

SIMULATION OF AIR TRAFFIC CONTROL AT A VFR AIRPORT USING EXTEND+MFG+OPEMCSS5

John R. Clymer, California State University Fullerton, Fullerton, CA 92834

Victor Harrison, California State University Fullerton, Fullerton, CA 92834

Abstract

A Complex Adaptive System (CAS) is a network of communicating, intelligent agents where each agent adapts its behavior in order to collaborate with other agents to achieve overall system goals. A VFR airport is modeled where each aircraft is an intelligent agent that communicates with other aircraft in order to decide when to enter the approach corridor. The control problem to be solved in this general aviation VFR airport is the variation in aircraft speed, which results in faster aircraft overtaking slower aircraft in the approach corridor. A graphical simulation library called Operational Evaluation Modeling for Context-Sensitive Systems (OpEMCSS) has been developed to simulate complex systems, including CAS. This simulation library includes a Classifier Event Action block that is a forward chaining, expert system controller used in intelligent agent decision-making. The Classifier Event Action block can implement both crisp and fuzzy rules. In one version of the model an equation is used to predict how long an aircraft is required to wait to enter the approach corridor in order to achieve the proper time and distance separation at final approach. Comparison of the required and actual performance is used to guide rule learning in the Classifier Event Action block. This paper discusses the performance of the proposed control strategy and the fuzzy rule-learning problem.

Introduction

A Complex Adaptive System (CAS) is a network of communicating, intelligent agents where each agent adapts its behavior in order to collaborate with other agents to achieve overall

system goals. Further, the overall system often exhibits emergent behavior that cannot be achieved by any proper subset of agents alone. CAS includes satellite communications networks, traffic control networks, multi-national corporations, the global economic system, ecological systems, flexible manufacturing systems, and military command and control C4ISR networks [1].

In this paper a VFR airport is modeled to study the affects aircraft agent collaboration. Normally aircraft circle in the queuing area until the approach corridor empties and then they all line up and enter the corridor in single file. This behavior results in long null periods when the runway is not utilized followed by short burst of frenzied activity. Similar behavior is observed as aircraft leave the tie-down area to queue up for take off. Pilots attempt to space their aircraft so as to not overtake the aircraft ahead, but they often come too close to the aircraft ahead, requiring the aircraft to leave the approach corridor and reenter the approach queue. Anytime a faster aircraft passes a slower aircraft in the approach corridor there is the potential for a midair collision. In this paper it is shown that by sharing GPS generated position and speed, aircraft are able to collaborate and space themselves out in the approach corridor so as to not overtake the aircraft ahead.

A graphical simulation library called Operational Evaluation Modeling for Context-Sensitive Systems (OpEMCSS) has been developed to simulate CAS [5-9]. OpEMCSS5 works with EXTEND5+MFG, a powerful yet relatively inexpensive simulation tool (www.ImagineThatInc.com). OpEMCSS implements the Operational Evaluation Modeling (OpEM) graphical Discrete Event Simulation (DES) language. The OpEM DES language, based on interacting concurrent processes, includes

primitives for process flow and concurrent process interactions such as resource contention, process synchronization, and process communication and adaptation [2,5,7,9].

The initial solution to regulate aircraft entering the approach corridor was to use an equation that determined the wait time required for an aircraft in the approach queue to wait in order to assure not overtaking the aircraft ahead. This equation assumes that each aircraft flies the same nominal path through the approach corridor as shown in figure 1 below. Based on actual observation of VFR airport operation this assumption is true within limits, but some deviation from the nominal path does occur. These deviations were simulated in the model based on statistical distribution data obtained from observing airport operation. The equation accounts for the worst-case variation, thus resulting in slightly longer average approach queue wait time than desired.

If an aircraft agent has rule learning ability, is it possible to generate a set of rules that would

perform better than the equation? Since the inputs to these rules are continuous values (Speed, Closing-Velocity, and Distance along the corridor) and the output (aircraft wait time to enter the approach corridor) is also continuous, a set of fuzzy control rules will be used. The Classifier Event Action block is capable of doing fuzzy control, and it has rule-learning (rule induction) capability to discover an optimal set of crisp control rules.

First, a VFR airport simulation program is presented. Second, the fuzzy Classifier Event Action block is described. Third, the performance of the fuzzy airport control is compared with that of the equation. The paper concludes with a summary of the fuzzy rule-generation problem and a discussion of future research.

Simulation of VFR Airport

Figure 1 shows the animation grid for the VFR airport simulation. The aircraft icons move on the grid in proportion to their simulated position and velocity. Watching the animated operation of the

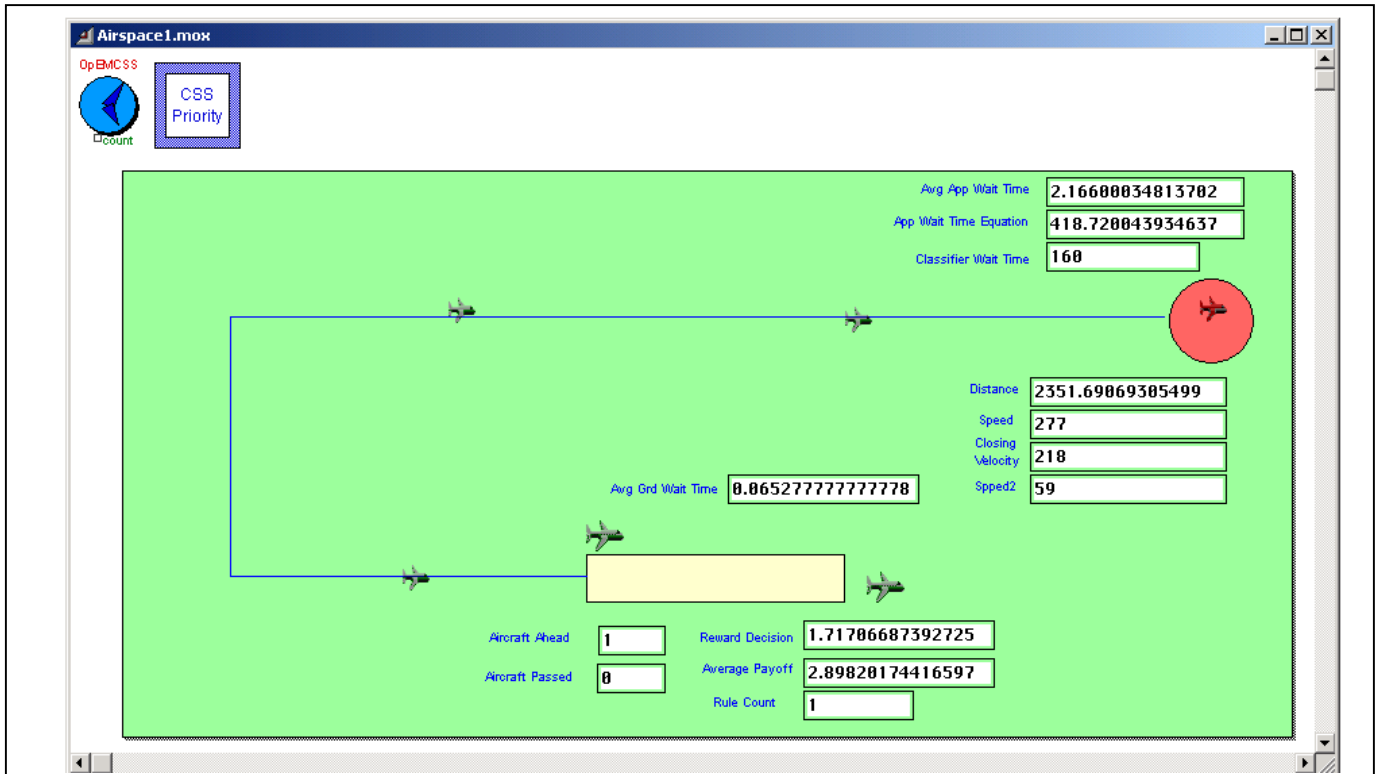


Figure 1. Animation Grid for the VFR Airport Simulation

airport allows us to determine if the rules are working correctly.

The approach aircraft enter the approach queue (circle on the right side of figure) where they wait on a first-in-first-out basis to enter the approach corridor. The runway is shown as a rectangle. The line that connects the approach queue circle to the runway rectangle represents the nominal path through the approach corridor. Ground aircraft enter the takeoff queue at the left side of the runway rectangle as shown by the aircraft icon there. Aircraft land and takeoff by moving from left to right with respect to the runway.

An agent is a CAS component capable of perceiving and acting on its own behalf, and it decides for itself what needs to be done to satisfy its own design objectives [12]. Agent operation is modeled using the OpEM language as a collection of communicating process instances that perform all agent functions as shown in figure 2. Each agent decision-making function is implemented using a

classifier system block. Holland [10] has promoted the idea of a classifier system for many years. In this paper, the OpEMCSS Classifier Event Action block, that can use both crisp and fuzzy rules to make decisions, is applied. This block is comparable to Holland's classifier system [8].

OpEMCSS Directed Graph Model for the VFR Airport Simulation is shown in figure 2. The Begin Event block in the upper left-hand corner initializes model control and data collection parameters and the Split Event Action blocks that follows connect to four concurrent / parallel processes that model aircraft operation. The top process simulates the arrival of aircraft into the approach queue. The second from the top process simulates the arrival of aircraft into the takeoff queue. The next process models the motion of each approach aircraft as it moves through the approach corridor and lands on the runway. The bottom process models the motion of each ground aircraft as it moves onto the runway and takes off.

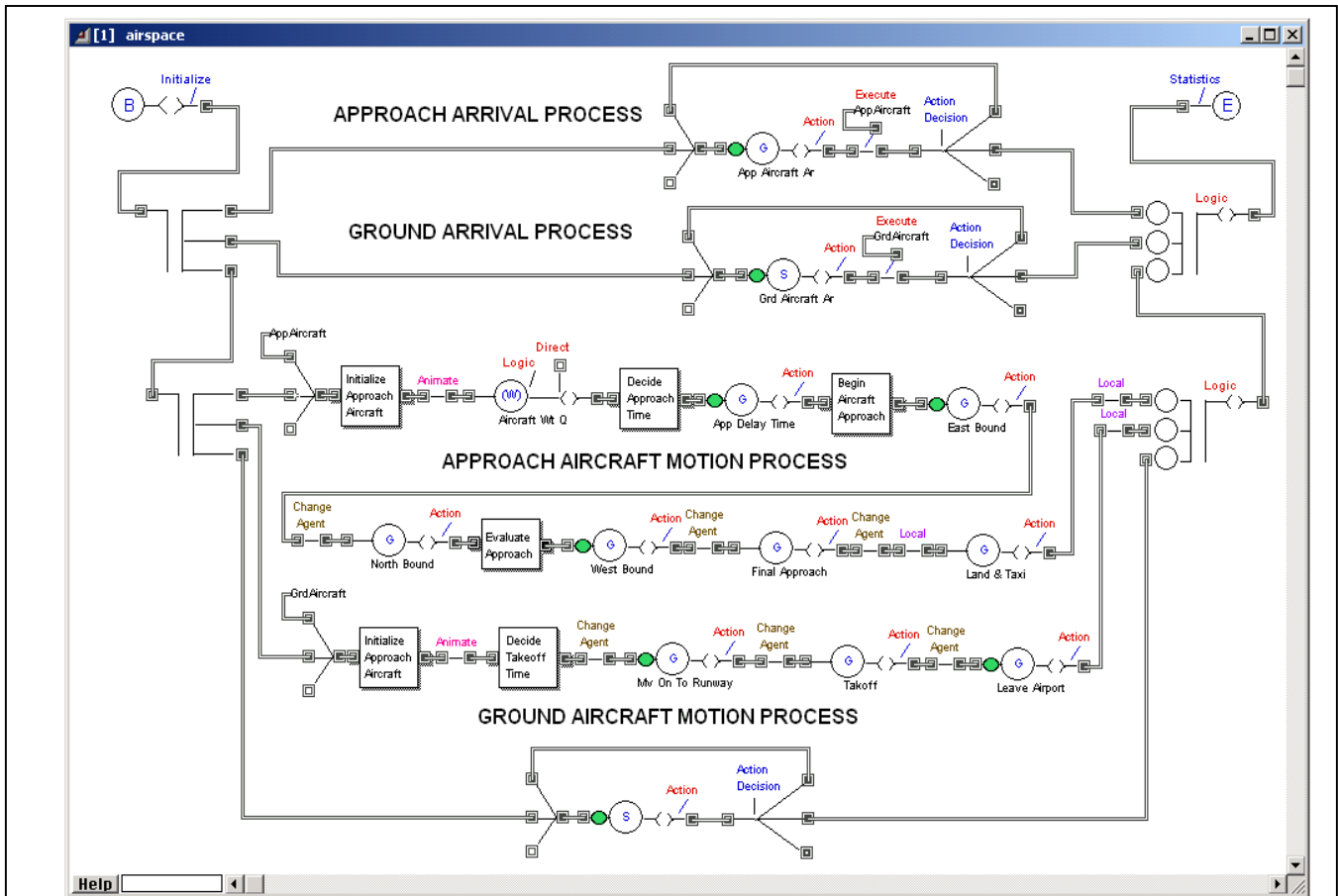


Figure 2. OpEMCSS Directed Graph Model for the VFR Airport Simulation

In general, the top two processes are generator processes that simulate the inter-arrival time of aircraft that enter the system. The bottom two processes model the behavior of aircraft while in the system. Each time an aircraft arrives, a generator process creates an aircraft process instance and sends it to a motion process. Thus, a motion process is instantiated for each aircraft in the system. In this sense, OpEMCSS is object-oriented because a single process diagram simulates a variable number of aircraft instances. This feature allows a variable number of agents to be modeled by a single static diagram rather than having to duplicate the process diagram for each agent instance desired.

Figure 3 shows the contents of hierarchical block “Decide Approach Time “. The Classifier Event Action block transforms feature facts Distance, Closing-Velocity, and Speed into the proper wait time before entering the approach corridor. The Distance is the distance in feet along the approach corridor between the aircraft in the approach queue and next aircraft ahead in the approach corridor. The Closing-Velocity is the difference in Speed between the aircraft in the approach queue and the next aircraft ahead in the approach corridor. If closing-velocity is negative then the aircraft in the approach queue is slower and cannot overtake the aircraft ahead. The Speed is the airspeed of the aircraft in the approach queue. Wait time in the queue is reduced when the average speed of the two aircraft is higher.

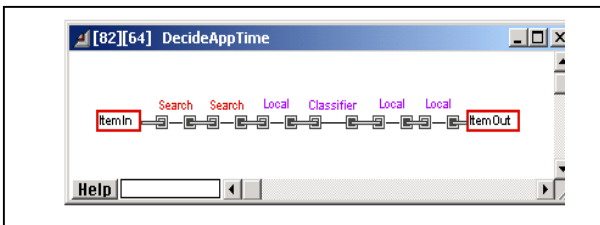


Figure 3. Contents of Hierarchical Block “Decide Approach Time “

Fuzzy Classifier Event Action Block

The Classifier Event Action block contains a forward chaining, inference engine that uses condition-action rules to transform condition attributes into action attributes. The condition attributes are obtained from a process instance item

passing through the block [9]. After inference is complete, action attributes are added to the process instance item passing through the block before the item is sent to the output connector. An example rule is as follows:

```
RuleName:IF
  ConditionName1 = ValueName1 AND
  ConditionName1 = ValueName2 AND
  ConditionName2 = ValueName3,
THEN
  ActionName1=ValueName4 AND
  ActionName2=ValueName5, CF=100.0%
```

Process attributes of the form "AttributeName = RealValue," that represent state variables in EXTEND, must be transformed into a symbolic form "AttributeName = ValueName" suitable for the inference engine. Low and High values are specified by a command in the rule file that defines the range of numerical values allowed for each symbolic value name. The result of each transformation is a set of one or more facts of the form "AttributeName=ValueName" that can be used during the inference process.

A similar command must be placed at the beginning of the rule file to accomplish the required reverse transformation when a rule fires. The Low and High values for action attributes define the range of numerical values allowed for each symbolic value name. The result of each transformation is one or more process attributes of the form "AttributeName = RealValue" added to the process instance item passing through the block.

```
LegalConditionVals (Distance)=Closest (0:625) ,
  Close (625:1875) , Near (1875:3500) ,
  Far (3500:6500) , VeryFar (6500:13000) ,
  VeryVeryFar (13000:50000)
LegalConditionVals (Speed) =Low (50:100) ,
  Medium (100:200) , High (200:300) ,
  VeryHigh (300:1000)
LegalConditionVals (ClosingVelocity)=
  Neg (-1000:-37.5) , Zero (-37.5:37.5) ,
  Low (37.5:112.5) , Medium (112.5:187.5) ,
  High (187.5:1000)

LegalActionVals (AppDelayTime)=No_Wait (0:0) ,
  Wait_50 (50:50) , Wait_100 (100:100) ,
  Wait_150 (150:150) , Wait_200 (200:200) ,
  Wait_250 (250:250)
LegalActionVals (AppDelayTime)=
  Wait_300 (300:300) , Wait_350 (350:350) ,
  Wait_400 (400:400) , Wait_450 (450:450) ,
  Wait_500 (500:500)
```

The rule file definitions, listed above, were used to generate a set of crisp rules that provide exemplars for a discrete mapping of the wait time equation. When these rules are used with their corresponding fuzzy rule definitions, fuzzy control results. More about this fuzzy rule generation approach will be discussed later.

The Rule learning process for crisp rules begins with a subset of the most general rules possible, given the above rule file definitions, as the initial knowledge base such that each possible value for one condition fact is covered. The classifier block uses these rules to make decisions and the reward block evaluates the quality of each decision, sending a payoff to the classifier block. The classifier uses the payoff to reward or punish the rules. The result is that rule strength increases for good rules and decreases for bad rules.

The rule induction algorithm randomly selects rules for modification, from the current set of rules that cover a situation, based on rule strength and decision ambiguity. Decision ambiguity is high when there are several rules with about the same rule strength but specify different actions. Decision ambiguity is low when one strong rule decides in each decision situation. Rule induction for a situation stops when one rule dominates, allowing rule learning to focus on other situations that still have high ambiguity.

The induction operators that can be applied are change a fact value in the selected rule, add a new fact to the rule, and delete an old fact from the rule. These operators are applied to either the condition or action of a rule based on a probability. Another probability is used to decide whether to change a rule fact or add / delete a fact. Given add / delete is selected, a probability function is used to select either add or delete. Initially, this probability function is zero and then increases exponentially, as rules are modified, until a maximum value of 0.9 is reached. The result is that rule search initially focuses on the most general rules at the top of the decision rule network but eventually generates more specific rules further down the rule network.

In summary, the search for the best rules is guided by situational ambiguity and it proceeds from the top to the bottom of the rule network. The result is that a minimal set of the most general rules that can make all decisions correctly is found.

Because of the inductive bias implemented in this evolutionary rule induction algorithm, only a small fraction of the total number of possible rules is examined.

For fuzzy rules, the inference engine requires a Confidence Factor (CF) for each condition in order to compute the rule support that is needed to calculate the weighted average for the action attribute real value. The format of the ConditionFuzzySet command is:

ConditionFuzzySet(AttributeName) =
ValueName(A, B, C)

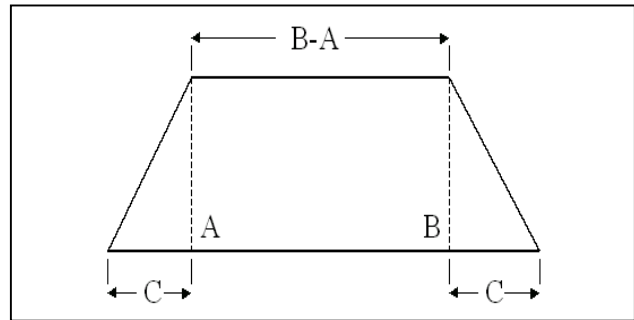


Figure 4. Fuzzy Set Membership Function Format.

The fuzzy set function defined by A, B, and C has a trapezoidal shape shown in figure 4. The top of the trapezoid (length [B-A]) is smaller than or equal to the base (length [[B-A]+2C]). At the top of the trapezoid, the CF is 100. The A and B values define where the function begins a linear descent to zero. The C value defines the slope, $ABS(100/C)$, of the descent. Thus, if C is zero, the trapezoid becomes a rectangle. If A equals B and C is not zero, the function has a triangular shape.

When the classifier system block is used as a fuzzy controller, several rules may be eligible in a decision context. A weighted average is calculated over all eligible rules to produce action attribute values used to control traffic light timing.

Fuzzy sets can be defined for action facts as well as condition facts. The fuzzy set function takes an action fact confidence factor (CF) and transforms it into an output, real attribute value "AttributeName = RealValue" that is added to the process instance item passing through the block. The real attribute value is computed as the weighted average over all occurrences of each

AttributeName, obtained from the set of eligible rules, using the fuzzy set definition. The ActionFuzzySet command that defines the conversion of Cfs into output real attribute values is as follows:

$$\text{ActionFuzzySet(AttributeName)} = \text{ValueName(A, B, C)}$$

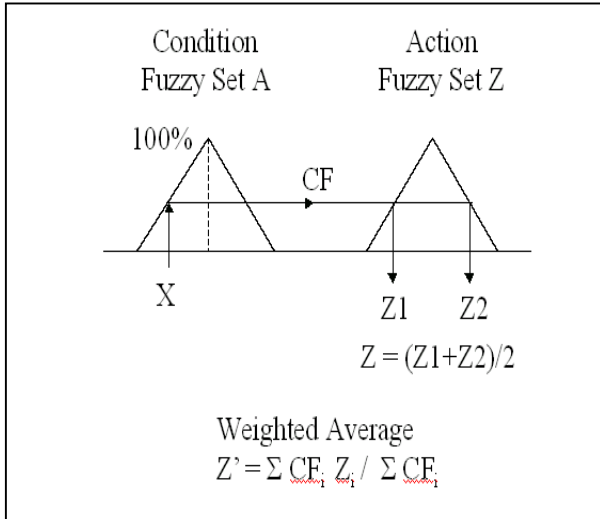


Figure 5. Weighted Average Method.

The value X shown in figure 5 is obtained for a rule condition fact. The condition fuzzy set definition for the fact is used to obtain the degree of fuzzy set membership CF as discussed above. If several fuzzy conditions are combined by an AND operation, the minimum CF for each fact is used for value X, and it is then used to obtain values Z1 and Z2 as shown in figure 5. The average value $Z = (Z1 + Z2) / 2$ is computed as shown. A weighted average Z' is computed for all eligible rules that have the same action name but different value names. This calculation is done for each action name specified by the eligible rules. Therefore, a rule that has several different fuzzy action attributes, has a fuzzy output value computed for each fuzzy action attribute. Each fuzzy action attribute, specified for the selected rule, is added to the process instance item passing through the block using the fuzzy action real value calculated for it.

The rule definitions for the fuzzy wait-time decision rules are shown below. Figures 6 shows the distance condition fuzzy sets used in the VFR airport air traffic control rules. The other condition and action fuzzy sets are similar.

```

LegalConditionVals (Distance)=
  Closest(-50000:1250), Close(0:2500),
  Near(1250:3750), Far(2500:7500),
  VeryFar(5000:15000),
  VeryVeryFar(10000:50000)
ConditionFuzzySet (Distance)=
  Closest(-50000,0,1250)
ConditionFuzzySet (Distance)=
  Close(1250,1250,1250)
ConditionFuzzySet (Distance)=
  Near(2500,2500,1250)
ConditionFuzzySet (Distance)=
  Far(5000,5000,2500)
ConditionFuzzySet (Distance)=
  VeryFar(10000,10000,5000)
ConditionFuzzySet (Distance)=
  VeryVeryFar(15000,50000,50000)
LegalConditionVals (Speed)=
  Low(-1000:150), Medium(50:250),
  High(150:350), VeryHigh(250:1000)
ConditionFuzzySet (Speed)=
  Low(1000,50,100)
ConditionFuzzySet (Speed)=
  Medium(150,150,100)
ConditionFuzzySet (Speed)=
  High(250,250,100)
ConditionFuzzySet (Speed)=
  VeryHigh(350,1000,100)
LegalConditionVals (ClosingVelocity)=
  Neg(-1000:0), Zero(-75:75),
  Low(0:150), Medium(75:225),
  High(150:1000)
ConditionFuzzySet (ClosingVelocity)=
  Neg(-1000,-75,75)
ConditionFuzzySet (ClosingVelocity)=
  Zero(0,0,75)
ConditionFuzzySet (ClosingVelocity)=
  Low(75,75,75)
ConditionFuzzySet (ClosingVelocity)=
  Medium(150,150,75)
ConditionFuzzySet (ClosingVelocity)=
  High(225,1000,75)
LegalActionVals (AppDelayTime)=
  No_Wait(-50:50), Wait_50(0:100),
  Wait_100(50:150), Wait_150(100:200),
  Wait_200(150:250), Wait_250(200:300)
LegalActionVals (AppDelayTime)=
  Wait_300(250:350), Wait_350(300:400),
  Wait_400(350:450),
  Wait_450(400:500), Wait_500(450:550)
ActionFuzzySet (AppDelayTime)=
  No_Wait(0,0,50)
ActionFuzzySet (AppDelayTime)=
  Wait_50(50,50,50)
ActionFuzzySet (AppDelayTime)=
  Wait_100(100,100,50)
ActionFuzzySet (AppDelayTime)=
  Wait_150(150,150,50)
ActionFuzzySet (AppDelayTime)=
  Wait_200(200,200,50)
ActionFuzzySet (AppDelayTime)=
  Wait_250(250,250,50)
ActionFuzzySet (AppDelayTime)=
  Wait_300(300,300,50)
ActionFuzzySet (AppDelayTime)=
  Wait_350(350,350,50)
ActionFuzzySet (AppDelayTime)=
  Wait_400(400,400,50)
ActionFuzzySet (AppDelayTime)=
  Wait_450(450,450,50)
ActionFuzzySet (AppDelayTime)=
  Wait_500(500,500,50)
    
```

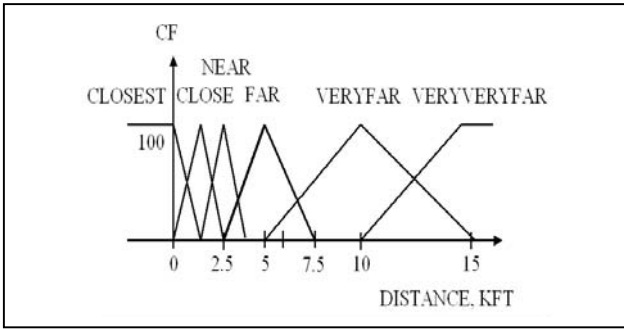


Figure 6. Distance-Condition Fuzzy Sets.

The four-dimensional control surface is represented by 120 maximally specific rules or cases (i.e., all three conditions in each rule). During VFR air traffic control simulated operation, some of the 120 cases defining the control surface occur very rarely, making the rule-learning problem very difficult, and some of the 120 cases never occur at all due to condition attribute interactions. The non-occurring cases are eliminated using a rule tax (probability 0.001). An additional simulation program was developed to make each combination of conditions or case equally likely to speed up learning. Rule generation using this program produced 193 rules in a default hierarchy; therefore, some of the 120 cases had several rules representing them, including both general and specific.

Summary and Conclusions

Figure 7 compares the performance of the air traffic control equation with the performance of the fuzzy controller. The performance is the same until around 1400 operations per day where the curves diverge.

For ground aircraft, if no approach aircraft is in final approach and no aircraft is on the runway, then the ground aircraft can move onto the runway. Thus, the average wait time for ground aircraft is small (between 0.1 and 0.5 minutes) because the traffic control equation is spacing the approach aircraft such that ground aircraft can move onto the runway and takeoff within the 40-second separation time. Therefore, ground aircraft wait time is not shown in figure 7.

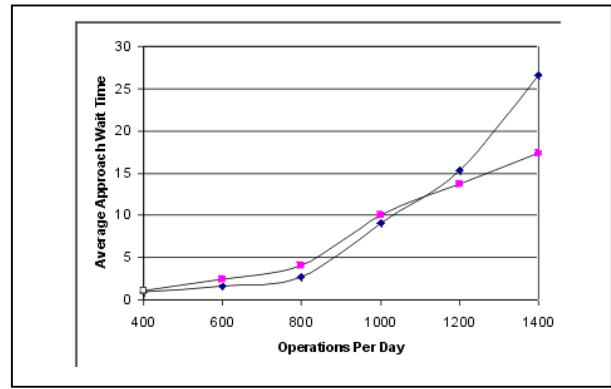


Figure 7. Average Approach Aircraft Wait time as Function of Operations Per Day.

For approach aircraft, as the number of operations per day (takeoffs or landings) increases, the longer the wait time in the approach queue. There is a knee in the curve at 800 operations per day. At 800 operations per day there is about a 5 minute wait in the approach queue and at 1000 operations per day the wait doubles.

The classifier block in OpEMCSS is designed for two types of learning: rule induction and reinforcement learning. Rule generation is the same for both as discussed above. The classifier block starts with a small set of very general rules and searches the rule space top-down from general to specific rules. The result is usually a default hierarchy of crisp rules, which consists of a set of general to more specific rules that all post the same action for a case. If the case is ambiguous, several rules will post different action values for the case. When such crisp rules are operated under fuzzy rule definition, the weighted average method combines all the different action values into a single value.

The crisp rules generated as discussed above were combined with the rule definitions for the fuzzy wait-time decision rules, discussed above. These rules were used in the airport simulation program to decide approach aircraft wait time. Evaluation of decision performance revealed cases where wait time was incorrect. The crisp and fuzzy set definitions were modified until decision performance was acceptable. In particular, it was noticed that cases where the distance is very small seem to be very sensitive to decision performance. More fuzzy sets were used for small distances in order to improve decision performance.

The wait-time decision values obtained from the equation and the fuzzy rules compare closely until around 1400 operations per day where average performance diverges, as shown in figure 7. At 1400 operations per day, aircraft queue up such that separation distance at each decision time is zero. As discussed above, decision performance is very sensitive to small separation distances, which may have caused the observed divergence. However, fuzzy wait-time values continued to compare closely to the equation generated wait time values at 1400 operations per day. Further, queuing theory tends to support the fuzzy average wait time performance curve rather than the equation generated curve. Therefore, this anomaly is still under investigation.

References

- [1] Casti, J. *Would-Be Worlds*; Wiley: New York, 1997.
- [2] Clymer, J. R., *Systems Analysis Using Simulation and Markov Models*, Englewood Cliffs, NJ: Prentice-Hall Inc, 1990.
- [3] Clymer, J. R., Corey, P. D., and J. Gardner, "Discrete Event Fuzzy Airport Control", IN IEEE Transactions on Systems, Man, and Cybernetics, Volume 22, Number 2, March-April 1992, pages 343-351.
- [4] Clymer, J. R., "Expansionist/Context-Sensitive Methodology: Engineering of Complex Adaptive Systems", IN IEEE Transactions on Aerospace and Electronic Systems, April 1997 issue, Volume 33, Number 2, pages 686-695.
- [5] Clymer, J.R., "Simulation-Based Engineering of Complex Adaptive Systems," In Simulation, Journal of the Society of Computer Simulation, San Diego, CA, Volume 72, Number 4, April 1999 issue, pages 250-260.
- [6] Clymer, J.R., "Optimization of Simulated Systems Effectiveness using Evolutionary Algorithms," IN Simulation, Journal of the Society of Computer Simulation, San Diego, CA, Volume 73, Number 6, December 1999, pages 334-340.
- [7] Clymer, J.R., "Optimizing Production Work Flow Using OpEMCSS," IN Proceedings of the 2000 Winter Simulation Conference, J.A. Joines, R.R. Barton, K. Kang, and P.A. Fishwick, Editors, Orlando, FL, December 10-13, 2000, pages 1305-1314.
- [8] Clymer, J.R., and D.J. Cheng, "Simulation-Based Engineering of Complex Adaptive Systems Using a Classifier Block," The 34th Annual Simulation Symposium, Seattle, WA, April 22-26, 2001, pages 243-250.
- [9] Clymer, J.R., *Simulation-Based Engineering of Complex Systems*, John R. Clymer & Associates: Placentia, CA, 2001.
- [10] Holland, J.H., *Hidden Order*, Addison-Wesley, 1995.
- [11] Solow, D., "On the Challenge of Developing a Formal Mathematical Theory for Establishing Emergence in Complex Systems," In Complexity, New York, NY: John Wiley & Sons, Inc., Volume 6, Number 1, September-October 2000, Pages 49-52.
- [12] Weiss, G., editor, *MultiAgent Systems: A Modern Approach to Distributed Artificial Intelligence*, Cambridge, MA : The MIT Press, 1999.