribution"*. The distribution strategy can be used as way to set up food attribute for each waypoint in ATS routing network. This means that before the route search starts, all waypoints should be initialized. On one hand, this may become a time consuming process in case of a big amount of waypoints. On the other hand, this process can be considered as a part of preparation to run algorithm, but not the route search algorithm itself.
- *Food "discovery"*. The discovery strategy implements a "lazy initialization" pattern: the value of food attribute for the waypoint is generated when it is needed. Once the waypoint was seen by a locust, a food quantity for the waypoint is generated. All following requests of the waypoint will use this value as an initialized. The same

special procedure (as in distribution strategy) to define a food stock at each waypoint can be used.

The combination of two sub-strategies for the initial preparation to run the algorithm is presented in Table I.

TABLE I. SWARM SIZE AND FOOD INITITIALIZATION STRATEGIES

Swarm size / Food	*Fixed-sized swarm*	*Growing swarm*
Distributed	Both food quantity and swarm size should be defined before route search starts.	Food quantity should be defined before route search starts. The starting swarm size is equal to 1.
Discovered	Food quantity is unknown and to be discovered at first analysis of a waypoint. Swarm size should be defined before route search starts.	Food quantity is unknown and to be discovered at first analysis of a waypoint. The starting swarm size is equal to 1.

Both cases of food initializations may rely on a special procedure calculating the quantity of food at a particular waypoint. This food quantity may also represent the distance to a destination (targeted) waypoint: closer to a destination gives bigger food quantity. Such policy in food initialization looks similar to A* cost of path function. The quantity of food will be changed in time while ALSR algorithm works. Using (3) the food initialization function can be defined like:

$$Q_{food}(\boldsymbol{w}_{cur}) = \mathrm{D}(\boldsymbol{w}_{dep}, \boldsymbol{w}_{des}) / (q_0 + \mathrm{D}(\boldsymbol{w}_{cur}, \boldsymbol{w}_{des})) + q_1, \quad (4)$$

where $\boldsymbol{w}_{cur}$, $\boldsymbol{w}_{dep}$ and $\boldsymbol{w}_{des}$ are current, departure and destination waypoints respectively; $q_0 = const > 0$ and $q_1 = const > 0$ are fitting parameters (q_0 allows to avoid zero in denominator and q_1 allows to avoid very small values near departure point). Such definition of $Q_{food}(\boldsymbol{w}_{cur})$ is not the best one, but gives a very simple way to "put" more food closer to destination waypoint.

The quantity of food can be given by an integer value. Once a locust come to a waypoint the quantity of food to be decremented. If a locust stays at a waypoint for the next algorithm iteration, then a quantity to be decremented again. The rule can be formulated like "*a locust eats 1 portion of food in a waypoint at each algorithm iteration*".

The **time distribution strategy** allows to choose which locust is better to pick for the next step. By default, each locust has the same "attention" of the algorithm at each step – every locust makes a move. Implementing a time distribution strategy (see Fig. 6) allows to skip (at least for a while) some locusts.

Locusts far from targeted waypoint (destination point) may have less time for moves compared to those closer to a target – probability of the move (due to location) for the locust should be used (means less "attention" to wrong locations).

Locust that came to a dead-end or facing an obstacle can get lower probability to move (down to zero to be considered as died), the same as those located very far from the target.

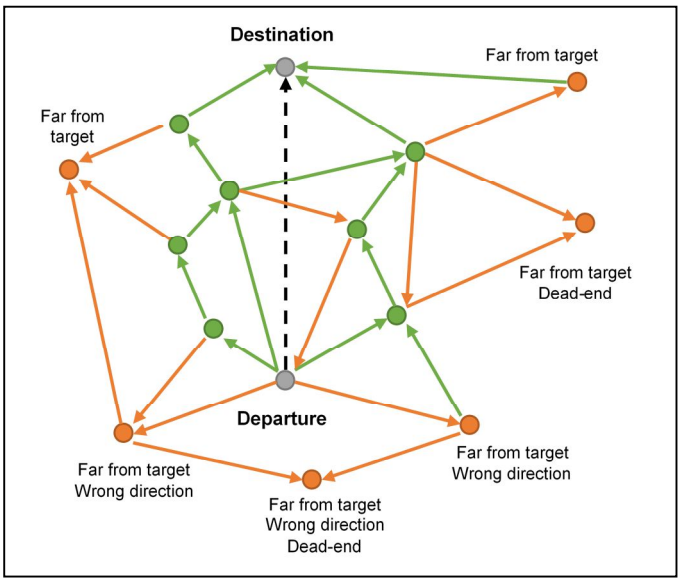

Figure 6. Time distribution strategy implementation

Giving an equal time to each locust may have following characteristics:

- *Always moving the whole swarm*. Each locust will move at each step. Each locust in a swarm is considered and granted a time for the move. This guarantees movement of every locust in a swarm.
- *Moving to a wrong direction*. Locusts can keep moving opposite to a destination point during a long time. This is similar to Dijkstra algorithm (searching wide).
- *Avoiding obstacles can slower the algorithm*. If there are many locusts faced the obstacle and unable to move forward, this may cause a waste of time to attempt a move. Edge lock can be the only way to avoid these obstacles if these edges leaded to a dead-end.

Using a special time distribution strategy according to probability of success may have following characteristics:

- *Moving only part of the swarm*. Those locusts going far from target or moving in "wrong" direction will be granted with less algorithm attention. This means after some time they can miss a step while others are moving.
- *More moves in a right direction*. Paying more attention to locusts that come closer to target may yield a quicker route search.
- *Probabilistic obstacles avoiding*. The "attention" of algorithm is focused on those locusts that can move. If a locust come to a dead-end or facing the obstacle, then the probability of granting a time for move can be lowered (down to zero) for these locusts. This means more attention to other locusts, which are possibly avoiding an obstacle or a dead-end.

The **best route parts strategy** can be used to make crossing routes better. If there are two routes with different paths between two waypoints (this means the routes has crossing), then one

978-1-7281-1401-9/19/$31.00 ©2019 IEEE

route may have a better route part between these waypoints (see Fig. 7).

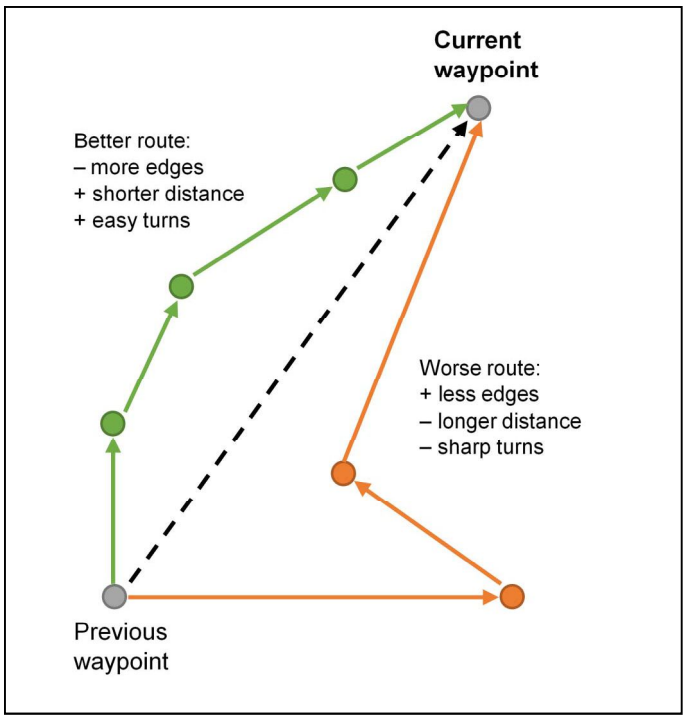

Figure 7. Best route parts strategy implementation

By default, there is no best parts replacement and the routes found remain the same as it were found. This can be named as a neutral to best route parts strategy with the following characteristics:

- *Keep the route as it was found*. Each locust finds its own route and this route remains the same until the destination is reached or locust can't move any more. This means the best parts of different routes cannot be combined to make a much better route.
- *Higher diversity*. All the routes in a resulting set will differ to each other. This can be a good and quick way for following route optimization (i.e., using GA, etc.).
- *No optimal route*. Among the routes found there could be no one optimal route.

When choosing to implement replacement with best route parts, then the algorithm can be named sensitive to best route parts strategy and will have following characteristics:

- *Detecting significant "cross-roads"*. If there are some locusts coming to the same waypoint by different ways, then the check for best way is performed. Both length and difficulty of the route should be considered to choose the best route to a "cross-road". Tracks for each locust moved through the "cross-road" should be replaced with the best route (replacement of route part is made).
- *Less diversity*. Resulted set of routes could become less diverse or even all routes (previously found as different) can be replaced with the same "best" route. An important option is to clearly identify what should be considered as a route part, because in general all routes will have crossing in departure and in destination points. So, for example, a route part can be defined to be not longer than a half or a quarter of the whole path.
- *Optimal route*. The best of the best route among all others can be found and it may be expected to be initially optimal.

E. Formal ALSR algorithm description

The pseudocode for the main algorithm flow can be presented like the following:

```
RequiredCountOfFinishers=5;
CountOfFinishers=0;
Swarm.Initialize(Start,Finish);
do {
   if(Swarm.PickNextLocust().Move()==Finish)
       CountOfFinishers++;
}while(CountOfFinishers<RequiredCountOfFinishers);
Swarm.DisplayFinishers();
```

Here the method "PickNextLocust()" is responsible for selecting a locust to perform a move.

```
Locust& Swarm::PickNextLocust()
{
   Locust *lctNext = NULL;
   if(Swarm::TimeDistributionStrategy) {
      lctNext = PriorityPick(Queue);
   }
   else {
      lctNext = Queue.PopFront();
      Queue.PushBack(lctNext);
   }
   return *lctNext;
}
```

The locust method "Move()" is responsible to perform a move of the locust, including analysis of neighboring waypoints and selection of a waypoint to move to.

```
void Locust::Move()
{
   Waypoint *wptNext = Position();
   Edge *edgNext = NULL;
   if(!FindNextPosition(wptNext,edgNext)) {
      CheckDeadEnd();
   }
   else if(wptNext != Position()) {
      if(!edgNext)
         edgNext = Edge::MakeDCT(Position(),wptNext);
      Track.Add(edgNext);
   }
   Eat();
   if(Swarm::BestRoutePartsStrategy) {
      /* following method was not implemented yet*/
      Swarm::ReplaceBestRouteParts(this);
   }
}
```

The locust method "FindNextPosition()" is responsible for setting values for next waypoint and edge from current position to this next waypoint. In case of using FRA the edge will be left empty, so a DCT is generated instead.

```
bool Locust::FindNextPosition(Waypoint*& w, Edge*& e)
{
   Waypoint *wptNext = NULL;
   Edge *edgNext = NULL;
   vector<Waypoint*> wptCandidate=
```

```
        Waypoint::FRAExitPoints(Position());
    vector<Edge*> edgCandidate=
        Edge::WaysOut(Position());
    RemoveVisitedWaypoints(wptCandidate,edgCandidate);
    ApplyRAD(wptCandidate,edgCandidate);
    BestPositionAssesment(
        wptNext,
        edgNext,
        wptCandidate,
        edgCandidate
        );
    w=wptNext;
    e=edgNext;
    return(wptNext?true:false);
}
```

The locust method "BestPositionAssesment()" allows to perform assessment of possible next waypoints among two lists: a lists of edges leading from current position and a list of exit points in FRA if a current waypoint is marked as an entry point of the FRA. This assessment can include both a quantity of food at neighboring waypoints, a swarm attraction force, a vector of wind, a distance from each neighboring waypoint to the swarm target waypoint and cost of flight by the edge or by the FRA. Previous to call of "BestPositionAssesment()" there are calls to "RemoveVisitedWaypoints()" and to "ApplyRAD()". These two methods should be used to avoid cycle in a flight (not to return twice to the same point) and to apply known restrictions for the flight (mark closed flight levels for the waypoints and edges, remove waypoints in a fully closed areas, mark mandatory usage of waypoints and edges).

V. CONCLUSION

The ALSR algorithm is still being developed and now it is offered as a concept to be implemented. A number of ideas described were not tested yet due to a wide range of parameters involved in the algorithm. Nevertheless, there were some tests made to check the ability of the algorithm to find routes on a randomly generated data set. These results for a fixed-size swarm were rather promising and had been presented in [4].

Current research was a test for growing swarm and a result is closely suitable for business use (see Fig. 8). This first route looks like: "EGNX DTY3N DTY UM605 FINMA M605 SILVA M183 CPT Y321 NUBRI Y321 PEPIS Q41 SAM M195 IBNID M195 MARUK N621 LELNA UN621 DOMOK UT260 UPALO UN862 REN A25 GODAN UN862 TERPO UT176 DESAB UN867 PEXOD UN867 NENEM UP75 OMILU UP75 NEA UN864 ORBIS ORBIS1C LEMD" – 782,75 NM. It looks appropriate enough to be used. But it is not as bad as it may look. The simulation settings were set up to consider better routes with less number of legs. This explains a number of turns and some longer legs within the route. Like it was mentioned above, a set of fast made different routes is suitable for further optimization. And a deeper analysis of a resulting set of routes is promising enough.

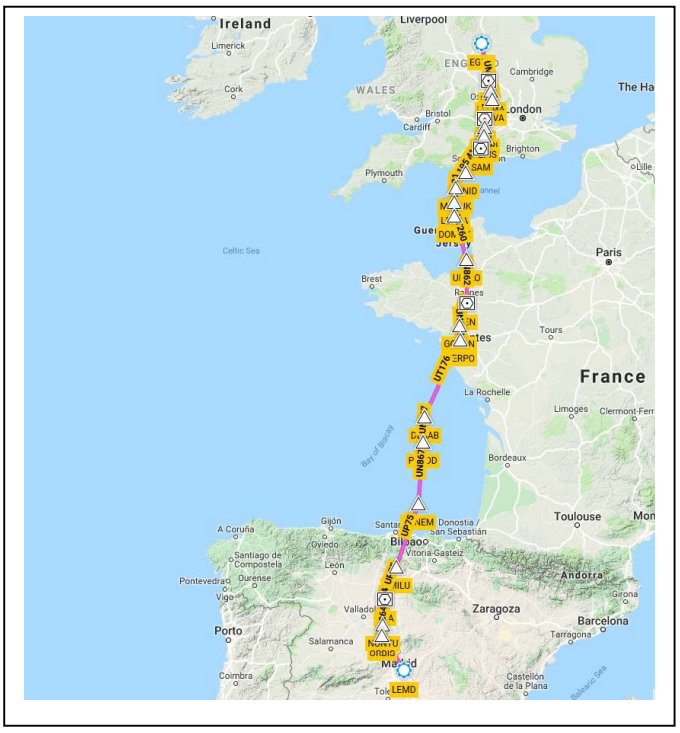

Figure 8. Example of a route for EGNX-LEMD by ALSR test

To compare to the best (shortest) route the RocketRoute routing engine was used. Among variety of other approaches, it allows to build a route according to Dijkstra algorithm, (see Fig. 9): "EGNX DCT DTY UM605 FINMA M605 SILVA Q41 PEPIS Q41 SAM N63 LELNA UN621 BASIK UN90 NOVAN UN864 ORBIS DCT LEMD" – 774,8 NM.

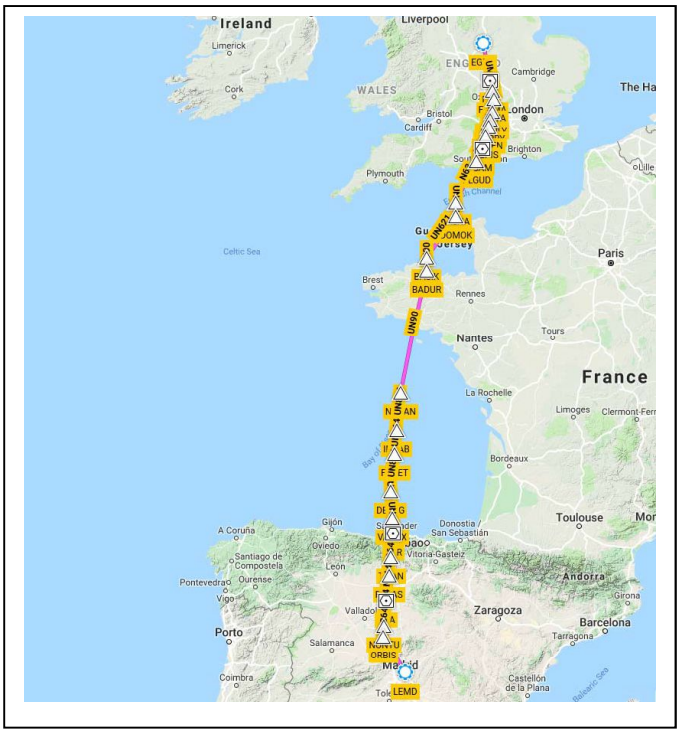

Figure 9. Example of a route for EGNX-LEMD by RocketRoute engine

978-1-7281-1401-9/19/$31.00 ©2019 IEEE

Both routes look similar and use many same waypoints. Nevertheless, the route from ALSR algorithm is about 8 NM longer. This is equal to 1% of difference and can be considered a very good result for the length of a sub-optimal route search. Moreover, the resulting set besides the discussed shortest one includes also some longer routes:

- "EGNX DTY3N DTY UM605 FINMA M605 SILVA M183 CPT Y321 NUBRI Y321 PEPIS Q41 SAM M195 IBNID M195 MARUK N621 LELNA UN621 DOMOK UT260 UPALO UN862 REN A25 GODAN UN862 TERPO UT176 DESAB UN867 PEXOD UN867 NENEM UP75 OMILU UP75 NEA UN864 ORBIS ORBIS1C LEMD" – 782,75 NM of length.
- "EGNX DTY3N DTY UM605 FINMA M605 SILVA M183 CPT Y321 NUBRI Y321 PEPIS Q41 SAM M195 IBNID M195 MARUK N621 LELNA UN621 DOMOK UT260 UPALO UN862 REN A25 GODAN UP87 TERPO UN862 UVUDO UN862 OSMOB UN857 VAVIX UN857 NETUK UN10 PPN UN857 VASUM UN857 BAN BAN3B LEMD" – 815,778 NM of length.
- "EGNX DTY3N DTY UM605 FINMA M605 SILVA M183 CPT Y321 NUBRI Y321 PEPIS Q41 SAM M195 IBNID M195 MARUK N621 LELNA UN621 DOMOK UT260 UPALO UN862 REN UN862 GODAN UN482 KORER UN26 NORMI UN867 TEPRA UN873 DESAB UN867 PEXOD UN867 NENEM UP75 OMILU UP75 UNOVI UN864 ORBIS ORBIS1C LEMD" – 809,75 NM of length.

To be more correct, let's also see the reverse direction flight from LEMD to EGNX. The best route result from ALSR is presented at Fig. 10 and it is compared to the best (shortest) route presented at Fig. 11.

The set of routes from ALSR includes following:

- "LEMD SIE1E SIE UN858 DGO UN867 BLV UN867 NENEM UN867 PEXOD UN867 DESAB UN867 TEPRA UN873 NORMI UN867 MOKOR UN867 TERPO UN867 RINSO UN867 EKRAS UN867 AKIKI N867 GARMI N867 VASUX UN867 AVANT UM184 HEMEL T420 BUZAD T420 OLNEY T420 WELIN PIGOT1J EGNX" – 779,965 NM of length.
- "LEMD SIE1K SIE UN865 BUGIX UN865 DELOG UN873 DESAB UN867 TEPRA UN873 NORMI UN873 MOKOR UP87 TERPO UN867 RINSO UM184 EKRAS UN867 AKIKI UN867 GARMI N867 VASUX N867 AVANT UM184 HEMEL T420 BUZAD T420 OLNEY T420 WELIN PIGOT1J EGNX" – 783,065 NM of length.
- "LEMD SIE1E SIE UN858 DGO UN867 BLV UN867 NENEM UN867 PEXOD UN867 DESAB UN873 TEPRA UN867 NORMI UN867 MOKOR UN867 TERPO UN867 RINSO UM184 EKRAS UT507 REVTU UP87 BOLRO UP83 KATHY DCT SAM ASTRA3A EGKK LAM5M LAM UN57 WELIN PIGOT1J EGNX" – 824,454 NM of length.

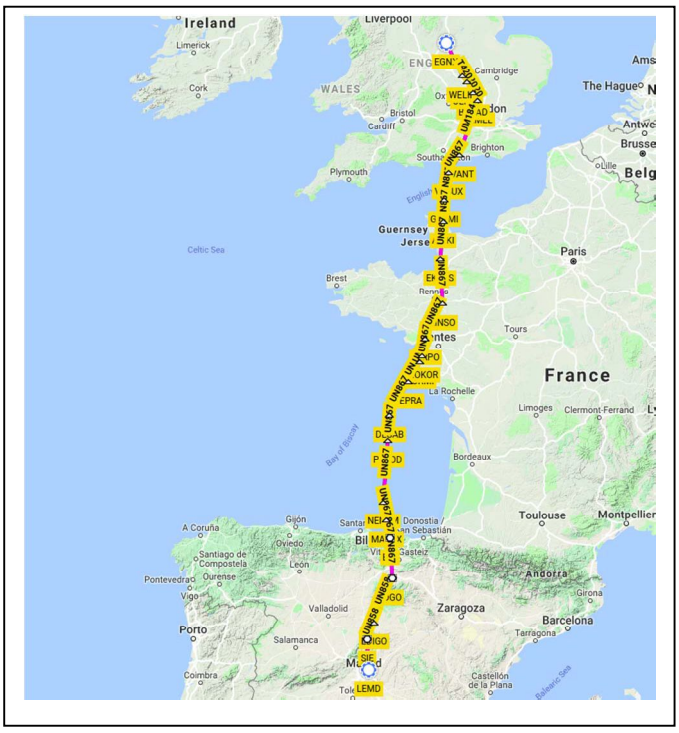

Figure 10. Example of a route for LEMD- EGNX by ALSR test

The compared shortest route (see Fig. 11) is "DCT RBO UN867 BLV UP87 BELEN DCT SUGOM DCT AGOTU DCT ARTIV DCT REVTU UP87 BOLRO UP83 KATHY DCT SAM Q41 PEPIS Y321 CPT UN859 HON DCT" – 759,6 NM.

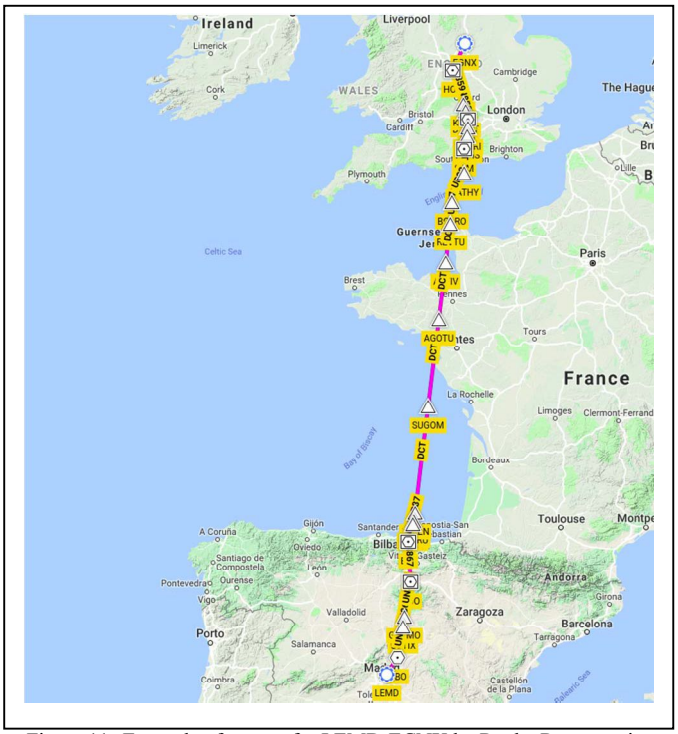

Figure 11. Example of a route for LEMD-EGNX by RocketRoute engine

The difference of the best route from ALSR is 20,365 NM, which means ALSR route is 2,7% longer. As one can see it has

some more differences on a route: many different waypoints and different airways were used in ALSR. Nevertheless, the result is still very satisfactory as for suboptimal routing algorithm, which the ALSR algorithm is. Most currently known to be used approaches in its basis rely on some algorithm [1–3], which requires knowing the full graph topology. And this is a separate serious problem of possible combinatorial burst considering full set of interconnections in FRA to find the path through this FRA. This problem typically makes it completely impossible or inefficient the routing in real life and does not match business needs. Unfortunately, there no other known or publicly available algorithms which are practically used to solve the problem of routing via FRA. Contrary, the ALSR algorithm approach has such features involved, that may allow efficient enough routing. The difference in length between the shortest route and the best route from ALSR is not significant.

This means the implementation of best route parts strategy allowed to achieve better or close to optimal first route generation. Also a good result for ALSR algorithm is its ability to generate a set of routes for further processing. This feature can become particularly interesting in situations when there would be a final validation failure for the "best" route (i.e., some unexpected restrictions yet unknown during route search), and not much time would be needed to offer alternative routes (these routes would be actually ready to be compared with a newly discovered restriction). Further researches would be made to fully implement conditions described by RADs and allow easy implementation of flights through the FRA within the single routing algorithm.

ACKNOWLEDGMENT

This research was fully funded by RocketRoute Limited (London, UK), owning the results of this research and all rights of intellectual property for the developed ALSR algorithm.

REFERENCES

[1] E. W. Dijkstra, "A note on two problems in connexion with graphs", Numerische Mathematik, Vol. 1, Iss. 1, pp. 269–271, December 1959.

[2] P. E. Hart, N. J. Nilsson, B. Raphael, "A formal basis for the heuristic determination of minimum cost paths", IEEE Transactions on Systems Science and Cybernetics, Vol. 4, Iss. 2, pp. 100–107, July 1968.

[3] H. Berliner, "The B* tree search algorithm. A best-first proof procedure", Artificial Intelligence, Vol. 12, Iss. 1, pp. 23–40, May 1979.

[4] V. Alieksieiev, "Air space routing and flights planning: a problem statement and discussion of approaches to solution", Mathematical Modeling, Vol. 2 Iss.4, pp. 139–142, December 2018.

[5] Z. Xie, Z. W. Zhong, "Aircraft path planning under adverse weather conditions", MATEC Web of Conferences 77, 15001 (2016), ICMMR 2016 – 4 p.

[6] A. V. Sadovsky. "Application of the shortest-path problem to routing terminal airspace air traffic", Journal of Aerospace Information Systems, Vol. 11, No. 3, pp. 118–130, March 2014.

[7] A. Murrieta-Mendoza, R. Botez, "Lateral Navigation Optimization Considering Winds and Temperatures for Fixed Altitude Cruise using the Dijsktra's Algorithm", Proceedings of the ASME International Mechanical Engineering Congress and Exposition, 2014, Vol. 1 – 9 p.

[8] C. Kiss-Toth, G. Takacs, "A Dynamic Programming Approach for 4D Flight Route Optimization", IEEE International Conference on Big Data, Oct. 27–30, 2014, pp. 24–28.

[9] C. K. Jensen, M. Chiarandini, K. S. Larsen, "Flight planning in free route airspaces", 17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017), 2017 – 14 p.

[10] M. Krzyanowski, "Conflict free and efficient flight routes planning in free route airspace", Prace Naukowe Politechniki Warszawskiej: Transport, 2013, pp. 277–285.

[11] L. Wirth, P. Oettershagen, J. Ambühl, R. Siegwart, "Meteorological Path Planning Using Dynamic Programming for a Solar-Powered UAV", IEEE Aerospace Conference, Mar. 07–14, 2015 – 11 p.

[12] H.K. Ng, S. Grabbe, A. Mukherjee, "Design and Evaluation of a Dynamic Programming Flight Routing Algorithm Using the Convective Weather Avoidance Model", AIAA Guidance, Navigation, and Control Conference (Chicago, Illinois), Aug. 10-13, 2009 – 13 p.

[13] Shijin Wang, Xi Cao, Haiyun Li, Qingyun Li, Xu Hang, Yanjun Wang, "Air route network optimization in fragmented airspace based on cellular automata", Chinese Journal of Aeronautics, No. 30 (3), 2017, pp. 1184–1195.

[14] R.J. Szczerba, D.Z. Chen, J.J. Uhran Jr. "Planning shortest paths among 2D and 3D weighted regions using framed-subspaces", The International Journal of Robotics Research, Vol. 17, No. 5, 1998, pp. 531–546.

[15] J.O. Royset, W.M. Carlyle, R.K. Wood, "Routing military aircraft with a constrained shortest-path algorithm", Military Operations Research, Vol. 14, No. 3, 2009, pp. 31–52.

[16] A. Murrieta-Mendoza, R.M. Botez, A. Bunel "Four-dimensional aircraft en route optimization algorithm using the artificial bee colony", Journal of Aerospace Information Systems, Vol. 15, Iss. 6, pp. 307–334, June 2018.

[17] M. Dorigo, "Optimization, Learning and Natural Algorithms", PhD thesis, Politecnico di Milano, Italy, 1992.

[18] M. Taherdangkoo, M.H. Shirzadi, M.H. Bagheri, "A novel meta-heuristic algorithm for numerical function optimization: Blind, naked mole-rats (BNMR) algorithm", Scientific Research and Essays, Vol. 7 (41), pp. 3566–3583, October 2012.

[19] S. Saremi, S. Mirjalil, A. Lewis, "Grasshopper optimisation algorithm: theory and application", Advances in Engineering Software, Vol. 105, pp. 30–47, March 2017.

[20] A.G. Neve, G.M. Kakandikar, O. Kulkarni "Application of grasshopper optimization algorithm for constrained and unconstrained test functions", International Journal of Swarm Intelligence and Evolutionary Computation, Vol. 6, Iss. 3, 2017 – 7 p.

[21] C. Topaz, A. Bernoff, S. Logan, W. Toolson, "A model for rolling swarms of locusts", The European Physical Journal Special Topics, Vol. 157, 2008, pp. 93–109.

 978-1-7281-1401-9/19/$31.00 ©2019 IEEE