Spring 2015

# Air traffic management under uncertain weather impact

Le Gao
*Purdue University*

*Graduate School Form 30*
*Updated 1/15/2015*

**PURDUE UNIVERSITY**
**GRADUATE SCHOOL**
**Thesis/Dissertation Acceptance**

This is to certify that the thesis/dissertation prepared

By Le Gao

Entitled

AIR TRAFFIC MANAGEMENT UNDER UNCERTAIN WEATHER IMPACT

For the degree of Master of Science in Aeronautics and Astronautics

Is approved by the final examining committee:

Dengfeng Sun
Chair

Inseok Hwang
Co-chair

Jianghai Hu
Co-chair

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Dengfeng Sun

Approved by: Weinong Chen 4/9/2015

Head of the Departmental Graduate Program Date

# AIR TRAFFIC MANAGEMENT UNDER UNCERTAIN WEATHER IMPACT

A Thesis

Submitted to the Faculty

of

Purdue University

by

Le Gao

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Aeronautics and Astronautics

May 2015

Purdue University

West Lafayette, Indiana

# ACKNOWLEDGMENTS

This research is supported and directed by Prof. Dengfeng Sun. I thank Prof. Dengfeng Sun who provided insight and greatly assisted this research. I thank Christabelle Bosson for assistance on using FACET.

Thanks to the providers of FACET software as well as CIWS data. Thanks to all online documents which helped me in programming.

TABLE OF CONTENTS

Page

# LIST OF TABLES

# LIST OF FIGURES

# SYMBOLS

| | |
|---|---|
| $N$ | number of cells in one level |
| $M$ | order of magnitude of polygons in weather data |
| $T$ | current time when data of each frame is recorded |
| $f$ | forecast time |

# ABBREVIATIONS

| | |
|---|---|
| ZAU | Chicage center |
| CIWS | Corridor Integrated Weather System |
| CWAM | Convective Weather Avoidance Model |

# ABSTRACT

Gao, Le M.S.A.A., Purdue University, May 2015. Air Traffic Management under Uncertain Weather Impact . Major Professor: Dengfeng Sun.

Air traffic management under uncertain weather impact is a difficult problem. In this thesis, by dividing the whole air traffic systems into small cells, a new method using database to integrate weather and flight data is proposed. An algorithm has been developed to project weather data onto cells. This research also uses a predictive view to find flights which are affected by severe weather. And a prototype detour algorithm is applied on those flights based on the status information of each cell.

# 1. INTRODUCTION

## 1.1 General idea

A typical method used in air traffic management is linearizing air flow and using linear equations to describe the system [1]. Such model uses optimization tools to get an optimal solution of the whole system. Models of this kind can perform ground hold or delay in the sky. However, this kind of model shares a feature that it must operate on stationary airport [2] or cells on designed routes [3]. Another big problem in air traffic control is severe weather avoidance. However, when describing weather systems such as clouds or thunderstorms movement, it may use geometry such as polygons or even velocity vector field. Common air traffic models are totally different with weather models.

This research is trying to find a method to combine air traffic system and weather system together. Considering two dynamic systems with different models at the same time may make the question too complex. The task is to find a new model which can represent air traffic as well weather information. In this thesis, I tried to divide whole air space into a large amount of small rectangles which called cells and using database as an integrating platform. Weather data and flight information are all projected onto each cell. Thus, from the view of each cell, it can tell whether this flight will have collision with other flights, or whether it is affected by severe weather. In this research, database is more than simply storing and providing data. It can join some task in finding aircraft which needs detour. Whats more, when planning detour route, detour program can fetch status of each cells it needed in database. It doesnt need to consider multiple polygons which represent weather information. Another feature of this research is it trying to achieve a predictive view. Program not only monitors what is happening at the current, but also tries forecast which

aircraft may need to detour in the future. Therefore, aircraft can start awareness and planning detour ahead of time. With this system, collision avoidance and severe weather avoidance can be done at the same time.

Section 1.3 declares some general information and global values which will be used in all other parts in this thesis. Chapter 2 introduces how main focus area is defined and divided into cells. Chapter 3 describes the organization of each table in database. Chapter 4 discusses on processing the flight data. Chapter 5 shows how weather data is projected onto each cell. Chapter 6 will discuss the detour algorithm including avoiding severe weather and collision.

### 1.2 Software and data source

In this research, origin data is provided by FACET. FACET is written in Java. Thus Java is required to start FACET and to extract flight data and weather data. The weather data, which is CIWS [4], contains VIL/Precipitation and Echo top information. All those information are represented by multiple polygons. Those data also contain weather forecast information which extended to 2 hours. Flight data contains flight plans, current flight position, height, speed and many other related data. Python is main programming language used in this research. It is easy in programming and debugging. And whats the most important thing is, python is easy at visualization. There are many data requires to be shown in figures to test its accuracy. Database, as a main platform in this thesis, is chosen as MySQL. MySQL python connector is also used to communicate between python and MySQL. In visualization part, Pygame is used as the main engine. It has a simple logic and easy to start with.

### 1.3 Research environment

The flight data and CIWS data used in this research is on date 4/10/2013, starts from 6:55:50 UTC. Refresh rate of flight data is 1 minute. Refresh rate of CIWS

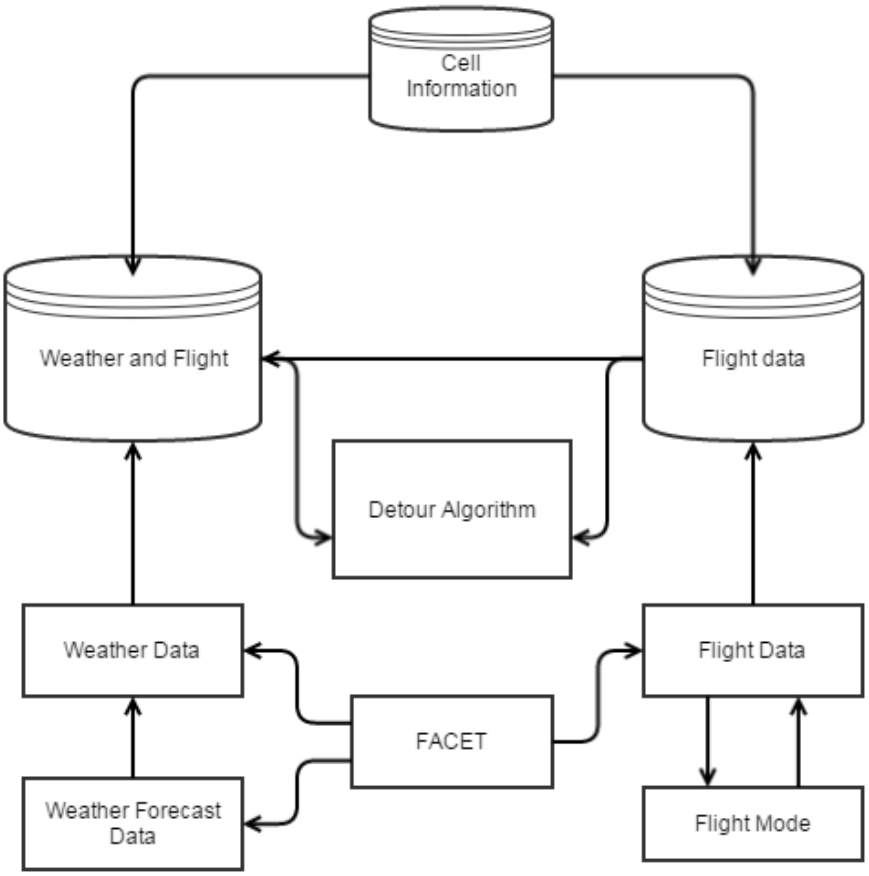



Fig. 1.1. Data Flow Diagram

data is 5 minutes. Refresh rate of forecast weather data is also 5 minutes. In this research, all data is considered as discrete. The refresh rate is set as the minimum value, which is one minute in flight data. Each minute, there will be a new frame of data even weather data is not provided. To get predictive view, forecast time is set as 10 minutes, or ten frames for each minute. For convenience, time used in this research is relative time. Time $T$ starts from 0 minutes. Every forecast time $f$ must related with one current time $T$. Time 0 means 6:55:50 UTC on date 4/10/2013. Time 1 means 6:56:50 UTC, and etc. Time 5 forecast 5 means at time 7:00:50 UTC, predicting the situation at 7:05:50 UTC.

All data and database programs are written and run on computer platform with processor Intel i5-3230M @ 2.60 GHz. Memory is 8 GB, and hard drive used is a normal HDD drive.

# 2. CELL DIVISION

First of all, a main stage is required to hold and run simulations. There are three main basic concerns when choosing main focus area. First, the size of the area must be suitable to be divided and analyzed. If the area is chosen too large, the number of cells will be too large for ordinary computer, such as computer platform used in this thesis, to manage. Second, the severe weather needs to affect this area. It is needless to say this research must interact with severe weather. Otherwise, discussion on weather affect will be pointless. Third, it must be a busy area such that it will have quite a large amount of flights inside the sector. If there are only several flights inside this area, there will be too few flight data to be analyzed. In this project, Chicago center (ZAU) is chosen as main focus area. ZAU center may not be the best choice, but it meets the basic requirements.

The most common way to define a position on earth is using latitude and longitude coordinates. Thus unit of latitude and longitude coordinates is in degree. However, the aircraft speed is in unit knot, which is 1 nautical mile per hour. In the definition of one nautical mile, 1 nautical mile is 1/60 of a degree of any great circle on Earths surface. Therefore, speed knot can be converted into degree per minute. This conversion will be discussed in section 4.3. In short, all units used in coordinate system to represent positions in this research are degree. The sphere effect is not considered in this research. ZAU center is stretched into plane.

However, choosing the main focus area exactly as the contour of ZAU center does not bring any convenience. The ZAU center is an irregular polygon which contains several small sectors. It is obviously easier to test a point inside a rectangular than test a point inside a polygon. To simplify the problem at the very beginning, main focus area is enlarged to a rectangle which contains ZAU center. The geometry information of main focus area and each cell is show in Table 2.1. Interested altitude

chosen is normal airspace which most aircraft will maintain a level flight at this altitude. Very low airspace is not chosen because it will contain a lot of taking-off and landing aircraft. ZAU center has a large international airport. If considering those low levels, it will incorporate airport management model, which has to consider ascending and descending motion of aircraft. Besides, the cell size will be too large when taking off and landing because aircraft speed is much slower at lower levels. One cell may contain the whole airport area. It is obvious not feasible using the same cell size. On the other hand, there still have some aircraft above 380 level. I have found aircraft which operate on 400 level. However, there will be no more than 10 aircraft operating at this high level at a time. Compared with large amount aircraft in middle level, adding more levels in very high altitudes will not bring great difference in this research.

When dividing an area into small cells, there are three possible choices. As the sphere effect is neglected, and ZAU center is stretched into plane, triangle, rectangle, and hexagon are common planar polygons which can cover the whole area without leaving any region uncovered. Here, because ZAU center is enlarged to a rectangle, it will be the easiest way to divide it into multiple small rectangles.

Unit of altitude used in this research is in feet. Level is a parameter which related with altitude. In most cases, level is altitude valued divided by 100. Height of each cell is one level.

Table 2.1.
Main Focus Aarea and Cell Information

| | | | |
|---|---|---|---|
| Left boundary | 266.39999 | Upper boundary | 44.341667 |
| Right boundary | 275.31665 | Lower boundary | 40 |
| Cell number in x direction | 50 | Cell number in y direction | 33 |
| Cell length in x direction | 0.1788 | Cell length in y direction | 0.1332 |
| Total cells in one level | 1650 | Interested Height | 180-380 level |

By definition, one degree in latitude is 60 nautical miles. However, the length in longitude varies at different latitude. The ZAU center average latitude is 41.845 degree. At this latitude, one degree in longitude is 44.697 nautical miles. Because ZAU center only has 4.9917 degree in wide, the difference in longitude can be neglected. Aircraft are considered to fly at least 2 nautical miles per minute. This value considers the speed of some propeller aircraft. The size of each cell is set according to this assumption.

One major target of this research is to avoid collision. As the whole ZAU center is divided into small cells, the easiest way to do so is to set each cell can only contain one aircraft at a time. Meanwhile, each cell is set to be larger than the average distance which aircraft can fly over in one minute. Once one aircraft flies into one cell, it will exclude all other aircraft to fly into this cell. This method can basically avoid collision. Although theoretically, if each cell need to be collision free, it has consider not only the cell which aircraft is currently located, but also the cell which aircraft is going to fly into. Moreover, because aircraft wont fly in exactly $x$ and $y$ direction, it may visit nearby cells in a short time. Because this research is just testing a new model, cells which aircraft will fly over or visit is not considered at current step.

Besides, according to the analysis above, if the length of each cell is the distance which aircraft can fly over in one minute, the capacity of each cell should be 1. However, the capacity of each cell is set as 2 not 1. However, in actual division test, the cell number is too large if cell length is 2 nautical miles. Using the division method in Table 2.1 (which is 8 nautical miles in length), it has more than 1500 cells per level, and almost 200 levels in height. This makes the whole airspace contains more than 3,000,000 cells. On this order of magnitude, it will cost several minutes create a table of cell information. If a smaller cell is used, for example, length and width are divided by 4, the number of cells will become more than 60,000,000 cells. This will cost almost an hour to create a table of cell information. What the most important thing is, the picture of divided cells will concentrated together and they cannot be seen clearly

even with 1080p resolution unless the picture is enlarged. Under the consideration of those problems, cells are chosen as larger ones.

Because flight data provided in FACET is collision free, it is supposed that if those data imported in the system used in this research, it should be collision free at the very beginning. However, if a larger cell is used, it may happen that two aircraft located in the same cell according to the origin data. This may fail the condition that aircraft are collision free in origin data. Just because each cell is 16 times larger than it should be, aircraft can have more space in single cell. It will be a reasonable assumption that two aircraft won't collide in such large space. Or in other words, there exists enough space for two aircraft to avoid each other in such large space. For example, one aircraft can fly at in upper half level and the other one fly at a lower upper level. Besides, the program will monitor whether each cell contains two or more aircraft. If this situation happens, it will send a message out and record such result. And, throughout simulation of 40 minutes, there is no cell which contains two aircraft. The actual chance which aircraft are inside one cell is very low. The collision free condition satisfies at most of time.

The result of cell division is shown in Appendix A. Black lines in those Figures show the ZAU center and its small sectors. The largest Green box is the main focus area. It is slightly larger than ZAU center to make every small cell in complete size. Green rectangles show divided cells. In Figures, they have small gap between some rectangles while each rectangle should be adjacent to each other. This is caused by some features of Pygame. The draw function in Pygame only accepts INT type. The gap is caused by rounded number when transferred to draw function. However, although those gaps should not exist, they bring some convenience in debugging, such as locating a specific cell. Therefore, those gaps are kept during developing.

# 3. DATABASE DEFINITION

Database is the key platform in this research. Almost all functions are achieved with the help of database. Database used in this research is MySQL. MySQL has programmable language to use. But it is not suitable for writing complex data structure and algorithm. What the most important thing is, I am not very familiar with this language and it does not contain graphical interface. Therefore, all complex algorithm and visualization is done by Python. Communication between MySQL and Python program is done by MySQL python connector. The key feature when using Python connector is it can use a For loop to read out all required data in the database. And as part of feature of python, it is not sensitive to types. One array can contain data with different types. This is especially helpful when reading data out of database because one may need to read out data in multiple types at the same time.

The main idea in this research is to create a table which contains weather information and flight information of each cell. It will have a great amount of rows and columns. Besides, there will have a large amount of fetching data operations. Moreover, the data structure created in MySQL will greatly influence the efficiency of the fetching operations. On the other hand, flight avoidance and weather avoidance need to work under a real time condition. Executing speed is another main concern. Thus carefully choosing data type and data structure in MySQL will be a very important issue in this research.

This chapter will discuss all three types of tables used in MySQL. It will include the column name, data type, and detailed explanation of each column. Besides, primary key is the index which is used by MySQL to search and save data. The definition of primary keys of each table will be introduced. A good definition of primary keys can greatly boost the speed of fetching data. As this is the first time I define and use a

database, the data structure in this part may not be the best options. But according to actual simulation result, it at least meets the time constraint in this research.

### 3.1 Cell Information table

This table is designed to save cell related data which is generated in chapter 2. With this table, each time when cell geometry related information is required, the program does not need to calculate (though it is not a complex procedure). The program only needs to fetch cell data in this table. The main advantage is to avoid the error caused by float and double calculation.

Table 3.1 contains the following data (all units are in degree)

Table 3.1.
Cell Status Table data Type Definition

| Column name | Data type | Detail |
|---|---|---|
| ID | unsigned int(10) | ID of each cell. |
| center_x | double | The longitude of center point |
| center_y | double | The latitude of center point |
| left_boundary | double | The left boundary of this cell |
| right_boundary | double | The right boundary of this cell |
| up_boundary | double | The upper boundary of this cell |
| low_boundary | double | The lower boundary of this cell |
| level | unsigned smallint(5) | Level of the cell is defined as height divided by 100. |
| height | unsigned mediumint(5) | This value denotes the lowest limit of this cell. Height of each cell is 100 feet. |

The primary key in this table is ID. Because ID and each cell is one to one relation, and in most cases, program will use id to find geometry information of each cell, it is the best option to use ID as primary key.

The time constraint of this table is not important because this table will be built at the very beginning. As long as the main focus area is ZAU center, the division of each cell will not change. Although building this cell requires several minutes to finish, this table wont be rebuilt in the future of this research.

### 3.2 Flight Data Table

This table contains all flight related data information: flight position, way point, speed, height, target height, landing airport and a sequence of forecasted flight position. Details of how forecasted flight positions are calculated will be discussed in chapter 4.

Data in this table is organized as Table 3.2. (Note, $i$=1,2. $f$=1,2,,10)

The primary keys in this table are Flight name and Time. There are two primary keys in this table because any column in this table is not one to one relation with each row. It is obvious that one flight will lasts for quite a long time, and at a single time, there will have multiple flights in operation. The only method is to combine flight name and time together to define one single row. At time $T$, there will be only one aircraft with that specific flight name.

In origin data, there is a list of multiple way points which designate the route of each flight. In this table, only two way points are saved. This may be one problem in the definition of this table because in simulation. If two way points are close to current aircraft position, the aircraft can fly over the second way point. Because way point information is needed when defining detour destination, some flights with this problem may have no detour destination to be provided. Nevertheless, this situation does not happen frequently and can be solved easily by extending the length of this table. And length of way point data varies from flights to flights, while columns of

Table 3.2.
Flight Data Table data Type Definition

| Column name | Data type | Detail |
|---|---|---|
| FlightName | char(8) | This is the unique flight identification given by FACET database. One flight name only corresponds to one aircraft at the same time. |
| Time | unsigned int(10) | This represents current time $T$ when data of this row is recorded, in minute |
| PositionX | float | Longitude value of this flight at time $T$, in degree |
| PositionY | float | Latitude value of this flight at time $T$, in degree |
| Height | float | Altitude of this flight at time $T$, in feet |
| TargetHeight | float | The altitude which this flight is supposed to fly at, in feet |
| Speed | float | Speed of this flight, in knot |
| WayPointX$i$ | float | Longitude value of next $i$-th waypoint, in degree |
| WayPointY$i$ | float | Latitude value of next $i$-th waypoint, in degree |
| LandingAirport | char(5) | Target airport which this flight is planned to landing at |
| LandingCenter | unsigned smallint(5) | Center ID at which the landing airport is located |
| F$f$_CellId | int(11) | Cell id number of which the $f$-th forecasted position is inside |
| F$f$_PositionX | float | Longitude value of the $f$-th forecasted position |
| F$f$_PositionY | float | Latitude value of the $f$-th forecasted position |

database must be set when creating this table. Saving all way point data is infeasible at this point of view.

Even with saving only two way points, this table has large amounts of columns. For just ten minutes forecast, it will have at least 30 columns storing flight prediction related data. Such long table is generated by using python script to create a repeated command.

### 3.3 Weather and Flight Table

This table is a functional table which stores almost all important information for this thesis. This table can tell which cell is affected by severe weather or which cell is overloaded with aircraft. I once tried to build all simulation data into one single table. However, this leads to too many rows in one table and makes fetching operation extraordinary slow. The time constraint is most important when fetching data in this table. Therefore, to boost fetching speed, this table will have multiple copies. Each copy has the name "weather_fligh_$T$", where $T$ represents time in minute. For example, if the program is dealing with flight and weather information at 50 minute, the table it worked with will be "weather_fligh_50". In simulation, this method greatly boosts the operation speed of fetching data because the table name itself is another kind of searching index.

For each forecast minute, there has four columns data. Data in this table is organized as Table 3.3. (Note, $f$=0,1,2,3,,10)

In this table, F$f$_result variable can directly show which flight should be detoured. By combining data in F$f$_capacity, and F$f$_result, the program can know which kind of influence this flight is facing. All details will be discussed in detail in 5.

F$f$_occupied and F$f$_flight_name information are related with flight data table. Flight data table must be built to import data of this type. Data of this column will not be imported until weather data is imported. F$f$_capacity is an information related with weather. Usually, flight name is string with no more than 6 chars. Because one

Table 3.3.
Weather and Flight Table data Type Definition

| Column name | Data type | Detail |
|---|---|---|
| id | unsigned int(10) | id of cells. Representing the cell with the same id in cell information table |
| F$f$_capacity | smallint(6) | Capacity of this cell at Forecast time $f$. It will set as default value 2 if there is no severe weather in this cell. Otherwise, it will set as 0. |
| F$f$_flight_name | char(15) | This column saves the flight name which comes into this cell at forecast time $f$. If there are two flights in this cell, a blank will separate the name of those two flights. |
| F$f$_occupied | smallint(6) | This value shows there are how many aircraft in this cell at forecast time $f$. |
| F$f$_result | smallint(6) | This value is a calculated value which equals capacity minus occupied. If the value in this cell is less than zero, it means all flights in this cell is required to detour. |

cell can hold 2 aircraft, F$f$_flight_name is set as a string with 15 chars in case two aircraft fly into the same cell.

This table is also a long table. It requires a Python script to create this table. Because this table has multiple copied, creation script will be called at the beginning of simulation of each minute.

# 4. FLIGHT DATA MANAGEMENT

This chapter mainly discuss about flight data related process. Flight data provided by FACET is massive and unprocessed. Not all flight data provided in FACET is absolute correct. On the other hand, it does not provide how those data are measured, which makes error detection even more difficult. Python will be used to extracted the useful part of flight data, and convert those data into a format which can be imported into the database. Besides, a very simple algorithm will be introduced in this chapter to exclude error data. This is a very important step to guarantee successful simulation in following parts. Also, a simple flight model is used in this chapter to predict flight path. Because flight model is separated from flight data, any other detailed flight model can be applied to this system to increase accuracy.

## 4.1 Flight Data Pretreatment

Data provided by FACET is massive and have multiple parameters related with single flight. Not all of them are helpful. Therefore, when extracting origin data, some parameters which may be useless to this research at first thought will not be extracted. Extracted data are saved in file name with format 'flight_data_i.txt', where $i$ means $i$-th minute. Origin data file saves following format:

First line: Flight name, latitude, longitude, height, target height, level, speed, heading, whether this flight is delayed, way point count $j$(this number tells how many next lines are way point data), target airport, target airport id, center id where target airport is located.

The following $j$ lines: way point $k$ latitude, way point $k$ longitude.

The simulation in FACET is minute by minute. Therefore each data file contains all flights in one minute. Flight name and airport related data is simple and does not need explanation.

Longitude value in this research is quite different. In definition, longitude value has a range from $-180$ degree to 180 degree. However, the longitude of way points has a range from 0 degree to 360 degree. On the contrary, value range of longitude and latitude value is the same as definition. The system cannot work with two different units. To unify units, and simplify graphic display in future programming, all longitude in this research has the same range as way points, which is from 0 to 360 degree. When read from origin data file, longitude of position and weather data are added 360 if they are less than 0.

Height, target height and level data are related. Units of height and target height are feet. Level always equals height divided by 100. Target height is the designed height which aircraft should fly at. When aircraft is climbing, this data will be useful to some degree. Unit of speed is knot as discussed. The unit of speed will be changed when prediction aircraft position.

Delayed data is thought to test accuracy of this research with extracted data. However, after extraction, delayed data are found to be false in all files. It may be an unused data member in FACET. Data of this type can be extracted but it is not managed by any function in FACET. Therefore, there is available data to check accuracy on weather affection forecast. So, an extra program is written to check whether weather affection function works correct or not.

On the other hand, FACET has provided too many flight data than this research actually needs. It not only contains flight data across U.S.A, but also some flights flying all over the world. However, only flights which are inside ZAU center need to be considered. To avoid wasting time processing flight which will not interact with ZAU center, the program will only concern flights which will fly over main focus area. However, only considering flights inside main focus area is not sufficient. Because aircraft are moving from time to time, aircraft can fly into main focus area several

minutes later even it is currently not inside this area. Because this research needs to have a predictive view, those flights cannot be discarded. Therefore, the program do the following to meet that requirement: find the next way point, exclude flight information which has error, forecast ten flight positions, check whether any position is located in ZAU center, and finally write related flight data into database. Figure 4.1 shows the flow chart in this chapter. With this procedure, flights which will fly over main focus area in 10 minutes will also be considered.

## 4.2 Finding Next Way Point

Origin data provided by FACET contains a long way point list from the departing airport to landing airport. However, it doesn't have the information about which way point this aircraft is currently heading. This is very import information to predict flight path no matter what aircraft model used. On the other hand, many aircraft may not fly exactly as planned route. There exist a lot of possible situations which aircraft may leave its designed route. For example, pilots have right to change the route if they found it is not safe. Autopilot on aircraft cant always maintain the aircraft on level flight. Even without considering those situations, aircraft position may have a large measuring error. Sometimes, the ground radar will get a false position when measuring aircraft. Aircraft may looks like jumping left and right continuously. With those possible exceptions in aircraft position, finding next way point becomes a very complex procedure if such data is not provided. This section will mainly discuss the method used in this research to find next way point. Functions mentioned in this section contain two steps in Figure 4.1, checking points inside considering zone and finding next way point.

The reason why it needs to check whether points are inside considering zone is to narrow the searching space as well as excluding strange points. Not all flights will intersect with ZAU center. Besides, flight coordinates are not always continuous. Therefore, it will cause a large problem when considering all flights. For example,

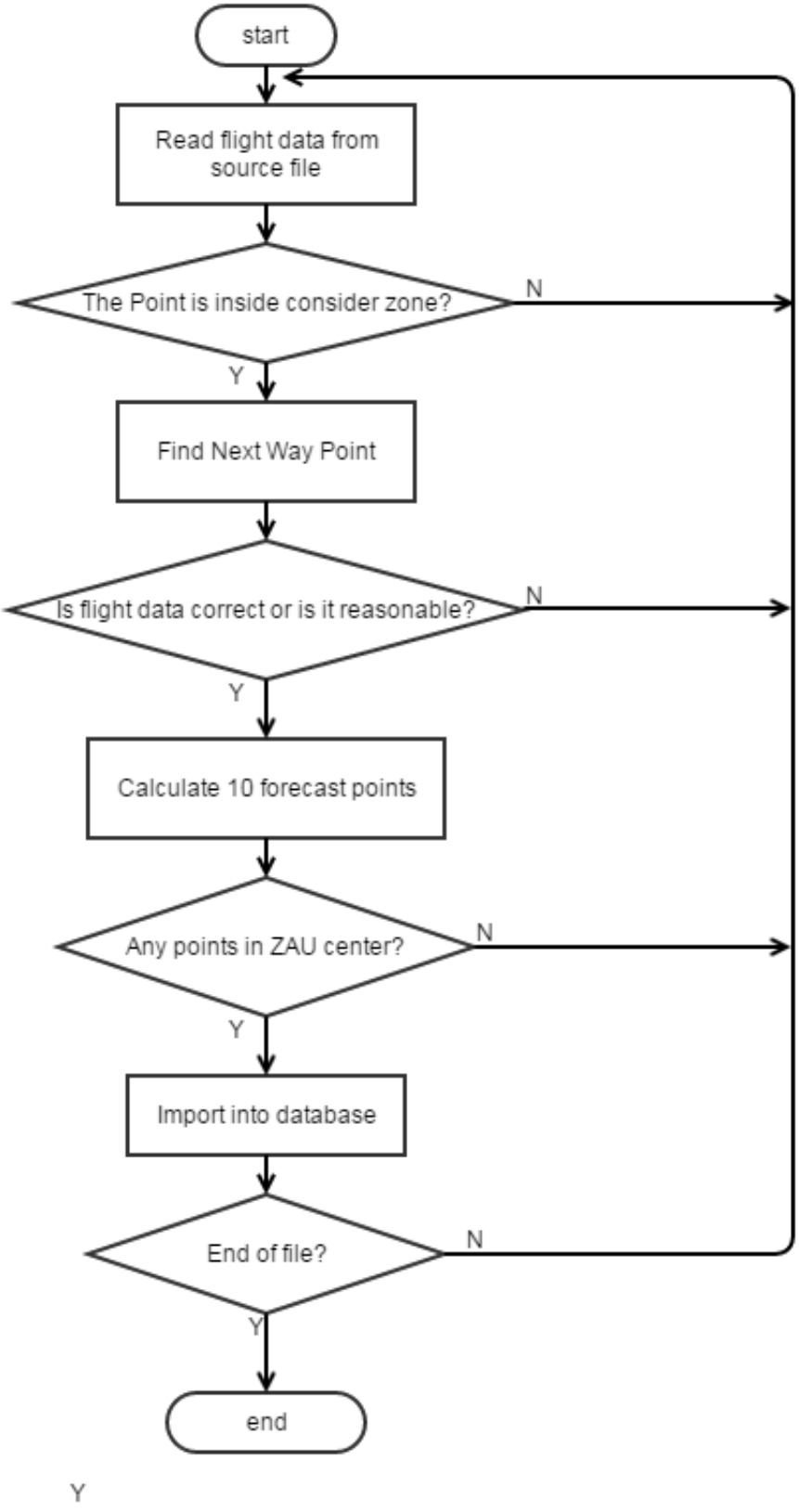



Fig. 4.1. Flow chart in Flight Data Management

when crossing prime meridian, longitude value will change from 360 to 0. From programs view, this aircraft will looks like jumping from right to the left. Fortunately, ZAU center locates inside continuous interval and most flight crossing this area will not cross prime meridian in a short time. Therefore, each way point will be checked whether they are inside considering zone when finding next way point to exclude discontinuous interval. The considering zone is simply set as a larger rectangle contains main focus area. The size of considering zone is defined as: 0 to 80 degree in latitude and 180 to 340 degree in longitude.

The main idea when finding next way point is very simple. If $WayPoint_i$ is the next way point, then current position must be located between $WayPoint_{i-1}$ and $WayPoint_i$. Therefore, the following inequality must be satisfied:

$$x_{WayPoint_{i-1}} <= x <= x_{WayPoint_i} \tag{4.1}$$

$$y_{WayPoint_{i-1}} <= y <= y_{WayPoint_i} \tag{4.2}$$

Figure 4.2 shows a more intuitional view of this inequality.

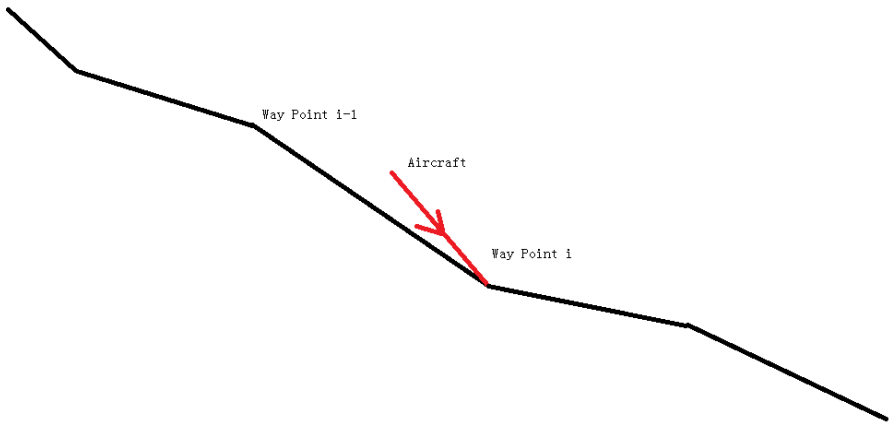



Fig. 4.2. Simple Example of Inequality 4.1 and 4.2

However, Figure 4.2 is just a very simple case. The real situation is much more complicated. There exist situations like Figure 4.3. The planned flight route zigzag

several times. Therefore, in $y$ direction, way point $i$, $i-2$ and $i+2$ all meets inequality 4.1 and 4.2. But, in $x$ direction, only one point is the correct answer. So, from Figure 4.3, it is obvious that if way points are continuously increasing in one direction, judging next way point along this direction will become more accurate. According to those ideas, the following algorithm is design to find next way point.

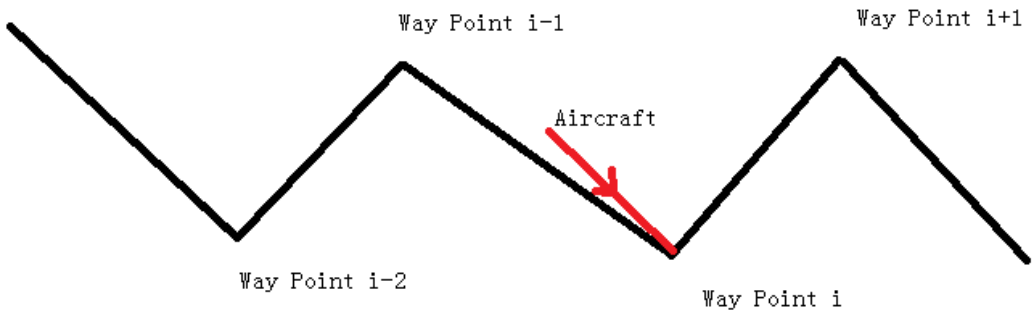



Fig. 4.3. Special Case in Finding Next Way Point

Way points are read in sequence and saved in a list. Each way point can have an ordinal number in that list. The program will count the increment of both directions. If the $i$-th point is greater than the $(i-1)$-th point in x direction, then increment of x direction is added one, and vice versa. Two lists in the program will save all points which satisfy inequality 4.1 and 4.2 in $x$ and $y$ directions. Moreover, each point saved result lists must be inside the considering zone.

After getting all feasible $x$ and $y$ point lists, the next target is to decide which one is more reliable. Because the errors in origin data and irregular shape of flight plans, values in two lists may not be the same or even each list has more than one result.

To discuss about this problem, all possible case are listed in Table 4.1. Condition 1 and 3 are easy to understand. It has the only result that can be returned.

In condition 2, one list is empty and the other is not. It is possible that the nonempty list contains more than 2 results. However, it may lack extra information

to tell which number in that list is more reliable. Therefore, the first result in that nonempty set will be returned. This is one of the uncertain cases.

In condition 4, two lists are length one, but they are different values. This may happen the aircraft is near a place where it has multiple way points nearby just as shown in 4.3, and it is quite far away from its designed path. Under such circumstance, the program will choose the result in the direction which has a higher absolute increment value. As discussed above, higher absolute increment value means higher reliability in that direction. Here, using absolute value is because continuous decreasing is also increasing in opposite direction.

Table 4.1.
Possible Cases in Finding Next Way Point

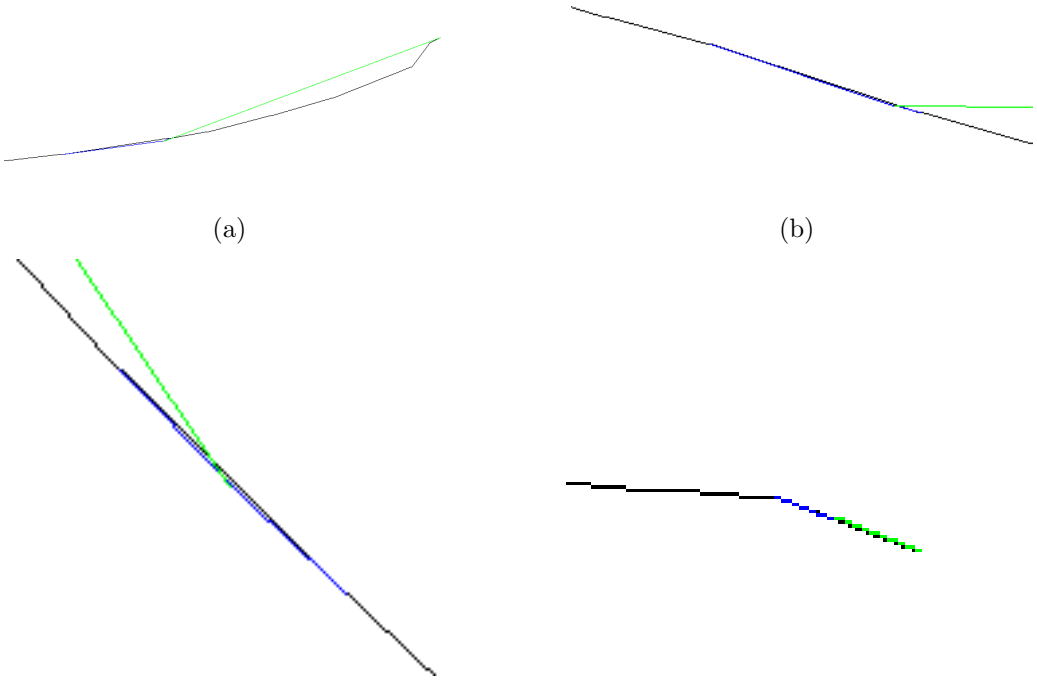
| | |
|---|---|
| 1.Two lists are empty | Return false. There is no next way point |
| 2.One list is empty | Return the first value of the nonempty list |
| 3.Length of both lists is one, and they are the same value | Return the only value |
| 4.Length of both lists is one, but they are different value | Return the value which has a higher absolute increment value |
| 5.Any list is length one | Return the length one list value |
| 6.Length of both lists is larger than one | Find nearest point pair x and y list, and return the one which has a higher absolute increment |

In condition 5, one list may have multiple solutions, but the other one only has one result. In this case, the single solution is always more reliable than the multiple solutions. It rarely happens that single solution has a lower absolute increment value.

Condition 6 is the most complex one. Both directions have more than one solution which meets inequality 4.1 and 4.2. In this case, increment value can tell which list is more reliable, but it cant tell which one in that list is the best. In this very special case, a function is used to find the nearest point pair in both lists. As way points saved in two lists are their ordinal number. The nearest is defined as the smallest difference between two ordinal numbers. For example, 2-nd and 3-rd way points are nearer than 4-th and 6-th way points. And of course, 2-nd and 2-nd is always the nearest pair of way points. Therefore, though this function, two way points can be selected out from two lists. They may be the same, or different. If two points are pointing to the same one, it must be the next way point we want. If two points are not the same value, then use increment value to judge which one is more reliable, just like condition 4. This is another case which has uncertainty.

There exist other methods which can be used to pick out points from two result lists, such as finding way point which is nearest to aircraft position. Although these methods work for some cases, they will also face situations which they cannot give the right answer. Usually, flights of this kind may have great problems in origin data. They will probably be discarded by error excluding function, such as case 2 and case 6 in this method. Therefore, it is unnecessary to discuss about those special cases and to find correct answer.

Mathematical proof of this method will not be discussed in detail. The reason is origin data provided by FACET is full of uncertainty. Aircraft didnt need to fly exactly as flight plan. Radar has measurement error, which may significantly make this flight data invalid. Flight plan itself may zigzag a lot, and there are no patterns and no ways to generalize those flight plans through mathematical methods. In general, this algorithm is just experimental. It is just trying to get a rough result. But throughout simulations, it works and gets a quite good result. On the other hand, in real time operations, this shall not be a problem because the next way point has to be known by any flight management system. Therefore, proving this algorithm is also doing something with no meaning.



(a) (b)

(c) (d)

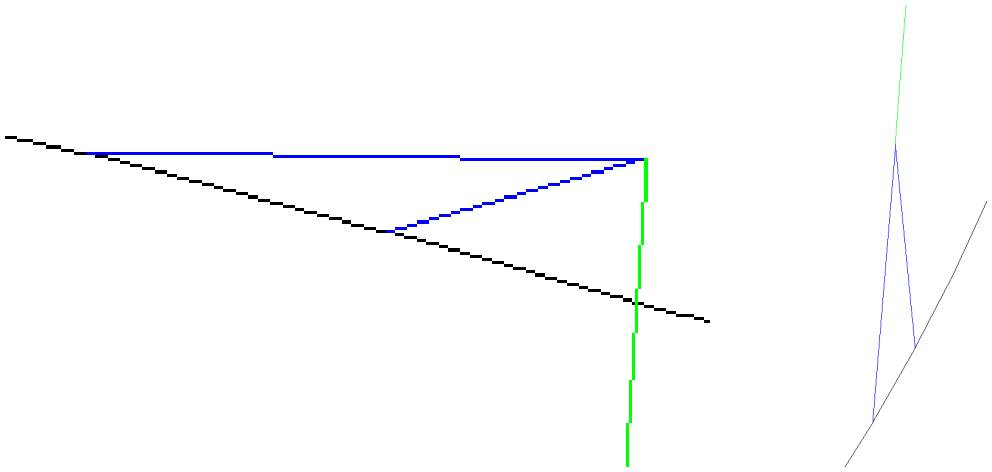
Fig. 4.4. Result of Finding Next Way Point

In Figure 4.4, black lines represent the whole flight plan. Blue lines connect flight position, next way point and previous way point. Green line connects flight position and the final point in flight plan. Green line shows a general direction where the aircraft is going. As we can see, the algorithm works quite well, as long as aircraft is not far away from its designed route.

### 4.2.1 Error Excluding

In some simulations, there exist some irregular conditions just like Figure 4.5. Aircraft is very far away from its designed route. Such irregular condition may be caused by measurement error, or the aircraft itself fly off course. If it caused by measurement error, this flight may not follow any flight model. If this aircraft fly off course, it may not fly to any way point on designed path. No matter what case it is,

using next way point to predict its path is not applicable. Thus, a simple method is used to exclude such cases.



(a) (b)

Fig. 4.5. Error Example in Finding Next Way Point

The idea is calculating distance between aircraft position and the line linking previous way point and next way point. Because unit of position is degree, the limit here is set as 2 degrees. Here it means a distance which equal to 2 degrees in longitude or latitude. It may different in $x$ and $y$ direction because one degree equals different distance in longitude and latitude. However, it is only excluding error here. Such difference will not include extra error.

On the other hand, this error is random. Therefore, one flight may be exclude at time $a$, but it becomes totally no problem at time $b$. This may cause this flight disappear for some time in the database, and reappear after some time. During simulation, the program will suddenly lose all data about this flight during simulation, or a flight may have no past data. The best way is to solve this problem here, not in simulation. An abandon list is introduced as a global value in this part of program. When a flight is excluded by function due to its error data, it will be added to the abandon list. And each time when program tries to find next way point for a flight, it will first check whether this flight is inside abandon list or not. Once this

flight is abandoned, it will be considered throughout the whole process in flight data processing part.

### 4.3 Flight Position Prediction

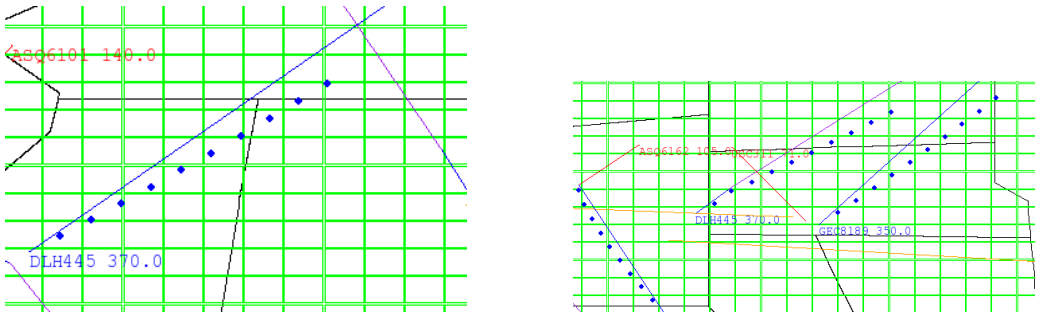
Because this research tries to use a predictive view to solve detour problem, getting positions of aircraft in the future is another important part. Even if there are existing future flight data for me to use, I still choose to predict aircraft position with another method. After carefully checking flight data in FACET, I found some flights faced a severe weather and will try to avoid it at some time. So, if I used existing future flight data as prediction, most severely affected aircraft will not be shown as affected because they avoided that bad weather by themselves.

The flight model used here is very simple. Aircraft are assumed as flying at a constant speed directly to the next way point. Once it fly over next point, the aircraft will be treated as flying directly to the following way point, until it reached the last way point. The turning motion of aircraft is not considered. First of all, because the refresh rate is one minute in simulation, aircraft can make a not very sharp turn in one minute time. Secondly, because aircraft are isolated by each cell, aircraft can fly any route as long as it is inside this cell. Finally, no flight path will take 180 turn. Therefore, any detailed turning model can be neglected. Besides, different aircraft has different aerodynamics models. It will become too complex to consider such a problem in this research. Whats more, the flight model is isolated with origin flight data. It can be replaced with any other accurate model if necessary. In this research, flight model is not the main focus.

Another key point in this flight prediction model is coordinate conversion. Speed of aircraft is knot which is one nautical meter per hour. Coordinate system used in this thesis is longitude and latitude, the unit of which is degree. Therefore, speed will be converted into nautical meter per minute at first. Then, in $x$ direction (longitude),

speed will be timed 44.679, which is median value in ZAU center. And in $y$ direction (latitude), speed will be timed 60.

### 4.3.1 Discussion on Accuracy




(a) (b)

Fig. 4.6. Error Example in Finding Next Way Point

As shown in Figure 4.6, there exists some error in position prediction. This is caused by longitude and latitude coordinates. Distance represented by one degree in longitude is slightly different from each other at different latitude. If the flight path closes to $y$ direction, it will gain more prediction error. To solve this problem, it may include even more complex trajectory prediction algorithm, such as finding great circle between current position and next way point. In this thesis, those errors are just regarded as acceptable.

# 5. WEATHER DATA MANAGEMENT

## 5.1 Pretreatment on CIWS Weather Data

CIWS [4] weather data provids two types of information. One is radar VIL/Precipitation. The other is radar echo top. VIL means vertically integrated liquid [5]. It is radar estimated liquid water volume in a vertical column. This can be used to estimate the thickness of clouds. Thinker clouds are usually more dangerous. Echo top [6] represents the highest altitude which reflectivity value exceeds a specific value. Echo top can be regarded as the lowest part of the thunder storm cloud, but it is not very accurate. In this research, main interest is put on VIL/Precipitation value. Thunder storms can generate strong turbulences outside or under the cloud [7]. Thus, avoidance of such severe weather will depend on a great amount of other information, such as pilot judgment, or on-board weather radar reflectivity map. On-board radar reflectivity map is not available. And pilot judgment includes human factors which are beyond my knowledge. Therefore, it will be very difficult to consider thunder storms as complex 3-D geometry as CIWS provided. In this research, the thunder storms or severe weather is regarded as uniform with respect to altitude. Thus, only reflection intensity data is useful under this assumption.

VIL/Precipitation intensity is represented by polygons with intensity value. Within one value, FACET will return a list of polygons which illustrate the shape of reflectivity map. CIWS in FACET has 7 intensity values, from 0 to 6. In FACET, there is another data type which is CWAM [8]. It estimates severe weather area where aircraft should not fly over. With simple comparison with CIWS data, I found CWAM data is almost analog to CIWS weather data with reflection intensity 4 or higher. Because CIWS have more detail and has a larger area, it is chosen as severe weather data

source. On the other hand, CWAM data are also represented by multiple polygons, it will be very easy to exchange CIWS data with CWAM data.

Now that the geometry of VIL/Precipitation strength which stronger than 4 is analog to CWAM data, the area covered by those polygons can be considered as affected zone. Aircraft are not allowed to fly into those areas, including the cells covered by those polygons. Refresh rate of CIWS data is five minutes. Due to refresh rate of flight data is one minute, weather data have a four-minute data unavailable time. Solution here is to regard weather as unchanged during this four-minute data unavailable time. When a new frame of weather data comes in, it will refresh the whole area. Besides, CIWS contains weather forecast information. Refresh rate of forecast data is also five minutes. Data format of forecast data is the same. Only polygons with VIL/Precipitation stronger than 4 are extract as origin data from FACET. For each minute, it needs to extract three times in CIWS weather: current time, forecast 5 minutes, forecast 10 minutes. For each extraction, three files will be used to save VIL/Precipitation strength 4, 5, and 6. Therefore when simulating, every minute has to read in those 9 files to rebuild the radar reflectivity map.

Weather forecast in CIWS has a pattern that all polygons which representing severe weather has a trend in shrinking. When compared with actual weather movement, the moving direction of clouds in forecast is almost correct, but the size of predicted clouds is usually smaller than actual situation. Usually, 10 or 20 minutes later, the area covered by clouds will become significantly smaller than actual situation. Thus, forecast data is considered as not so accurate. This is one consideration why forecast time is chosen as 10 minutes.

## 5.2 Mapping Weather Data onto Cells

The goal of this section is to get weather information of each single cell, in other words mapping weather data onto each cell. Of course the simplest way is to test intersection of each cell with each polygon. However, the cost of this method is

$O(N \times M)$. Most polygons are not very large and it has to be tested with cells which can't intersect with it at all. The first improvement is to pick out the smallest rectangular which contains the testing polygon. All cells outside that rectangular can't intersect with that polygon. Therefore, the cost is reduced to $O(a \times b \times M)$. $a$ and $b$ is the count of cells in $x$ and $y$ direction of that rectangle. $a \times b$ are usually smaller than 100, while $N$ is more than 1500.

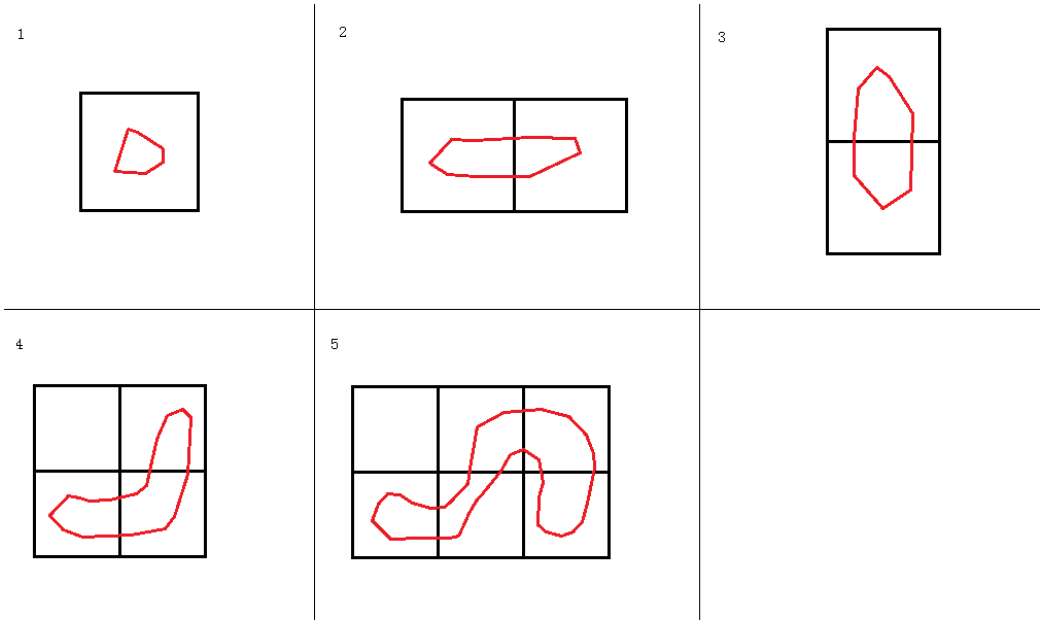



Fig. 5.1. Special cases when using Point in Polygon

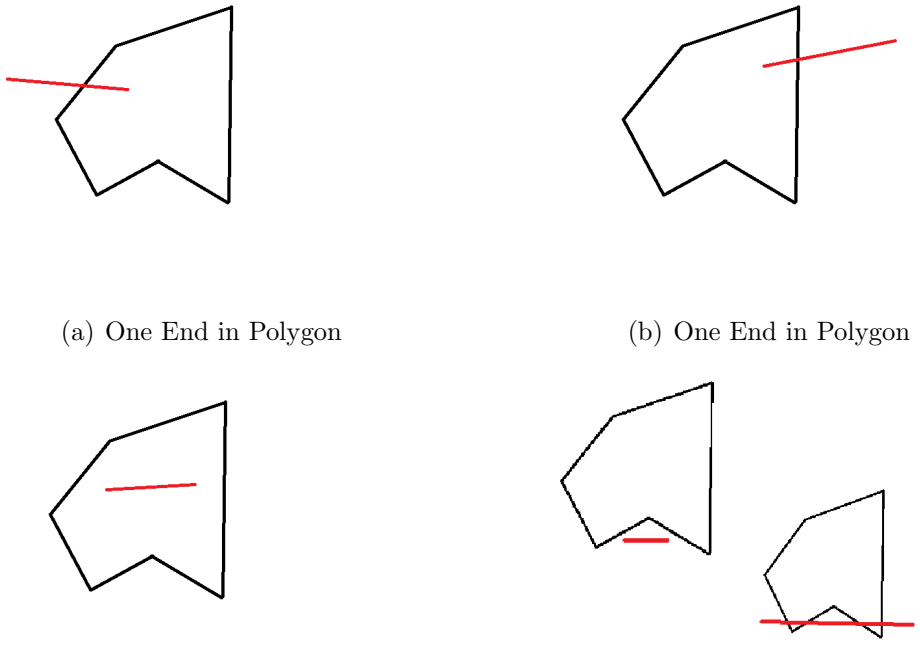
The simplest idea is to check whether each vertex of cells inside the polygon. As long as this vertex is inside polygon, all polygons which are related with this vertex intersect with this polygon. This algorithm looks simple and feasible, but it may face a lot of special cases which cannot be detected as intersected. For example, there may only one single cell contains the polygon. Therefore there is no testing point (case 1 in Figure 5.1). There may exist only two cells contain the polygon, which has no testing point either (case 2, 3 in Figure 5.1). There also may exist one testing point is not inside polygon, but the polygon cross most of cells (case 4, 5 in Figure 5.1). Moreover, we can found the intersection defined here is more than intersection.

Polygon contains or contained by cells are also some kind of intersection. Especially, only case one is cell contains polygon. Other cases are polygon contains cells.

Taking a quick look in those special cases, polygons share an obvious pattern. No matter how irregular that polygon is, it must cross or contains edge of cells. Although in case 2, there is no vertex inside that polygon, that polygon intersects with the only edge shared by two cells. In case 4 and 5, if we look at the only cell which is not covered by polygon, we can found both its vertexes and edges are not contained and crossed by polygon. Now that vertexes method doesn't work, may be edges can get the right solution.

Inspired by algorithm "Point in polygon" [9], this research uses line intersection with polygons to test relationships between rectangle cells and polygon. Point in polygon can return whether testing point is inside the polygon. Figure 5.2 shows four typical situations. As long as one end of that line is inside the polygon, the line must across (or contained by) that polygon. The only problem is Figure 5.2(d) which both end of that line are outside polygon. This line may or may not intersect with that polygon. On the other hand, "Point in polygon" will record how many times each side of polygons cross the y coordinate of testing point on left and right side. By extending this information to both $x$ and $y$ directions, line crosses polygon can be tested by following method.

As shown in Table 5.1 and Figure 5.3, all possible cases are listed. Because all points are outside polygons, all counts are even according to algorithm Points in polygon. It can be shown that in all two false cases, point counts of both points do not changed. On the contrary, if the line crosses the polygon, count on left and right side will be different because some points are moved from left to right. This conclusion is the same when counting the cross in $x$ coordinates of test point. However, in special case 1 in Figure 5.1, there is no lines and points can be tested. Yet, this is the most uncommon case, which is always true, and can be excluded before running the algorithm.



(a) One End in Polygon

(b) One End in Polygon

(c) Both Ends in Polygon

(d) Both Ends outside Polygon

Fig. 5.2. General Cases for Line Crosses Polygon

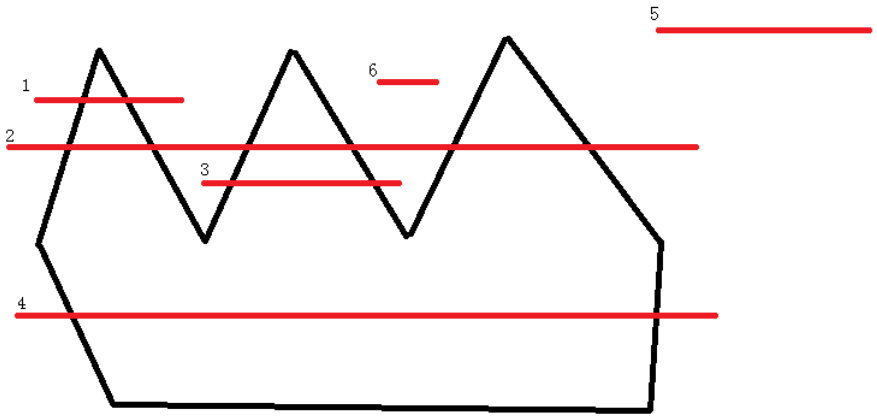



Fig. 5.3. All Cases in Both Ends outside Polygon

With this idea, Point in polygon algorithm is modified a little to test line crosses polygon. In origin code, the testing polygon will be looped from the beginning point

Table 5.1.
All Cases in Both Ends outside Polygon

| Case | Left counts of end 1 | Right counts of end 1 | Left counts of end 2 | Right counts of end 2 | Cross or not |
|---|---|---|---|---|---|
| 1 | 0 | 6 | 2 | 4 | TRUE |
| 2 | 0 | 6 | 6 | 0 | TRUE |
| 3 | 2 | 4 | 4 | 2 | TRUE |
| 4 | 0 | 2 | 2 | 0 | TRUE |
| 5 | 0 | 0 | 0 | 0 | FALSE |
| 6 | 4 | 2 | 4 | 2 | FALSE |

to the last point once. It will counts cross in $y$ direction and only save the value of crosses on left side. In my version, it will counts $x$ and $y$ directions and save cross counts on four sides. For example, suppose $x_i$ and $y_i$ represents the $i$-th vertex of testing polygon, $x$ and $y$ represents the testing point coordinate. In $y$ direction, if one of $y_i$ and $y_{i-1}$ is smaller than y, the other one is larger than $y$, then edge $(i-1, i)$ crosses testing point in $y$ direction. Then if those three points meets the following inequality:

$$x_i + (y - y_i)\frac{x_{i-1} - x_i}{y_{i-1} - y_i} < x \tag{5.1}$$

Then this edge crosses testing point on the left. In $x$ direction, if edge $(i-1, i)$ cross testing point, the inequality is written as:

$$y_i + (x - x_i)\frac{y_{i-1} - y_i}{x_{i-1} - x_i} < y \tag{5.2}$$

If points on edge $(i-1, i)$ meets inequality above, it crosses the testing point at lower side.

Here is a simple demo of how testing algorithm runs. Take $3 \times 3$ cell grid as an example. All vertexes of 9 cells are numbered in sequence. To cover all cells, the

algorithm needs to test every edge inside rectangular 1-4-13-16. In other words, all lines marked with red X in Figure 5.4 has to be tested whether it cross the testing polygon. If any line crosses the polygon, two cells related with this line must intersects with that polygon. To reduce the cost in counting cross of each point, four $4 \times 4$ matrices are used to store the number of crossing on left, right, up and lower side of each vertex. When each vertex is tested crossing, returned value will be saved in those four matrices. For example, point 6 will be tested twice in both $x$ direction and $y$ direction. If each time it comes to time point 6, it has to test crosses in $x$ direction twice, and $y$ direction twice. The testing polygon has to be looped fourth. However, when using four matrices, only one loop of testing polygon is required and testing in crosses can be done at four sides. This will save at least 3/4 in running time.

Using this method, when testing $a \times b$ size of cell grid with one single polygon, the computational cost of this algorithm will be $4(a+1) \times (b+1) \times m$, where m is the number of edge of testing polygon. Considering all polygons in VIL/Precipitation radar map, the cost will be $4(a+1) \times (b+1) \times m \times M$. m here denotes the average number of edge of testing polygon.

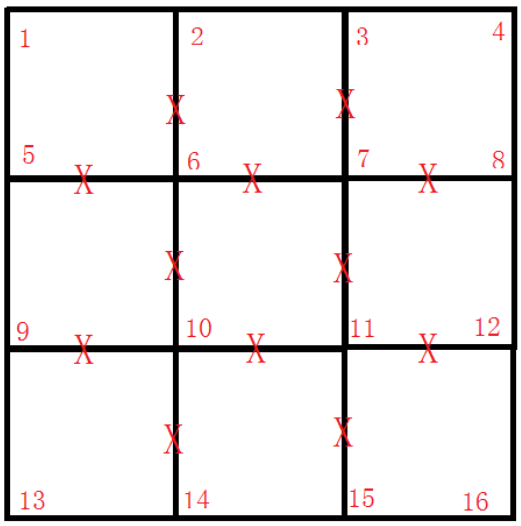



Fig. 5.4. An Example Cell Grid

Till now, the discussion about weather data mapping is only in two dimensional. After getting the 2-D mapping, weather data has to be expanded to all levels in airspace. Because severe weather effect is considered as uniform in altitude, this step will be quite simple. Each affected cell found in previous algorithm has a corresponding cell in each level. They have the same boundary except at different altitude. So, by adding the number of cells in one level can raise that cell to a higher level. Repeat this until it reaches the highest level in airspace.

Besides, weather data maintains unchanged in every 5 minutes. Thus, when each new frame of weather data comes in, the program will map those data onto cells. In the following 4 minutes, it will continue using this frame of weather data and importing them into database. This is also the same when dealing with forecast weather data.

## 5.3 Import Weather Data into Database

After mapping, weather data will be imported into Weather and Flight data table. Flight forecast information calculated in chapter 4 will be combined into this table at the same time.

According to some observation on flight and weather data, I found it is not efficient to build rows for each cell in weather and flight data table. First of all, building a row for each cell will cost a long time. Building a row in MySQL may need to rearrange the whole table. In MySQL, fetching data with primary key almost has no time cost. Updating one row is much faster than creating a new row. Creating a new row will cost a great amount of time if the table contains great amount of entries. On the platform used in this research, at most 2000 rows can be created in database per second. And this speed is boosted by buffered function. If all cells are built, it will cost more than 1000 seconds to finish this job. It will cost a long time and it does not contain any useful information at all. Secondly, most airspace is not affected by weather. Even we prebuild each table before we started simulation, most entries in

Weather and Flight data table will be empty or in a list of NULL. Besides, when debugging, it will be more convenient if each row returned by database contains at least weather information or flight data. However, if this table contains all cells, most rows returned will not contain any useful information. From the point of developing and debugging, each row in Weather and Flight data table will be created when necessary. If there is not flights or weather related with that cell, this id will not exist in Weather and Flight data table. If the program found the id of one cell already exists in data table, it will just update data in that row.

Time constraint in importing data will be very important in this section. If importing work cannot be done within one minute, the idea of using database will be pointless. In actual simulation, importing one minute of weather and flight information costs more than half minute, but a little bit smaller than one minute. However, this is achieved by creating all necessary rows. If program uses a prebuilt table with all cells inside, and only updating data, time used can be greatly reduced. On the other hand, this simulation is run on a normal computer platform. The performance has potential to be improved by using a more efficient computer platform. Therefore, the time constraint can be regarded as satisfied.

Importing flight data is done by fetching flight data at different time and getting the list of forecast cell id. Then according to each cell id in that list, the program will know at what time what forecast this flight is at what cell. Updating in that Weather and Flight data table is almost the same as importing weather data. The only difference is flight data updates F$f$_result, F$f$_fligth_name, and F$f$_occupied. F$f$_result always equals to F$f$_capacity  F$f$_occupied.

Because this table does not contain all cell id, the data logic in this table will be very important when making decision in detour algorithm. There are two commonly used conditions in this research. First is whether aircraft in this cell requires detour. Second is whether this cell can hold one more aircraft.

In Weather and Flight data table, capacity is set as default 2. When severe weather affects this cell, capacity equals 0. F$f$_result data is updated after flight data

is imported. Therefore, if this cell is only affected by weather and not visited by any aircraft, F$f$_result will still remain NULL.

With the assumption above, cells which meet the first condition can be checked by finding F$f$_result which is negative. Because in this condition, it must have aircraft in this cell, thus F$f$_result value must be updated. Second condition is quite complicated. Because it has to check any cell in main focus area, cells which are not created in Weather and Flight data table have to be considered. If this cell is not contained in Weather and Flight data table, it means no weather and no aircraft interact with this cell. In other words, it is clear and available for aircraft to fly over. If this cell can be found in table, and F$f$_result is not NULL. With simply check F$f$_result, availability of this cell can be known. If this cell can be found in table but F$f$_result is NULL. This means no aircraft interacts with this cell. But the weather condition in this cell cannot be determined by F$f$_result. F$f$_capacity must be checked to know weather condition in this cell.

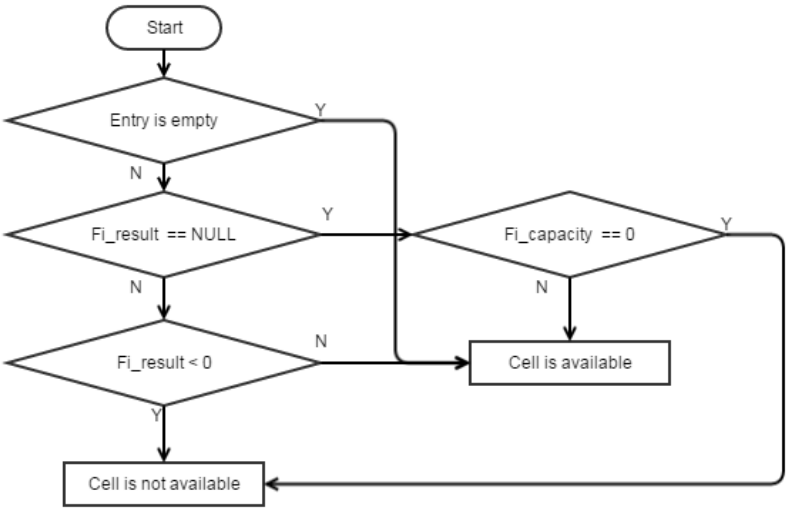



Fig. 5.5. Logic in Weather and Flight Table

In the discussion in chapter 6, logic availability or clear has the same meaning as shown in Figure 5.5. With this logic, all available cells are ensured severe weather free and collision free for extra aircraft to fly into.

## 5.4 Discussion on Result

Figures in Appendix A shows several results of mapping weather data onto cells. The polygons in different color are VIL/Precipitation radar map polygons. Different color means different reflectivity strength. Smaller polygons usually have a higher strength. Black dots at the center of each cell means this cell is detected and regarded as affected with severe weather. In database view, the capacity of this cell will be set 0. From those results, it can be shown that this algorithm can deal with special situations listed in Figure 5.1. The only disadvantage of this result is affected cells are quite larger than radar reflection map. This is can be improved if cell size is chosen smaller.

# 6. DETOUR ALGORITHM

## 6.1 Common Detour Method

This section discusses about common detouring methods which are used by aircraft. In most cases, aircraft will use ascending, descending, or choosing another route to avoid severe weather. In some special cases, aircraft can speed up and quickly pass through severe weather if the cloud is narrow just as what UPS 988 does in Appendix B. If severe weather affects so large an area that landing airport is full or not capable for landing, aircraft can choose to land at an alternative airport. Those three methods may not contain all possible detour methods, but they must be the most commonly used methods in air traffic system.

For three possible detour methods, this research mainly focuses on picking another route in first method. For weather information are considered uniform in altitude, ascending and descending is not suitable for this model. Crossing severe weather data may need to follow a complex flight regulation and depends on actual situation on the aircraft. For example, UPS 988 is obviously a cargo aircraft. Regulations on cargo aircraft are definitely different with passenger aircraft. On the other hand, FACET data doesn't contain such information and the only solution is to treat every aircraft as the same. For alternative airport landing, it is another story if this aircraft can't reach its destination by any detouring method. Therefore, the detouring in first methods is most important and usually first considered. This is why this research mainly focuses on this.

The best detour path should be following the contour of severe weather polygon and goes straight back to its planned as soon as possible. Although with a predictive view, if aircraft knows it will be affected by weather several minutes later, it can leave its origin path to take a shorter detour route instead of following the contour

of polygon. However, weather system is changing and weather forecast is not always accurate. It is possible that cloud goes in another direction and this flight does not need to detour any more. Besides, once this aircraft leave its planned path, it is flying into airspace which belongs to other flights. The longer it stays off course, the more it will affect other flights. Leaving its designed path is not always acceptable. This is a difficult trade off in detour method. In this research, detour method will not consider leaving origin path at early time, and it will try to return to its path as soon as possible. When leaving it origin path, aircraft will avoid conflicts with other aircraft, and try to follow the contour of polygons.

## 6.2 Classification on Weather Affection

With the help of database platform, finding affected flight becomes very fast and simple. It only need to fetch F$f$_result values which are less than 0 in Weather and Flight tables at time $T$. For one single table, it needs to fetch eleven times because each table contain one current result and 10 forecast results. The program can get 11 lists of aircraft. In those lists, aircraft may have faced severe weather or it flies into a cell which is occupied by other aircraft. One important thing is that one aircraft may appear in those lists several times. However, only one detour path needs to be applied onto this flight. Therefore, it needs a function to get unrepeated flights throughout those 11 lists.

The next step is to fetch cell id list from Flight data table with the information of flight name and affected time. Suppose $id_i$ is the $i$-th item in this id list. It means this flight fly over this cell at time $T$ forecast $i$. Then with $id_i$ and $i$, the program will fetch F$f$_result again in Weather and Flight data table to get a list of F$f$_result. The data can be illustrated in the following format.

In this list, affect status is defined as: if F$f$_result is larger and equal than zero, it is not affected. If F$f$_result is smaller than zero, it is affected. At any time, each

| | Time 0 | Forecast 1 | Forecast 2 | ... | Forecast 8 | Forecast 9 | Forecast 10 |
|---|---|---|---|---|---|---|---|
| Flight name | Affect status Cell id | Affect status Cell id | Affect status Cell id | ... ... | Affect status Cell id | Affect status Cell id | Affect status Cell id |

affected flight can get this single list of affect status information. With this list of information, each affected flights can be classified into the three cases:

1. Currently affected flights
2. Affected but goes out of severe weather in the forecast time
3. Affected and keep affected until the end of forecast time

The first case has the following feature in status list.

| | Time 0 | Forecast 1 | Forecast 2 | ... | Forecast 9 | Forecast 10 |
|---|---|---|---|---|---|---|
| Flight name | affected | xxx | xxx | ... | xxx | xxx |

The current time when forecast time $f$ is 0, this flight is affected by severe weather. If any flight is classified into this case, it will not be considered by detour algorithm because it is supposed that each aircraft shall be in clear airspace when detour starts. If we consider this situation, it may happen that this aircraft is current inside a large cloud. This cloud may be safe judged by pilot on situation, or the aircraft is flying beneath or above the cloud. However, from the program view, it will see all cells are affected not only at the current position, but also all the other cells around it. The aircraft may have no place to go at the first step. Any detour algorithm cannot start with this initial condition. From another view, this aircraft must be at a clear airspace several minutes ago then it flew into affected zone. Therefore, this flight shall

be considered and detoured several minutes ago. Discussing this case at this moment is pointless.

The second case has the following feature in status list.

| | Time 0 | ... | Forecast $i$ | ... | Forecast 8 | Forecast 9 | Forecast 10 |
|---|---|---|---|---|---|---|---|
| Flight name | Not affected | ... | affected | ... | xxx | xxx | Not affected |

In this case, time 0 must be clear just to avoid the first case. At any forecast time $i$, the aircraft is affected by severe weather, which is the least condition to be returned by check affected function. Besides these conditions, this flight must be not affected at forecast 10 or earlier. The advantage of this case is that it gives a clear cell, which is not affected. And this cell is on the route of the flight path. This cell is a very good to be chosen as detour destination. If forecast 8, 9, 10 are all clear, this flight can have at least three alternative destinations. The detour algorithm will try to return it original path in forecast time.

The last case has the following feature in status list

| | Time 0 | ... | Forecast i | ... | Forecast 8 | Forecast 9 | Forecast 10 |
|---|---|---|---|---|---|---|---|
| Flight name | Not affected | ... | affected | ... | xxx | xxx | affected |

In this case, time 0 is the same as second case. Forecast $i$ is affected to meet the condition of check affected function. The only difference is even at forecast time 10, it is still affected by the severe weather. This means, even the last forecast position of this flight is not suitable to be chosen as detour destination. Detour destination has to be chosen through other methods. This is the worst case because no points on predicted path can be chosen as target. Aircraft cannot return to its original path in forecast time.

After finishing deciding which case this flight belongs to, the program will call corresponding solution function to find detour path. The main idea of detour algorithm is the same for both case 2 and case 3.

### 6.2.1 Result on affected flight detection

Because the origin flight data doesnt provide delayed information, affected flight detection has to be tested manually. A program is written to show flights inside main focus area and mapped weather data together. This program has options to show all aircraft or only affected aircraft. Result of this section is shown in Appendix B. Blue lines are predicted flight path of flights which are not affected by weather at time $T$ forecast $i$. Red lines means this flight path is affected by weather at time $T$ forecast $i$. Therefore, flights will turn red only when it flies into affected cell. As we can see, affect detection function can pick out flights which fly over affected weather correctly, and it can tell exactly when and where this aircraft will be affected by severe weather.

Time constraint in finding affected data is very satisfied. It is almost finished immediately. Most time consumed in this checking program is the visualization part.

## 6.3 Detour

### 6.3.1 Finding Detour Destination for Case 2

In this case, any point which is clear after the last affected position can be chosen as destination. However, as weather is changing time by time, it may happen that cell $i$ is clear at forecast 4, but it is affected since forecast 5. Although each detour algorithm will try to reach its destination as soon as possible, it is obvious that detour will cost longer time to reach the destination. If such happens, this destination will become unavailable and aircraft will circle around cloud trying to reach it. To avoid choosing an unavailable point as a target, each destination cell will be checked before it is selected. As long as it becomes affected in forecast time, the program will try

next cell in the status table. Logically, if cell in forecast 10 is clear, it must be one feasible destination. But if forecast 10 is chosen as destination, in most cases this point will not be reached within 10 minutes.

### 6.3.2 Finding Detour Destination for Case 3

In this case, the program will try to use next way point as destination point. Now that all cells on forecasted path are not available, choosing way point as destination will be the simplest way, though this point may not be reached during forecasted time. On the other hand, aircraft shall not deviate from its planned path for a very long time. Thus, the closest next way point will be the best choice.

However, though next way points are found and imported into Flight Data table, at different forecast time, the aircraft may fly to way point 1 as well as way point 2. It has such information when predicting flight path, but such information is lost when importing into database. Thus, a function is used here to find which next way point this flight is flying to. The idea is using dot product. If two vectors are pointing at the same direction, the dot product of those two vectors must be greater than 0. On vector is built by linking current position to next position. The other vector is built by linking current position to a possible way point. Besides, because error data have be excluded by programs in section 4.2.1, data in database cannot cause any unexpected cases. However, it rarely happens a case that aircraft fly over two way points in 10 minutes, which lead to no possible way point to be chosen as destination. In this case, detour function will be terminated and return failed in finding detour path.

### 6.3.3 Main Detour Algorithm

Because each step in detouring algorithm will be doing the same thing, the program will be concise and easy to understand if written in recursive function. Whats most important thing is, forecast time is limited to 10 in this research. Therefore, re-

cursive function will be called at most 10 times. Even if forecast time becomes longer, it is also a limited number. Thus, recursive function is the best way to achieve this target.

The recursive function requires the following parameters to work:

time, recursive deep, current position cell id, target cell id, current position coordinates, altitude, speed, heading direction, detour start direction (DSD), heading changing direction (HCD), SQL handler.

Usually, when people facing an obstacle, there are two options to avoid it, one is try left way, the other is try right way. When case 2 and case 3 solution functions trying to call recursive to find detour path, it will call twice to try both directions. And solution function will decide which path is a better one.

Following is a pseudo-code which shows basic structure of recursive function. In this code, SQL handler is omitted, and it only shows the simplest logic sequence.

```
def Recursive(time,deep,id,targetid,x,y,
              alt,speed,heading,DSD,HCD):
  if deep >= 10:
    return True
  searchlist = GetAlternativeAngle(x,t,targetid,
                                   heading,DSD,HCD)
  for direcion in searchlist:
    [nextx,nexty,nextdirection] = AircraftMotion(x,y,speed,
                                                 direction)
    nextid = Getid(nextx,nexty,alt)
    if nextid == targetid:
      return True

    if GetAvailabile(nexid):
      result = Recursive(time,deep+1,nextid,targetid,
                         nextx,nexty,alt,speed,nextdirection,
```

```
                                    -HCD, HCD)
        if result:
            return True

    return False
```

### Alternative Detour Direction List

As shown in pseudo-code, each step will get an alternative direction list to search for available cell. This direction list can greatly change the performance of Recursive function. This section will discuss how this direction list is generated.

At the first time calling that recursive function, detouring function will be given two HCD values. -1 means aircraft will fly to left side at first step, and biggest turning direction will appear on the left side. +1 means the opposite direction. A function will be used to generate this alternative direction list. This function requires parameters such as current position, target cell id, heading direction, DSD and HCD.

It will always be the best direction if the aircraft can fly directly to the target position. Thats why this function requires current position and target cell id. A best direction will be calculated and appended at the first position of alternative direction list if the angle between heading direction and best direction is in $[-135, 135]$ degree interval. It is supposed aircraft can make a 135 degree turn in one minute. Of course, this value can be changed if necessary. Then according to DSD and HCD, up to four alternative detour directions will be appended into list. A simple example is shown in Figure 6.1. DSD can have two values, +1 and -1. It is the start angle which will be first appended into list. In most cases it equals the negative value of HCD. This means that although this aircraft will try to turn left, it will first try turn right a little bit to see whether this is allowed or not. Only at the first call of recursive function, DSD is the same value as HCD, because the first step is required to turn left at the

very beginning. If DSD is +1 and HCD is -1 as shown in Figure 6.1, the appended list is $[45, 0, -45, -90]$.

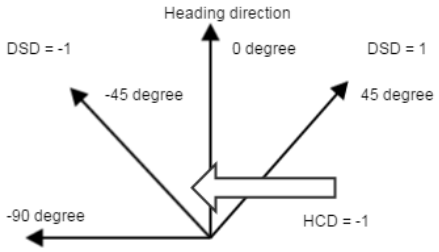



Fig. 6.1. Example of Alternative Direction List

Therefore, with this alternative detour direction list, the aircraft will first try to fly directly to destination, and then change direction as it is required. If HCD is -1, it will try turning right a little bit. If this fails, it will try tuning 90 degree to the left. If aircraft first choose left turn to avoid severe weather, in most case, weather polygon will appear on right hand side of aircraft. Thus, a little right turn will keep the aircraft follow the boundary of that polygon. The largest turning angle is at left side, so if severe weather still exists on its path, it can turn left to avoid it. On the other hand, if only keep following the boundary of polygons, aircraft will keep circling around that polygon. Therefore, best direction must have a higher priority. This will keep aircraft return to its path if possible.

**Repeating Recursive**

The recursive function will check whether it exceeds the deepest step at beginning. If it exceeds its deepest step, which is 10 in this research, it will return true as regarding detour gets a satisfied result.

After getting alternative direction list, the recursive function will try all possible direction provided in that list. For each direction, it will call aircraft motion model to fly in that direction and try to get next aircraft position and cell id. Aircraft

motion model used in this section is almost the same as section 4.3. Aircraft are supposed to fly at a constant speed. And altitude change is not considered because weather is considered as uniform in altitude. This aircraft model is not very accurate, and it can be replaced by a more accurate model if necessary. If the target cell id equals the returned cell id, detour is considered as finished and function will return true with necessary detour information. Detour information is not included in pseudo-code. In other situations, it will check whether cell returned by aircraft motion model is available to reach using logic in Figure 5.5. If it is available, then call recursive function to go next step. If all directions in alternative direction list are not available, it will return false and ask previous step to try next direction.

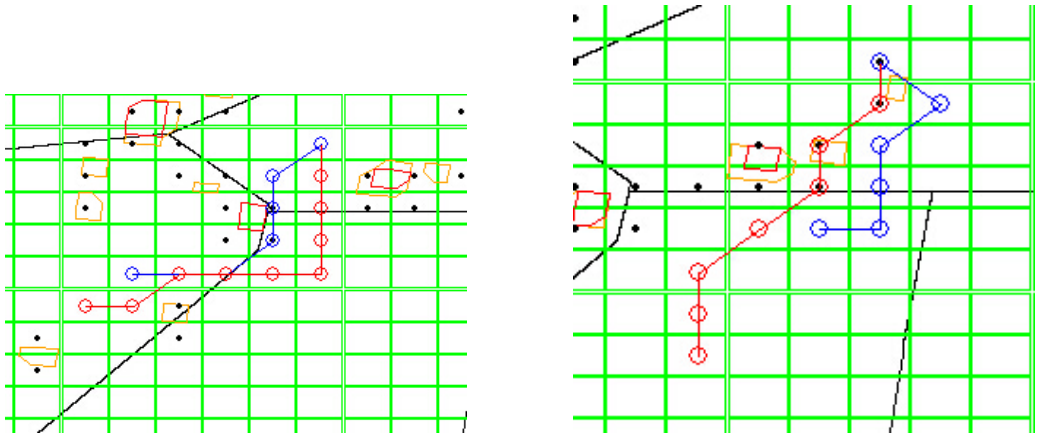
### Picking out a Better Path

Because at the very beginning, when case 1 and case 2 call recursive function, it will give two DSD to start detouring. Therefore, it is necessary to pick out one detour result which is better. The rule here is to check the last position each detour can reach. If one detour reaches a closer position to the destination, it is defined better. It is obvious that if aircraft reaches its target cell in forecast time, this will always be the best path. If one direction fails to get a feasible detour path, the other direction is always a better path.

## 6.4 Discussion on the Detour Algorithm

This section will show some results of the detour algorithm. Because pictures can only show a static view, weather information shown in all figures in this section is the last minute of forecast. Flight path may come across a cell which used to be clear but is shown as affected at last minute. Red hollow circle means origin flight path of this flight. It is obviously affected by severe weather. Red hollow circle is linked by red lines to show continuity. Blue hollow circles as well as blue lines are detour path. Blue and red circles are always located at the center of that cell. In those figures, the

origin CIWS weather data are also displayed for comparison. CIWS weather data are represented by polygons with different color. Affected cells are marked with black dot at the center of that cell.



(a) (b)

Fig. 6.2. Example 1 of Detour Algorithm

Figure 6.2(a) and 6.2(b) shows two typical optimal brought by alternative direction list. One of the best ways taking detour route is to fly along the contour of weather polygons. This is also the reason why alternative direction list is generated as Figure 6.1. This method works best if the weather contour is convex. And we can see that the best direction drive aircraft back to its path as soon as possible and avoided circling around affected cells. Another advantage brought by best direction is aircraft can take shortcuts as shown in Figure 6.2(a). Most flight path follows way points which are not always a straight line. If there is severe weather on that flight path, best direction can make aircraft pick a straight path if weather is clear on that path.

Sometime, a concave weather contour will lead to a dead end. This is avoided by using recursive function. As part of its advantage, if aircraft flew into a dead end, it will return to several steps ago and pick another direction to avoid goes into this dead end.

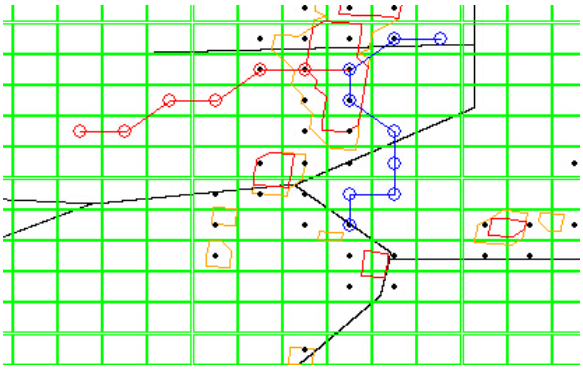


Fig. 6.3. Example 2 of Detour Algorithm

This detour algorithm still has some space to be improved, as shown in Figure 6.3. The first backward turn is caused by the movement of clouds. Clouds moved right during the simulation. Therefore, the detour algorithm went backward a step to avoid it. However, if contour of that cloud has a sharp turn, the algorithm will follow the contour and makes a 90 degree turn at the same time. It is obvious that one 45 degree turn is much better than one straight step and a 90 degree turn. This is one weakness of this algorithm.

Although part results of this algorithm are not good enough, it at least gives a boundary which a detour may have. In some existing algorithms, it needs a largest rectangle as a stage to contain all polygons as obstacles. It can take the shortest path between starting and ending point when avoiding obstacles. With the result of this algorithm, the largest rectangle can be found and makes that algorithm feasible. On the other hand, in civil aircraft operations, pilots have a very high authority to take what path when facing severe weather. So, the purpose of this detour algorithm is trying to provide some suggestions which may be helpful when making detour decisions.

# 7. SUMMARY

In this research, an experimental algorithm is developed to find next way point for flight data which lost such information. Error excluding and fault tolerance is a very important issue in this part. The origin data may contain any special case which may cause a list of error in simulation. Algorithm in this part finishes this task successfully. When processing with weather data, a new algorithm is designed based on a reliable Point in polygon algorithm. This algorithm is proved to be feasible to any special cases. Finally, a completely new detour algorithm is introduced to give some basic suggestions when making detour decisions.

Limited with time and knowledge, this research can be improved in many ways. If we consider the Earth as a sphere, rectangle division is obviously not good enough. In flight data, it is best to use real time data source, or using a data source which is more accurate. Therefore, error excluding can be skipped and more information such aircraft type will be available. With detailed aircraft type, a more accurate flight model can be applied. Flight predication can be more accurate. Weather data system and detour algorithm can also be improved to get an ideal result.

However, one advantage of this research is it separates each small system. Database works as a platform, its responsibility is accepting and providing information to other application and systems. With this structure, flight prediction model is not related with flight data systems. It can be replaced with any other better flight model by simply using another function. Weather data source and detour algorithm is also the same as flight model. They only depend on data provided by database. Another advantage of this research is trying to find potentials of database. In most cases, database is only considered as storage of large data. In this research, database is more than just saving flight and weather data. It is given tasks to combining those

data and find affected flights. This idea is proved to be feasible and can be improved in many ways.

Time constraint is considered but not strictly complied. Because this research is a testing possibility of a new structure, time constraint is not the most important topic. On the other hand, time consumed by building Weather and Flight data table greatly depends on the computer platform and the size of main focus area. In this research, computer platform has a great space to be improved. SSD hard drive can read and write much faster than HDD drive. On the other hand, because this research views the whole system from the view of cells, a large area can be divided into many small area which managed by multiple computers while keeping the size of cells unchanged. Decreasing cell number can significantly reduce time consumption in building tables.

In general, using database as a platform to integrate weather and flight data to predict severe weather affection is reliable and promising. Using database data to predict flight detour is proved to be feasible and can be improved in many ways.

# REFERENCES

# REFERENCES

[1] B. Sridhar, S. R. Grabbe, and A. Mukherjee, "Modeling and optimization in traffic flow management," *Proceedings of the IEEE*, vol. 96, no. 12, pp. 2060–2080, 2008.

[2] D. Bertsimas and S. S. Patterson, "The air traffic flow management problem with enroute capacities," *Operations research*, vol. 46, no. 3, pp. 406–422, 1998.

[3] D. Sun and A. M. Bayen, "Multicommodity eulerian-lagrangian large-capacity cell transmission model for en route traffic," *Journal of guidance, control, and dynamics*, vol. 31, no. 3, pp. 616–628, 2008.

[4] "Mit lincoln laboratory: Faa weather systems: Corridor integrated weather system," https://www.ll.mit.edu/mission/aviation/faawxsystems/ciws.html.

[5] "Vertically integrated liquid vil," http://www.wdtb.noaa.gov/courses/MRMS/ProductGuide/SevereWeather/vertically-integrated-liquid.php.

[6] "Doppler radar imagery: Echo tops," http://www.lakeeriewx.com/CaseStudies/EchoTops/EchoTops.html.

[7] "Ac00-24c thunderstorm," Feb. 2013, http://www.faa.gov/documentlibrary/media/advisory_circular/ac%2000-24c.pdf.

[8] "Mit lincoln laboratory: Weatheratm integration: Pilot decision making on weather avoidance," https://www.ll.mit.edu/mission/aviation/wxatmintegration/pilotdecision.html.

[9] D. R. Finley, "Point-in-polygon algorithm," http://alienryderflex.com/polygon/.

# APPENDICES

# A. MAPPING WEATHER DATA RESULT

Figures in this appendix also show the result of cell division. Except the last figure in this appendix, they represent the whole main focus area of ZAU center defined in this thesis. Some of weather polygons which are not entirely inside ZAU center are also considered.

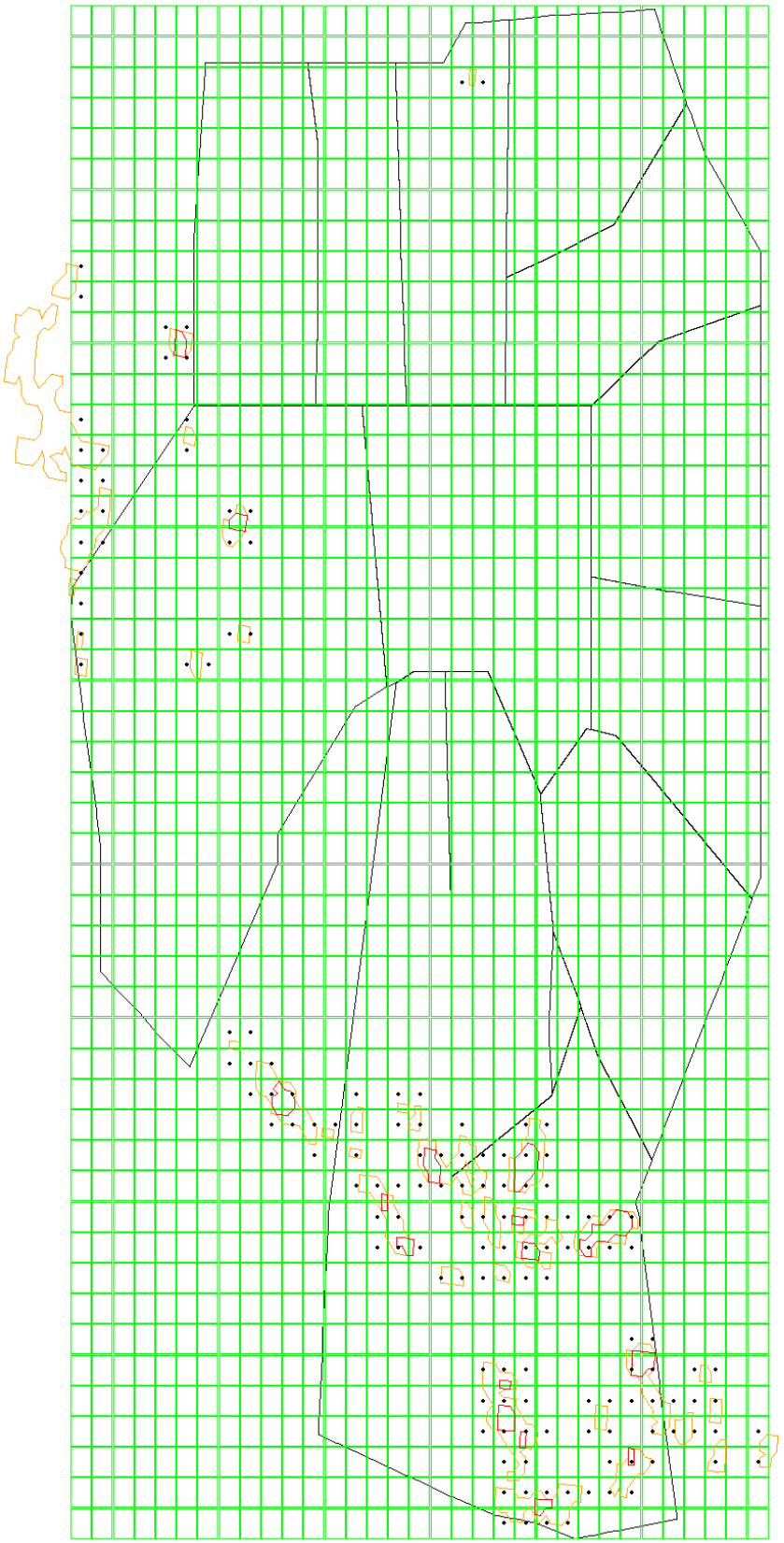


Fig. A.1. Weather Mapping at Time 60 forecast 0

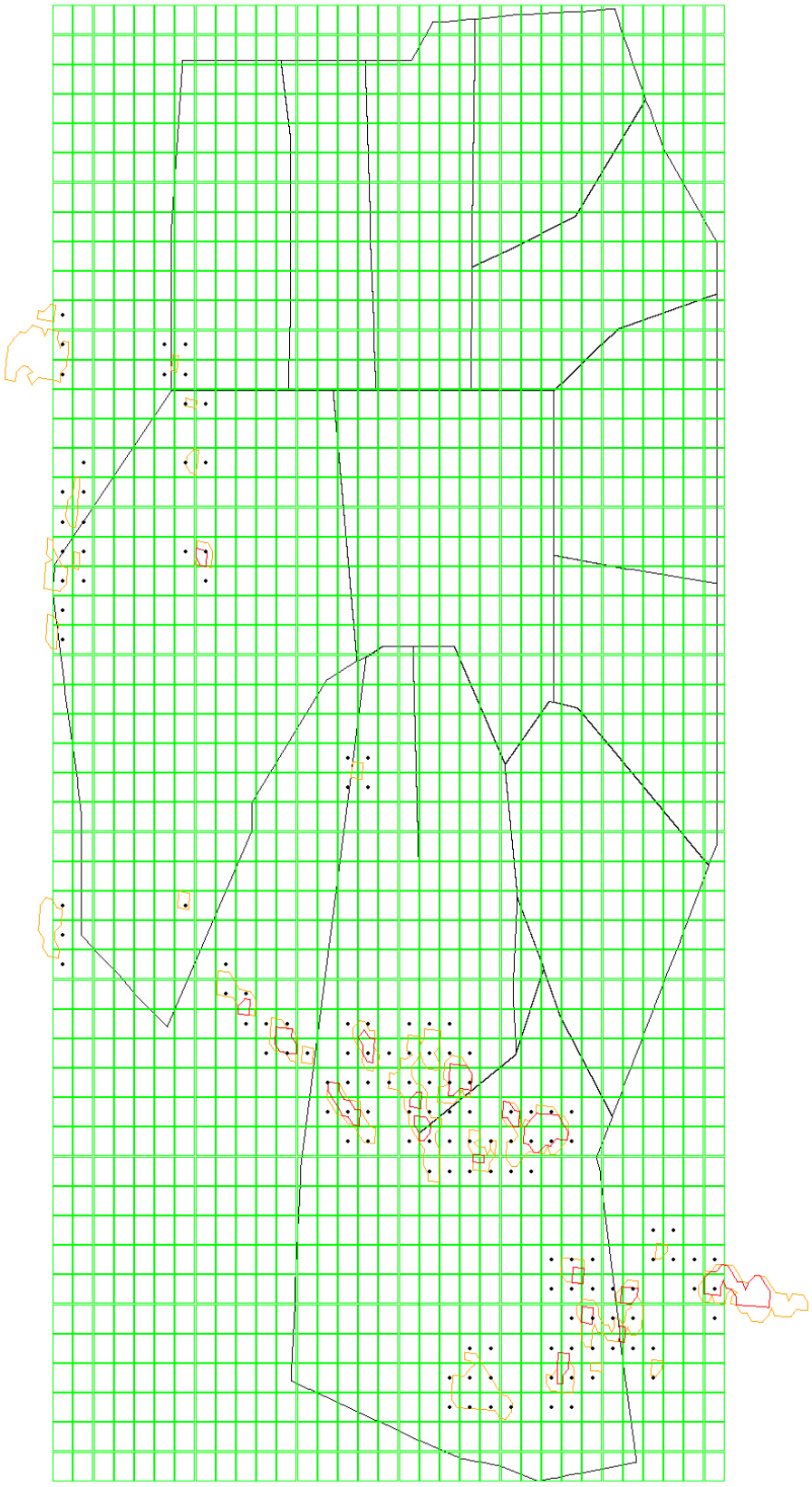


Fig. A.2. Weather Mapping at Time 80 forecast 0

Fig. A.3. Weather Mapping at Time 300 forecast 0

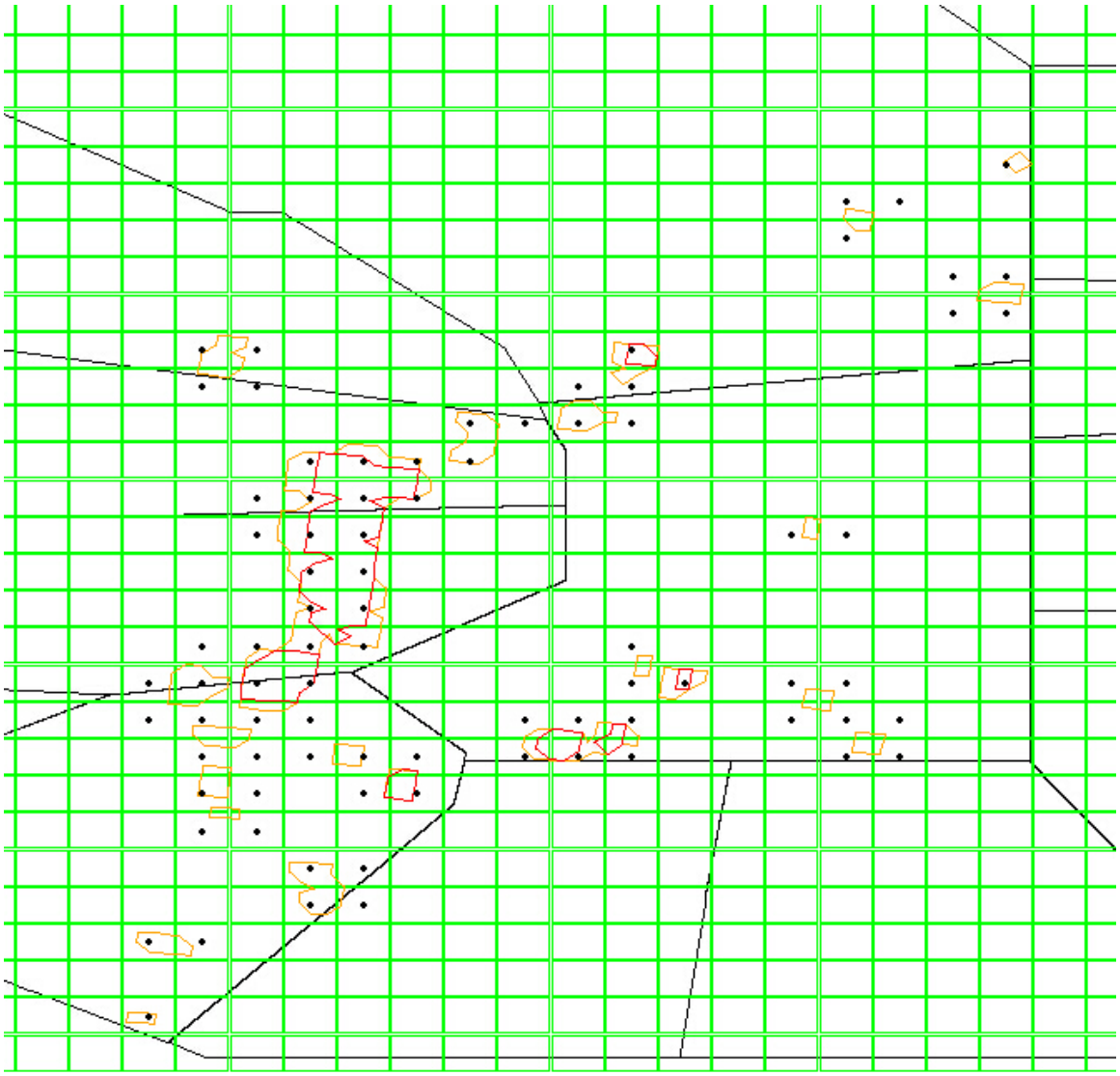


Fig. A.4. Weather Mapping at Time 280 forecast 0

# B. SPECIAL CASE UPS988

This appendix shows part of UPS 988 flight path which is a typical special case.

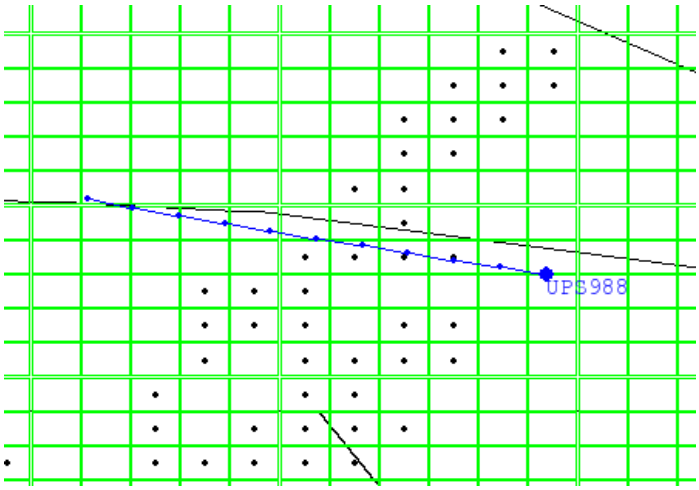



Fig. B.1. UPS988 at time 60 forecast 0

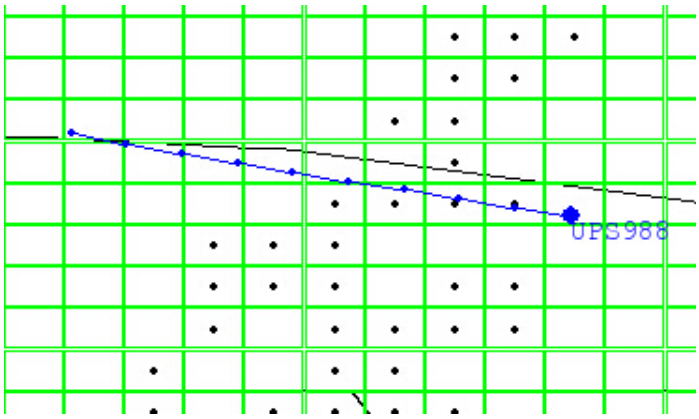



Fig. B.2. UPS988 at time 60 forecast 1

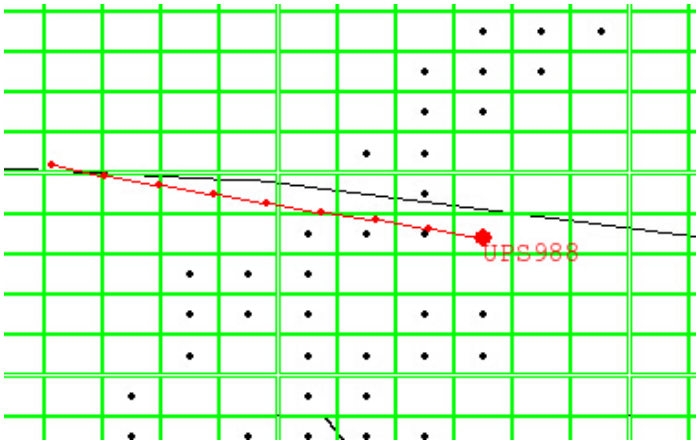



Fig. B.3. UPS988 at time 60 forecast 2, affected by severe weather

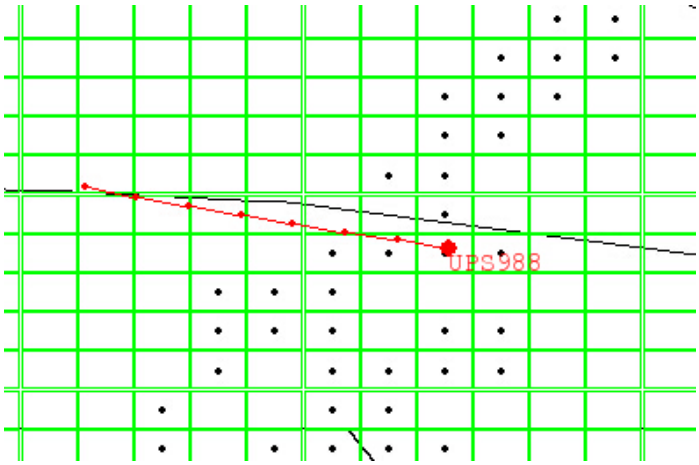



Fig. B.4. UPS988 at time 60 forecast 3, affected by severe weather

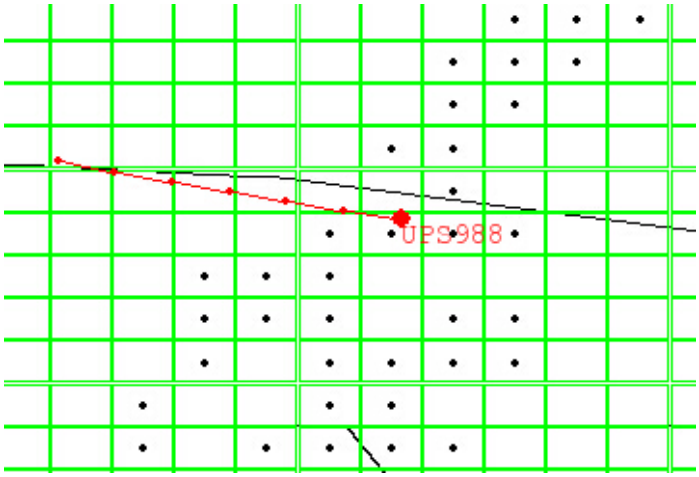



Fig. B.5. UPS988 at time 60 forecast 4, affected by severe weather

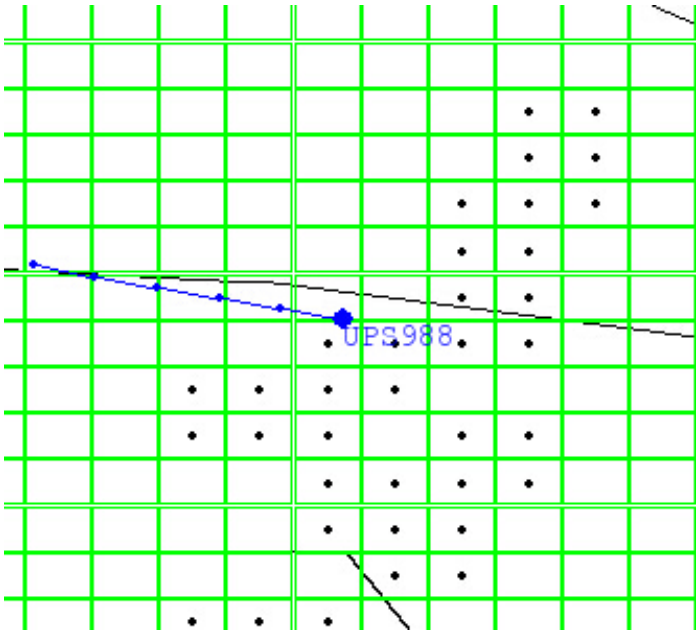



Fig. B.6. UPS988 at time 60 forecast 5