A Pittsburgh Multi-Objective Classifier for User Preferred Trajectories and Flight Navigation

Viet V. Pham, Lam T. Bui, Sameer Alam, Chris Lokan, and Hussein A. Abbass

Abstract— **An efficient design of a Multi-Objective Learning Classifier System for multi-flight navigation is presented. A classifier is represented by a set of rules, which are used to simultaneously navigate all the flights in the airspace. Navigation of a flight is based on the relation of the flight with factors of the air traffic environment such as wind, storm as well as other flights. This system continually learns and refines the rules of classifiers by a multi-objective optimization algorithm - NSGAII - to discover the trade-off set of classifiers which navigate flights without any conflict, minimal distance of flying, minimal discomfort defined by storm level and the time duration of flights passing through storm areas, and minimizing total delay time of flights.**

We propose to detect conflicts between flights by grouping trajectory segments in 3-D (abscissa–x, ordinate–y, and time–t) boxes. The conflict detection is only implemented in a box, thus the number of conflict detection times approximates to the number of conflicts. Further, conflicts between flights are resolved using a hill climber by propagating delays in the take-off time of conflicting flights. The advantage of the proposed system is that the classifier outputs its rules in a symbolic representation, making the overall process transparent to the user and reusable. Moreover, the system successfully discovered rules in all runs to optimize its performance.

I. Introduction

Aviation has grown dramatically over the last decades [1]. Worldwide air traffic is expected to continue to grow at rates of 3-5% per year [2]. It should be noted that there are some areas in the world with a very high air traffic density. For example, on any given day, more than 87,000 flights are in the skies of the United States, in which about one-third are commercial carriers, like American, United or Southwest. On an average day, air traffic controllers handle 28,537 commercial flights (major and regional airlines), 27,178 general aviation flights (private planes), 24,548 air taxi flights (planes for hire), 5,260 military flights and 2,148 air cargo flights (Federal Express, UPS, etc.). At any given moment, roughly 5,000 planes are in the sky above the United States. In one year, controllers handle an average of 64 million takeoffs and landings. There are 14,305 air traffic controllers that work for the Federal Aviation Administration in 2006 [3]. In these high density areas, decision support systems in general and user preferred trajectory in particular become more important and necessary in air traffic control.

There are many methods of formulating the trajectory generation problem in the literature. Several methods work with a single aircraft only (e.g. in [4]), while others focus on multiple aircraft. However they usually deal with a small number of aircraft. For example, in [5], only experiments with one, three and four aircraft were reported, while [6] reported only the case with 3 aircraft. Besides that, the problem is usually formulated without taking weather conditions into account, such as wind and storms which affect the movement of aircraft, such as in [4], [7], [8], or in discrete environment [6].

Further, most methods consider pre-determined potential flight segments for finding an optimal or near-optimal path. These include a Hybrid A* algorithm [9], [10], Voroni polygons [11], [12], probabilistic maps and other graphical methods [13], [14]. Some researchers are experimenting with various analytical techniques to solve these path-planning problems, including singular perturbation [15], genetic algorithms [16], [17] and neighboring optimal control [18] as well as other analytical techniques. Genetic-based machine learning techniques are also used for multi-agent path planning in discrete environments, where the movement of one agent is from one cell to another cell in an environment of grid cells [6].

In this paper we extend Pittsburgh Learning Classifier Systems (LCSs), taking into account multi-objectivity for multi-flight navigation in continuous environments. We use NSGAII to evolve the population of classifiers. The algorithm will search for classifiers which can navigate flights with minimum distance, minimum discomfort, minimum delaying time and without any conflict. Weather conditions are considered while optimizing flight trajectories.

The paper is organized as follows. In Section II, we present the background of Multi-Objective Learning Classifier Systems. In Section III, the problem of trajectory optimization is formulated. In Section IV, the methodology is presented with description about the whole multi-objective learning classifier system and the details of algorithms to initialize flights, to initialize a classifier population, and to evaluate a classifier based on simulated trajectories using the classifier. The details of two genetic operators (cross over and mutation) are also presented. Section V is about experimental design. Results and analysis are presented in Section VI. Conclusions are drawn in Section VII.

II. Learning Classifier Systems

The origin of LCSs can be seen in Holland's work on complex adaptive systems and his early proposal on schemata [19]. This established the basis for the first practical

Viet V. Pham, Sameer Alam, Chris Lokan, and Hussein A. Abbass are with the School of SEIT, UNSW@ADFA, University of New South Wales, Australia, e-mails: v.pham@student.adfa.edu.au, s.alam@adfa.edu.au, c.lokan@adfa.edu.au, h.abbass@adfa.edu.au

Lam T. Bui is with the Department of Software Engineering, Faculty of IT, Le Quy Don University, Vietnam; email: lam.bui07@gmail.com

implementation of a classifier system which was called CS-1 (Cognitive System Level One) [20]. The system inspired a stream of research named as the Michigan approach. Coexisting with the early developments of Michigan LCSs, a parallel line of LCSs was also under investigation named as the Pittsburgh approach. It emerged with the LS-1 classifier system [21], which inspired a main classifier scheme called GABL [22].

In Pittsburgh LCSs each individual codifies a complete rule set, which eliminates the need for cooperation among individuals required in the Michigan approach. A such, the operation of the GA is simpler, the GA does not need to converge to a single solution. However, since Pittsburgh LCSs searches in the space of possible rule sets, the search space is larger and usually takes more computational resources than the Michigan approach. In addition, few controls can be exercised on the rule level. Two additional operators were designed in GABIL to overcome these issues. GIL [23] was another proposal that included a large set of operators acting at different levels, whose purpose was also to gain control over the type of rules evolved, but at the expense of an increased parameterisation.

While an increase in flexibility resulted from the use of variable sized individuals, parsimony pressures [24] and the use of multiobjective fitness [25], [26] were necessary to control the excessive growth of individuals.

III. Problem Formulation

In this paper, aircraft are considered to fly at constant altitude, resulting in planar motion. This is a common restriction in air traffic models, as air space is commonly structured in layers [27]. The problem is formulated as below

Given the following:

- a 2-D airspace, which is defined by a 2-D cell grid, in which each cell has wind and storm information, where wind information includes wind direction and wind speed and storm is represented by the storm level.
- the number of flights, each flight has origin, destination, normal speed, and a given user-preferred trajectory option–Utility.

Find: 3-D (abscissa–x, ordinate–y, and time–t) trajectories for flights, so that flights can

- avoid conflict with each other
- minimize delay time
- minimize distance travelled
- minimize discomfort based on their utility.

Details about the discomfort and the utility of a flight and objectives are described as follows.

The discomfort of a flight represents the amount of time the flight passes storm areas and how high storm levels at these areas are:

$$dc = \sum_{i}^{nsegment} \sum_{j}^{nsubsegment_i} time_{i,j} * stormlevel_{i,j} \tag{1}$$

where dc is the discomfort encountered in a flight, $nsegment$ is the number of segments of a flight trajectory, one segment is defined by two consecutive control points in the trajectory, $nsubsegment_i$ is the number of sub-segments of i^{th} segment which are created by the intersections of the segment with weather grid. The storm and wind in these sub-segments are constant. The ground speed of the flight in a sub-segment is updated based on the wind speed, wind direction and normal flight speed, then the time duration the flight uses to pass through the sub-segment is computed by dividing the distance of the sub-segment by the ground speed. $time_{i,j}$ is the time duration the flight passes through the j^{th} sub-segment of the i^{th} segment. $stormlevel_{i,j}$ is the storm level in the j^{th} sub-segment of the i^{th} segment.

The utility of a flight presents the user-preference with the flight. In this paper, the utility of a flight ranges from 0 to 1. If the utility is high, minimizing the distance travelled is preferred, otherwise minimizing the discomfort is preferred. For Cargo flights that have no passengers, an airline is likely to prefer a quicker route than a comfortable one, if the two objectives become in conflict.

This problem needs to minimize the total delay time of all the flights. The reason is that all the flights firstly need to take off at a given time (in this paper, we consider it to be time 0). In order to resolve conflicts between flights, we firstly try to resolve them by choosing the best direction for a flight at a segment. However, this cannot resolve all the conflicts, thus we need to resolve the rest of the conflicts through a hill-climber which propagates delays. With these goals in mind, we design two objectives as in Equations 2 and 3.

$$obj[0] = \sum_{i=1}^{nflight} u_i * (\frac{d_i}{sd_i} - 1) + (1 - u_i) * \frac{dc_i}{sdc_i} \tag{2}$$

$$obj[1] = \sum_{i=1}^{nflight} st_i + 10000 * nconflict \tag{3}$$

In Equation 2, $obj[0]$ is the first objective; $nflight$ is the number of flights; u_i is $Utility$ of i^{th} flight; d_i is the distance of actual trajectory of i^{th} flight; sd_i is the distance of straight line trajectory of i^{th} flight; dc_i is the discomfort of i^{th} flight; sdc_i is the discomfort of straight line trajectory with assumption of maximum storm level in every cell a flight passes, calculated as Equation 4, based on $MaxStormLevel$ (maximum storm level) and fs_i (normal speed of i^{th} flight).

$$MaxStormLevel * \frac{sd_i}{fs_i} \tag{4}$$

In Equation 3, $obj[1]$ is the second objective; st_i is the starting time of i^{th} flight; $nconflict$ is the number of conflicts between flights.

Minimizing $obj[0]$ gives preference to minimizing the distance traveled by a flight when the utility of the flight is high, and to minimizing the discomfort of a flight when the utility is low.

By minimizing $obj[1]$, we can minimize the total delay time and minimize the number of conflicts between flights. With a penalty on the number of conflicts, solutions involving flight trajectories without any conflict are preferred.

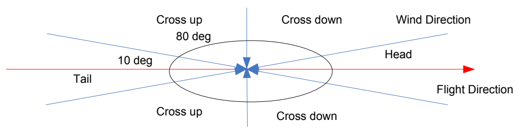


Fig. 1. Relative Wind Direction with Flight Direction

Parameter	RelativeWindDirection	StormLevel	Utility	Action
Value	rwd	sl	u	act

TABLE I
RULE PARAMETERS AND VALUES

IV. METHODOLOGY

A. Multi-Objective Learning Classifier System

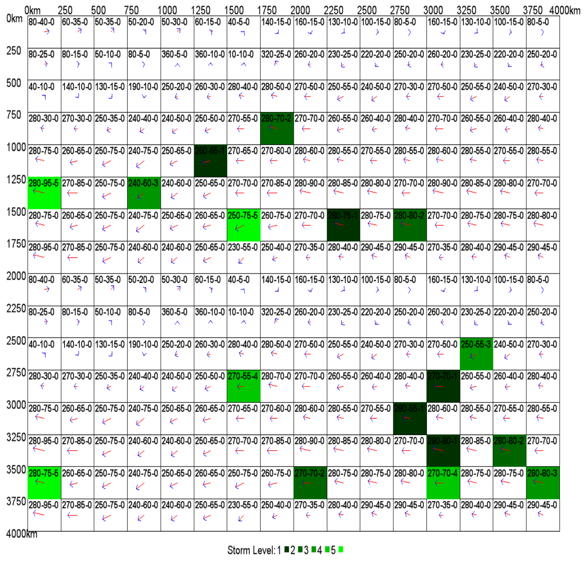
We adopt NSGAII [28] as the evolutionary multi-objective search technique. A population of classifiers is maintained over time in order to track the non-dominated set of classifiers for flight navigation. Each classifier is a set of normal rules and one default rule. A normal rule includes two parts: condition and action. The condition of a normal rule is defined by a relative wind direction with a flight direction and a storm level, where the relative wind direction may be either tail, cross up, cross down, or head. It is determined as in Figure 1, the storm level may be no-storm, low, medium-low, medium, medium-high, or high. The action of a rule is defined by an integer number from 0 to $ndirection-1$, where $ndirection$ is the number of possible directions a flight can choose to move. The index of a direction is determined clockwise starting from 0. The action of a rule represents the direction suggested by the rule when a flight satisfies the condition of the rule. A default rule does not contain any condition. It contains the action part only.

Table I lists the parameters (condition and action) of a normal rule and their values, where $RelativeWindDirection$ is the relative wind direction, $StormLevel$ is the storm level. The magnitude of $Utility$ of one rule is to present the user preference to the trajectory of a flight. If $Utility$ is high, minimizing distance travelled by the flight is preferred, otherwise minimizing discomfort is preferred. This rule is interpreted as "if (relative wind direction = rwd) and (storm level = sl) and (flight's utility $\geq$ u) then the direction with maximum acceptable level and closest to direction act determined clockwise is chosen". The acceptable level of a direction will be described in Section IV-E.

Algorithm 1 is to evolve the population of classifiers based on NSGAII. In this main algorithm, the algorithms to assign rank and crowding distance for the parent population, to select two classifiers from the parent population, to merge the parent population and child population to create a mixed population, and to sort the mixed population based on the crowding distance and including the most widely spread solutions to the parent population, are the same as in NSGAII. The other algorithms are presented in the following sections.

Algorithm 1 Main algorithm of Multi-Objective Classifier System

```
1: popsize is the number of classifiers in a population.
2: ngen is the number of generations in the evolution process.
3: Load weather grid including wind and storm cells.
4: Initialize flights.
5: Initialize parent population of popsize classifiers.
6: Evaluate the parent population.
7: Assign rank and crowding distance for the parent population.
8: for (i=2; i≤ngen; i++) do
9:     select every two classifiers in the parent population, cross-over the two to create
       two new classifiers and add them to a new population called the child population.
       This is repeated until the child population has popsize classifiers.
10:    Mutate the child population.
11:    Evaluate the child population.
12:    Merge the parent population and child population to create a mixed population.
13:    Sort the mixed population based on the crowding distance, and include the most
       widely spread solutions to parent population.
14: end for
```



0km 250 500 750 1000 1250 1500 1750 2000 2250 2500 2750 3000 3250 3500 3750 4000km
0km 250 500 750 1000 1250 1500 1750 2000 2250 2500 2750 3000 3250 3500 3750 4000km
Storm Level: 1 2 3 4 5

Fig. 2. An example of weather grid

B. Weather Grid Loading

The weather grid is stored in a flat file and loaded when the system starts. A storm level is only assigned for 10% of cells and for the cells with wind speed higher than average wind speed. The grid is an approximation of the size of the Australian airspace. The grid is four million square kilometers. Each cell is 250x250 square kilometers. The wind direction in one cell is $-\pi$ to π, speed 0 to 80 m/s, and storm level is between 0 and 5.

Figure 2 is an example of weather grid, where the vector in a grid cell represents wind direction and wind speed. Wind direction is determined based on North axis (zero axis) and clockwise. Wind speed is represented by the length of the vector. Wind vector with maximum speed is the longest one. The length of other vectors is proportional to the length of the longest vector according to the rate of the two speeds. The 3 numbers in the top-left corner of a cell are the wind direction, wind speed, and storm level at the cell.

C. Flight Initialization

Flights are initialized randomly with the following conditions:

- The distance between the origin and destination of one flight needs to be in a given range from $flightmindistance$ to $flightmaxdistance$;
- Flight normal speed is from $minACSpeed$ to $maxACSpeed$;
- The distance between the origins of two flights also needs to be greater than a given distance $minorigindistance$;
- Several values of Utility are used and each value is allocated with a certain amount of flights. The specific allocation is presented in Section V.

The straight line distance of a flight is calculated and stored in sd property of the flight as the distance between its origin and destination. The discomfort of a flight with assumptions of straight line trajectory, normal flight speed movement and maximum storm level at every cell is calculated and stored in sdc property of the flight. The boundary of one flight is determined by the rectangle created by the flight's origin and destination with expansion of a given margin.

D. Classifier-Population Initialization

A classifier population is initialized one classifier at a time, where a classifier is initialized by initializing its set of normal rules and its default rule.

A normal rule is initialized as follows:

- $Wind$ of the rule is initialized randomly from 0 to 3, corresponding to 4 relative directions between wind direction and flight direction: $tail$, $crossup$, $crossdown$, and $head$;
- $Storm$ of the rule is initialized from 0 to 5, corresponding to 6 storm levels: $no-storm$, low, $medium-low$, $medium$, $medium-high$, and $high$;
- $Utility$ of the rule is initialized from 0 to 1, which presents user preference to the trajectory of a flight;
- $Action$ of the rule receives an integer value from 0 to $ndirection - 1$, which corresponds to $ndirection$ possible directions the flight can choose.

The default rule of a classifier contains an action part alone. The $Action$ of a default rule is initialized in the same way as the action of a normal rule. Each rule also includes an index ID and its father index $FatherID$. The father of a rule is the rule it is generated from by the mutation operator, as presented in a subsequent section. When a rule is initialized, its $FatherID$ is 0. When a new rule is created by applying the mutation operator on a rule (father rule), its $FatherID$ is the ID of the father rule. These indices are used to track the changing progress of a rule.

E. Evaluation

A population is evaluated one by one. The evaluation is repeated until there are no conflicts between flights, or the number of conflicts does not decrease after a given number of steps $nSimStep$. In each step of the loop, flight trajectories are simulated, conflicts between flights are detected, and the objective values of the classifier are calculated. If there are conflicts between flights and the current step is not the final one, conflicts between flights are resolved by delaying the starting time of one of the conflicting flights.

The simulation is repeated until all flights complete their journey. In each simulation step, if the distance between the current point of a flight and its destination is more than R (meter), the flight will move to one of the $ndirection$ next possible points by $ndirection$ possible directions. Otherwise, it will move straight to the destination and the trip is completed. We maintain a vector storing the index of flights which did not finish. If a flight finished, the index of the flight will be removed from the vector. By maintaining this vector trajectory, the simulation only needs to go through flights which have flight index in the vector. This helps to improve the speed.

In order to determine the next point for the flight to move, firstly the rule to navigate the flight is chosen. If the flight at the current state satisfies a normal rule in the classifier, this rule is chosen to navigate the flight, otherwise the default rule is used. Then all possible next points are determined based on the current point of the flight and the next $ndirecion$ possible directions. The acceptable level of one point is also calculated. The acceptable level of one point is first assigned to 0. Then it is updated based on 3 conditions. First is whether the angle between the previous flight direction (between the previous point and the current point) and the next flight direction (between the current point and the next point) is less than PI/2 or not. Second is whether the next point is in the flight's boundary or not. Third is whether the new segment (created by the current point and the next point) causes conflict or not, where the segment only needs to detect with segments in boxes it belongs to. If the first condition is satisfied the acceptable level is 1. If the first and second conditions are satisfied the level is 2. If all 3 conditions are satisfied the level is 3. If there is only one point with highest acceptable level, this point is the next point the flight choose to move to, otherwise the point with highest acceptable level and closest to the direction suggested by the rule the flight satisfies is the next point.

The distance and discomfort of the flight are updated by those created by the new segment; in which the distance of the new segment is calculated as Euclidean distance. The discomfort of the new segment is calculated by summing the discomfort of every subsegment created by intersections between the segment and weather grid, where wind and storm are constant in a subsegment.

In order to detect conflicts between flights, the 3-D (x, y, t) space is divided to the 3-D grid of 3-D boxes. When a new segment is determined for a flight, it is added into the segment vector. Its index is also added to the segment index vector of 3-D boxes that the 3-D box enveloper of the segment intersects with. If the index of a box in these boxes is not in the box index vector, it is also inserted in the vector in an ascending order. The earliest arrival time of all the flights is also updated, based on the arrival time at the next point of the new segment.

Conflict detection is implemented in every step of the simulation for segments in boxes whose starting time is earlier than the earliest arrival time of all the flights. By maintaining the box index vector, the algorithm only needs to go through boxes which contain at least one segment index. The box indices are also sorted, thus the algorithm only needs to start from the first index and stop when it meets the index of a box whose starting time is later than the earliest arrival time. In order to maintain the box index vector, when a box is first added a segment index, binary searching is implemented to find the proper position to insert the index of the box in the box index vector, so that the indices of boxes are sorted in the ascending order. When conflict detection for a box finishes, the index of the box in the box index vector is removed; the segment index vector of the box is also released. This helps to reduce frequently the amount of system memory required. When all the flights finish, conflict detection is applied for the rest of boxes. Conflict detection in a box is implemented between every two segments in the box. Because two segments may appear in several boxes, thus two segments in a box are only applied conflict detection if the box's index is the smallest in the list of indices of boxes containing the two segments. This index is determined by the maximum x, y, t indices of the minimum ones from the 2 terminal points of each segment.

In order to detect conflict between two segments of the two flights, the intersection point is firstly determined between the two spatially. Then the arrival time at the intersection point of the two flights is calculated. If the difference between the two arrival times is less than 5 (minute) the two flights conflict.

The arrival time of a flight at a point from a point is determined by Equation 5:

$$AT_2 = AT_1 + \sum_{i=1}^{m} TD_i \qquad (5)$$

where AT_1 is the arrival time of the flight at the first point; AT_2 is the arrival time of the flight at the second point; m is the number of subsegments that are created by the intersections between the segment between the first and second points with weather grid. TD_i is the time duration the flight passes i^{th} subsegments.

TD_i is calculated by Equation 6:

$$TD_i = D_i / GS_i \qquad (6)$$

where D_i is the distance of i^{th} subsegment, GS_i is the ground speed of the flight at i^{th} subsegment.

The ground speed gs of a flight at a subsegment is constant. It is determined, based on the flight normal speed fs and relative wind direction rwd and absolute wind speed ws at the subsegment (they are also constant). If rwd at the position is tail wind, the gs is fs + ws. If rwd at the position is cross up, the gs is fs + ws/2. If rwd at the position is cross down, the gs is fs - ws/2. If rwd at the position is head wind, the gs is fs - ws.

In order to track conflicting flights with a flight, each flight has a conflicting flight vector that stores the indices of flights with which the flight is in conflict. When a conflict is detected between the two segments of two flights, the index of a flight is added into the conflicting flight vector of the other flight if the index is not available in the conflicting flight vector. For each flight f, the algorithm goes through its conflicting flights one by one until it finds a conflicting flight which has not delayed and finished later than f, or all the conflicting flights are inspected. If f and the conflicting flight have not delayed yet, the algorithm will delay the starting time of the flight which finished earlier than the other by 5 (minute). If f is delayed, the algorithm will stop going to the next conflicting flight of f.

F. Genetic Operators

The crossover operators crosses two classifiers $parent1$ and $parent2$ to create two new classifiers $child1$ and $child2$. Firstly, it generates a random number $rand$. If $rand$ is less than the cross over probability $pcross\_classifier$, the algorithm will use the cross over point $site$ generated randomly from 0 to $nrule - 1$ to cross $parent1$ and $parent2$. The default rule $DefaultRule$ of $parent1$ and $parent2$ is also crossed. If $rand$ is higher than $pcross\_classifier$ $child1$ and $child2$ will be the clone of $parent1$ and $parent2$, respectively.

The mutation operator goes through every normal rule. For each one it generates a random number $rand$. If $rand$ is less than the mutation probability, the rule will be mutated. In order to mutate a rule, an integer number mp from 0 to 3 is generated randomly, which indicates the mutated part of the rule. If mp is 0, the relative wind direction of the rule is mutated. If mp is 1, the storm level of the rule is mutated. If mp is 2, the utility of the rule is mutated. If mp is 3, the action of the rule is mutated. The default rule of the classifier is also mutated.

V. Experiment Design

Table II presents flight utility allocation.

Utility	0	0.1	0.3	0.5	0.7	0.9	1
Percentage	5%	15%	15%	15%	15%	25%	10%

TABLE II
Flight Utility Allocation

The parameters we used are: number of flights is 100, the minimum and maximum speed of a flight is 150 m/s and 250m/s respectively, the minimum and maximum distance of a trip is 100km to 4000km respectively, the minimum distance between the origins of two flights is 50km, number of generations is 100, population size is 100, number of rules per classifier is 50, the crossover probability is 0.9, the mutation rate is the reciprocal of the number of rules + 1, and the distance of a flight's moving step is 25km.

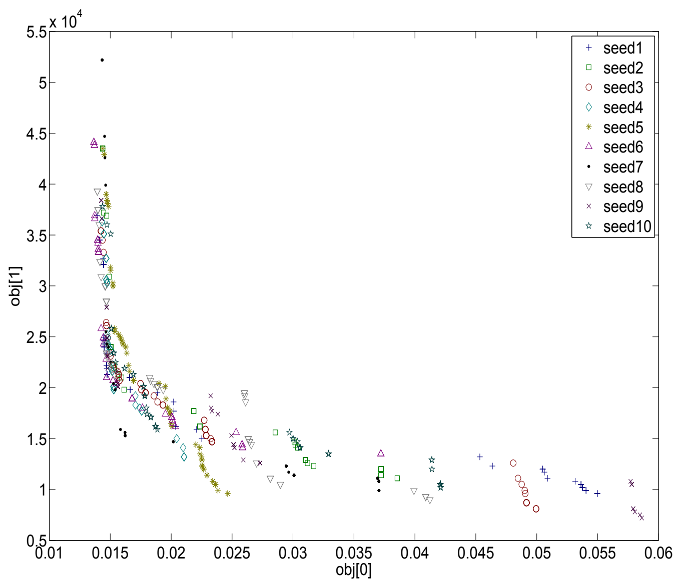


Fig. 3. Final Non-dominated Front of 10 Seeds

The system is designed to run in parallel for the evaluation process. In which one individual is assigned to a separate CPU for evaluation when the CPU is free. The experiment runs on 5 CPUs. The CPU type is Intel Nehalem 2.93GHz and the memory allocated for each CPU is 73.75 MB.

VI. Results & Analysis

A. Non-dominated front

We did experiments with 10 seeds (1/11, 2/11,...10/11), which are called seed1, seed2,... and seed10. Figure 3 shows the non-dominated front of the 10 seeds in the final generation.

We found that the range of the two objectives improves significantly from the first generation to the last. For example, in the experiment with seed2, the range of $obj[0]$ is from about 0.0171 to 0.0511 in the first generation, while it is from about 0.0144 to 0.0386 in the final generation. The range of $obj[1]$ is from 24,900 to 79,500 in the first generation, while it is from 11,100 to 43,500 in the final generation. Both the lower and upper bounds of $obj[0]$ and $obj[1]$ reduce from the first generation to the last generation.

B. Trajectory and Time Span Track

Table III presents the number of conflicts, the value of objectives each conflict resolution step of the two best classifiers with smallest $obj[1]$ in the final generation of seed2. The objective values in the final conflict resolution step in the table show that while one solution is the best in one objective, it is the worst in the other objective. Aircraft trajectories created by both the classifiers and the other classifiers of the non-dominated set are free from conflicts.

Figure 4 presents simulated trajectories and time spans of flights in the final conflict resolution step, by delaying starting-time of conflicting flights for a non-dominated classifier in the final generation of seed2. In this figure, the following information is presented:

- Two objective values ($obj[0]$, $obj[1]$) in top-left of the figure with red color
- Trajectories of all the flights, each trajectory has starting time and ending time with blue color
- An intersection in 3-D box between 2 trajectories (but not conflict) is presented by a green circle
- A conflict is presented by a red circle
- Arrival time of first flight at the conflict point in green (second)
- Arrival time of second flight at the conflict point in black (second)
- Four texts below the time spans are the number of delay flights, total delay time, the minimum delay time, and the maximum delay time.

Note that the time difference of the two flights at their conflict point is less than 300 second. There is not any red circle in the figure because all conflicts are resolved by delay propagation in the take-off time of conflicting flights. Aircraft trajectories in Figure 4 are long, because objective $obj[1]$ is to minimize the number of conflicts and delay time. Flights choose longer trajectories to avoid conflicts with each other.

C. Generation-rule track

Table IV shows the generation-rule track of a non-dominated classifier in the final generation of seed2. The track of a rule in one row presents the rule from the first generation to the last generation it changes. Each rule has usage (the number of times the rule is used to navigate flights), id (rule index), father id (index of the father rule), and content. The last rule is in the rule set of one of the two classifiers. The content of a rule may be the same as that of the father rule, because although the rule is subject to the mutation operator it does not change. From Table IV we can see that the action suggested by the default rule of the classifier whose $obj[0]$ is the smallest is 3. The direction index of 3 is the straight direction from the current point of a flight to its destination, because in our experiment we choose the number of possible directions for a flight to choose to move is 7 and the direction index is determined by clockwise from 0. The reason for the straight direction suggested by the default rule is that objective $obj[0]$ is to minimize the distance travelled and the discomfort of flights.

TABLE III
Number of conflicts and objective values of the classifier whose $obj[1]$ is smallest in each step of conflict resolution by delaying starting time, seed2

Step	conflicts	obj[0]	obj[1]
1	46	0.0387423	460000
2	26	0.0386249	266300
3	15	0.0386493	159300
4	4	0.0385274	50500
5	0	0.0385526	11100

TABLE IV

GENERATION-RULE TRACK OF THE CLASSIFIER WHOSE OBJ[0] IS THE SMALLEST, SEED2

rule	usage	gen	id	father id	content
0	3	0	2092	0	IF (RWD = Cross Down) AND (SL = Medium High) AND (U $\geq$ 0.2162147) Then 5
1	64	0	4184	0	IF (RWD = Head) AND (SL = No-Storm) AND (U $\geq$ 0.5942413) Then 2
2	10	0	2094	0	IF (RWD = Tail) AND (SL = Low) AND (U $\geq$ 0.4757735) Then 5
		33	8320	2094	IF (RWD = Tail) AND (SL = Low) AND (U $\geq$ 0.4757735) Then 4
3	11	0	2095	0	IF (RWD = Tail) AND (SL = Low) AND (U $\geq$ 0.251077) Then 3
		30	7988	2095	IF (RWD = Tail) AND (SL = Low) AND (U $\geq$ 0.8248061) Then 3
		57	10677	7988	IF (RWD = Tail) AND (SL = Medium) AND (U $\geq$ 0.8248061) Then 3
4	1627	0	2096	0	IF (RWD = Cross Up) AND (SL = No-Storm) AND (U $\geq$ 0.5868399) Then 3
5	33	0	2097	0	IF (RWD = Cross Down) AND (SL = Medium Low) AND (U $\geq$ 0.1730221) Then 4
		20	6992	2097	IF (RWD = Cross Down) AND (SL = Medium Low) AND (U $\geq$ 0.1730221) Then 4
6	10	0	4189	0	IF (RWD = Head) AND (SL = Low) AND (U $\geq$ 0.652337) Then 3
7	444	0	4190	0	IF (RWD = Tail) AND (SL = No-Storm) AND (U $\geq$ 0.6701415) Then 3
8	0	0	2967	0	IF (RWD = Tail) AND (SL = High) AND (U $\geq$ 0.4261159) Then 5
		71	12037	2967	IF (RWD = Tail) AND (SL = High) AND (U $\geq$ 0.3700761) Then 5
11	58	0	2970	0	IF (RWD = Cross Up) AND (SL = Medium High) AND (U $\geq$ 0.4347744) Then 3
		88	13866	2970	IF (RWD = Cross Up) AND (SL = Medium High) AND (U $\geq$ 0.1039975) Then 3
12	0	0	2971	0	IF (RWD = Cross Up) AND (SL = High) AND (U $\geq$ 0.8356706) Then 5
13	9	0	2972	0	IF (RWD = Tail) AND (SL = Medium Low) AND (U $\geq$ 0.02297481) Then 2
14	0	0	2973	0	IF (RWD = Head) AND (SL = High) AND (U $\geq$ 0.1870232) Then 6
		22	7187	2973	IF (RWD = Head) AND (SL = High) AND (U $\geq$ 0.1870232) Then 6
		80	12970	7187	IF (RWD = Tail) AND (SL = High) AND (U $\geq$ 0.1870232) Then 6
15	2137	0	2974	0	IF (RWD = Cross Down) AND (SL = No-Storm) AND (U $\geq$ 0.4222877) Then 3
16	12	0	2975	0	IF (RWD = Cross Up) AND (SL = Medium High) AND (U $\geq$ 0.018061) Then 3
		82	13243	2975	IF (RWD = Cross Up) AND (SL = Medium High) AND (U $\geq$ 0.018061) Then 5
18	0	0	2416	0	IF (RWD = Head) AND (SL = Medium High) AND (U $\geq$ 0.8860291) Then 4
		11	6160	2416	IF (RWD = Tail) AND (SL = Medium High) AND (U $\geq$ 0.8860291) Then 4
		57	10678	6160	IF (RWD = Tail) AND (SL = Medium High) AND (U $\geq$ 0.8860291) Then 4
21	10	0	4051	0	IF (RWD = Cross Up) AND (SL = No-Storm) AND (U $\geq$ 0.7535361) Then 4
		18	6868	4051	IF (RWD = Cross Up) AND (SL = Medium) AND (U $\geq$ 0.7535361) Then 4
22	21	0	4052	0	IF (RWD = Head) AND (SL = No-Storm) AND (U $\geq$ 0.3961009) Then 4
23	100	0	4053	0	IF (RWD = Cross Up) AND (SL = Medium) AND (U $\geq$ 0.2372111) Then 4
		37	8749	4053	IF (RWD = Cross Up) AND (SL = Low) AND (U $\geq$ 0.2372111) Then 4
24	0	0	4054	0	IF (RWD = Head) AND (SL = Low) AND (U $\geq$ 0.7490645) Then 4
		31	8174	4054	IF (RWD = Tail) AND (SL = Low) AND (U $\geq$ 0.7490645) Then 4
		44	9427	8174	IF (RWD = Tail) AND (SL = Medium) AND (U $\geq$ 0.7490645) Then 4
25	10	0	4055	0	IF (RWD = Head) AND (SL = Medium Low) AND (U $\geq$ 0.8258837) Then 3
26	0	0	4056	0	IF (RWD = Tail) AND (SL = Medium) AND (U $\geq$ 0.7132748) Then 2
28	9	0	4058	0	IF (RWD = Cross Down) AND (SL = High) AND (U $\geq$ 0.01458683) Then 4
		30	8070	4058	IF (RWD = Cross Down) AND (SL = High) AND (U $\geq$ 0.7737651) Then 4
32	0	0	2124	0	IF (RWD = Head) AND (SL = Medium) AND (U $\geq$ 0.7893357) Then 6
33	0	0	2125	0	IF (RWD = Tail) AND (SL = Medium High) AND (U $\geq$ 0.777858) Then 1
		79	12870	2125	IF (RWD = Tail) AND (SL = Medium High) AND (U $\geq$ 0.4513453) Then 1
34	2	0	2126	0	IF (RWD = Cross Up) AND (SL = High) AND (U $\geq$ 0.7757045) Then 0
		43	9322	2126	IF (RWD = Cross Up) AND (SL = High) AND (U $\geq$ 0.6241231) Then 0
37	14	0	2129	0	IF (RWD = Head) AND (SL = Medium Low) AND (U $\geq$ 0.1078173) Then 4
		74	12427	2129	IF (RWD = Head) AND (SL = Medium Low) AND (U $\geq$ 0.1078173) Then 3
39	72	0	2131	0	IF (RWD = Head) AND (SL = No-Storm) AND (U $\geq$ 0.04602336) Then 3
41	0	0	2133	0	IF (RWD = Tail) AND (SL = Low) AND (U $\geq$ 0.4066338) Then 4
42	0	0	2134	0	IF (RWD = Head) AND (SL = High) AND (U $\geq$ 0.3508267) Then 0
		4	5405	2134	IF (RWD = Head) AND (SL = Medium High) AND (U $\geq$ 0.3508267) Then 0
43	0	0	4226	0	IF (RWD = Head) AND (SL = Medium High) AND (U $\geq$ 0.41808) Then 2
44	0	0	4227	0	IF (RWD = Head) AND (SL = Medium) AND (U $\geq$ 0.6708537) Then 5
		35	8513	4227	IF (RWD = Head) AND (SL = Medium) AND (U $\geq$ 0.6708537) Then 6
46	1144	0	2138	0	IF (RWD = Cross Up) AND (SL = No-Storm) AND (U $\geq$ 0.2928225) Then 3
50	2598	0	2142	0	3

D. *Time Consumption and Result Comparison*

The experiment was completed in 27 minutes 33 seconds. If we use 20 CPUs, this time can be reduced to 7 minutes. Therefore, running the system in the distributed environment can scale the application with higher number of flights and larger air space.

We did another experiment where all flights use straight-line trajectories from origin to destination and conflicts between flights are resolved by propagating delays of conflicting flights as presented above. The two objectives $obj[0]$ and $obj[1]$ provided by this experiment are 0.0169062 and 48600 respectively, while $obj[0]$ is from about 0.0144 to 0.0386 and $obj[1]$ from 11,100 to 43,500 when we did experiment with seed2. All the solutions $obj[1]$ provided by the experiment with seed2 is less than 48600 and also a lot of solutions with $obj[0]$ less than 0.0169062. This established more confidence in the proposed approach.

VII. CONCLUSION

In this paper, we describe an efficient Multi-Objective Learning Classifier System for multi-flight navigation using NSGAII. A classifier is represented by a set of rules. These rules navigate all the flights in the airspace in every simulation step based on the interaction between flights with the air traffic environment such as wind, storm and other flights. This system continually learns and refines the rules of classifiers by NSGAII to find out the non-dominated set of

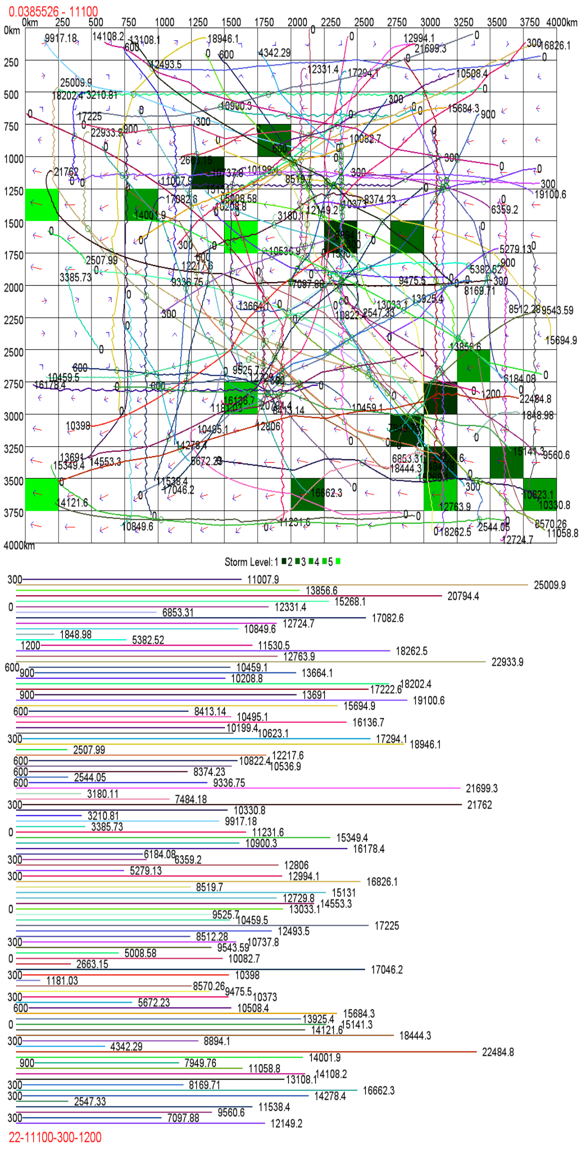


Fig. 4. Simulated Trajectories and Time Spans in Final Conflict Resolution Step of the classifier whose *obj*[1] is the smallest, seed2

classifiers to minimize distance, discomfort, minimize total delay time of flights and avoid conflicts between flights. The results show that the system works well with a high number of flights and can provide a variety of solutions for different objectives. Many solutions are better than the baseline straight-line trajectories with delay-propagation.

References

[1] S. Michot, N. Stancioi, C. McMullan, S. Peeters, and S. Carlier, "AERO2k Flight Movement Inventory," EUROCONTROL Experimental Centre, Bretigny-sur-Orge, France, Tech. Rep. EEC/SEE/2003/005, December 2003.

[2] J. E. Penner, D. Lister, D. Griggs, D. Docken, and M. MacFarland, *Aviation and the Global Atmosphere*. New York: Cambridge University Press , 1999.

[3] National ATCs Association, "ATC: By the Numbers," 2010. [Online]. Available: http://www.natca.org/mediacenter/bythenumbers.msp

[4] S. Mittal and K. Deb, "Three-dimensional offline path planning for uavs using multiobjective evolutionary algorithms," in *IEEE-CEC*, Sept 2007, pp. 3195–3202.

[5] A. Richards and J. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," in *American Control Conference*, Anchorage, AK, May 2002.

[6] K.-Y. Chen, P. A. Lindsay, P. J. Robinson, and H. A. Abbass, "A hierarchical conflict resolution method for multi-agent path planning," in *Proceedings of the Eleventh IEEE-CEC*, 2009, pp. 1169–1176.

[7] J. Doebbler, P. Gesting, and J. Valasek, "Real-time path planning and terrain obstacle avoidance for general aviation aircraft," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2005.

[8] S. A. Bortoff, "path planning for uavs," in *Proceedings of the American Control Conference*, 2000.

[9] J. Kehoe, A. Watkins, and R. Lind, "A time-varying hybrid model for dynamic motion planning of an unmanned air vehicle," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2006.

[10] J. Amin, J. Bokovic, and R. Mehra, "A fast and efficient approach to path planning for unmanned vehicles," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, August 2006.

[11] P. R. Chadler, M. Pachter, and S. Rasmussen, "Uav cooperative control," in *American Control Conference*, June 2001.

[12] T. W. McLain and R. W. Beard, "Trajectory planning for coordinated rendezvous of unmanned air vehicles," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, August 2000.

[13] G. Yang and K. Vikram, "Optimal path planning for unmanned air vehicles with kinematic and tactical constraints," in *41st IEEE Conference on Decision and Control*, December 2002.

[14] H. Wong, V. Kapila, and R. Vaidyanathan, "Uav optimal path planning using c-c-c class paths for target touring," in *43rd IEEE Conference on Decision and Control*, December 2004.

[15] R. Windhorst, M. Ardema, and D. Kinney, "Fixed-range optimal trajectories of supersonic aircraft by first-order expansions," *Journal of Guidance, Control, and Dynamics*, vol. 24, no. 4, pp. 700–709, 2001.

[16] N. Yokoyama and S. Suzuki, "Modified genetic algorithm for constrained trajectory optimization," *Journal of Guidance, Control, and Dynamics*, vol. 28, no. 1, pp. 139–144, 2005.

[17] M. Anderson, J. Lopez, and J. Evers, "A comparison of trajectory determination approaches for small uav's," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, August 2006.

[18] M. R. Jardin and A. E. B. Jr., "Neighboring optimal aircraft guidance in winds," *Journal of Guidance, Control, and Dynamics*, vol. 24, no. 4, pp. 710–715, 2001.

[19] J. H. Holland, "Processing and processors for schemata," *In Associative information processing*, pp. 27–146, 1971.

[20] J. H. Holland and J. S. Reitman, "Cognitive systems based on adaptive algorithms," *Pattern directed inference systems*, pp. 313–329, 1978.

[21] S. F. Smith, "Flexible learning of problem solving heuristics through adaptive search," in *In Proceedings of IJCAI*, 1983, pp. 422–425.

[22] K. A. D. Jong and W. M. Spears, "Learning concept classification rules using genetic algorithms," in *In Proceedings of IJCAI*, Sydney, Australia, 1991, pp. 651–656.

[23] C. Janikow, "Inductive learning of decision rules in attribute-based examples: a knowledge-intensive genetic algorithm approach," Doctoral dissertation, University of North Carolina, Chapel Hill, July 1991.

[24] J. M. Garrell, E. Golobardes, E. Bernad´-Mansilla, and X. Llorà, "Automatic diagnosis with genetic algorithms and case-based reasoning," *Artificial Intelligence in Engineering*, vol. 13, pp. 367–372, 1999.

[25] X. Llor'a, D. E. Goldberg, and . B.-M. E. I. Traus, I., "Accuracy, parsimony, and generality in evolutionary learning systems via multiobjective selection," *LNAI*, vol. 2661, pp. 118–142, 2003.

[26] J. Bacardit, "Pittsburgh genetics-based machine learning in the data mining era: Representations, generalization, and run-time," PhD thesis, Ramon Llull University, Barcelona, Spain, 2004.

[27] A. Bicchi, and L. Pallottino, "On optimal cooperative con ict resolution for air traffic management systems," *IEEE Transaction on Intelligent Transportation Systems*, vol. 1, no. 4, pp. 221–231, 2000.

[28] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.