Wings of Wisdom: Learning from Pilot Decision Data with Interpretable AI Models

Boris Djartov[0009−0007−9647−5607], Anne Papenfuß[0000−0002−0686−7006], and Matthias Wies[0000−0001−6514−3211]

German Aerospace Center (DLR),
firstname.lastname@dlr.de
https://www.dlr.de/de

Abstract. This paper explores how an AI model might aid pilots facing time-sensitive, multi-criteria decision-making challenges, focusing on the dynamic alternate airport selection problem. Traditional decision-making methods from the literature are ill suited in time-constrained, stressful situations. This has prompted an exploration into how incorporating AI models might provide decision-makers, pilots in this case, recommendations in such predicaments. Within the paper we explore how a Learning Classifier Systems (LCS), might be employed to tackle the problem. To train the LCS, an augmented dataset is derived from an online survey study featuring scenarios simulating alternate airport decision-making problems where state variables, reflecting aircraft conditions, and three airport options were presented to pilots. The LCS system showed promising results and appears to be a suitable model for the task.

Keywords: mutli-criteria decision-making · learning classifier system · artificial intelligence.

1 Introduction

Making decisions that involve multiple criteria is inherently challenging, and the complexity of such tasks is further exacerbated when the decision needs to be made in a time-sensitive emergency situation. A real world example of this is the dynamic alternate airport selection (DAAS) problem [1]. The DAAS problem aims to represent a situation that pilots may find themselves in, specifically when certain factors necessitate the selection of a new destination airport midflight. In [1] and here, the problem is presented as a multi-criteria decision-making problem in which pilots need to gather information on possible airports, compare their characteristics, and select a new final destination. Although structured schemes such as FORDEC (Facts, Options, Risks, Decision, Execution, Control) or TDODAR (Time, Diagnosis, Options, Decide, Assign Task, Review) are used [2] [3], pilots in interviews expressed their desire for additional decision-support in emergency situations [4].

At the DLR (Deutsches Zentrum für Luft- und Raumfahrt e.V., engl. German Aerospace Center), an Intelligent Pilot Advisory System (IPAS) [4] is being

developed to explore how an AI-powered decision support system might be integrated into the cockpit. Currently, one of the focuses behind this advisory system is to provide pilots assistance during a DAAS problem situation. The idea is that the system would take in relevant input data and provide the pilots with a suggested course of action. This paper delves into how the AI of such a system might be trained using data gathered from a survey, where pilots were tasked to solve a DAAS problem consisting of three alternate airports. Additionally, given the importance of safety in aviation, emphasis was placed on using a more interpretable model whose recommendation would be easier to check. Thus, the lesser-known Learning Classifier System (LCS) was selected to tackle the pilot dataset.

The structure of this paper is as follows: The follwing section provides a description of the pilot dataset used to train the model Afterwards, the LCS is described along with the precise implementation used. Then in 4 the experimental design and results analysis are presented. Finally we have the conclusions drawn and avenues for future research.

2 Dataset

2.1 Data collection

The pilot dataset used was compiled as part of the master's thesis titled "Are there any factors that make the pilots' decisions predictable? An analysis of the impact of conditions on the choice of alternate airports." [7]. The data was gathered through an anonymous online survey using LimeSurvey. The survey took place between October 28, 2022 and November 12, 2022 and had 46 participants. The recruitment of the participants was carried out by direct mail with contacts who were in the the Institute of Flight Guidance's mailing list and were not compensated for their participation. The study consisted of:

- An inclusion criterion: active type approval for the Airbus A320 family (yes or no question).
- Sociodemographic questions: year of birth, gender.
- Experience questions: total flight hours, flight hours last year, flight hours on the A320 type approval, pilot rank.
- Twelve decision-making scenarios.

The scenarios consist of a decision-making problem, in which the participants are asked to rank three airports in terms of what they would choose if they had to select a new destination airport midflight. Within the scenarios the reason to select the new airport is not specified. The aircraft in the scenarios are defined to be fully operational with no technical failures. The following information was provided for the twelve scenarios to provide additional context details to the participants:

Airbus A320 fully loaded with passengers and luggage which weighs 64.5 tons along with the maximum possible landing weight. Cruising altitude FL350.

Normal weather according to ISA conditions without wind until you reach the alternates. Manual landing, A/THR off, auto brake medium, reversers. The information provided was to indicate that the status of the aircraft reflected that of what would be considered average for the aircraft type. Regarding the twelve scenarios themselves, the following characteristics for each airport were provided:

1. Distance alternate to original airport: This factor describes the distance between the originally planned airport and the alternate airport.
2. Related Landing Performance: This code translates runway conditions, such as wet, snowy, icy, etc., into a classification of the expected landing performance of the aircraft.
3. Margin landing distance: This factor is the difference between the length of a runway and the braking distance of the aircraft.
4. Crosswind: the wind blowing across the direction of movement.
5. Tailwind: This factor is calculated from the wind direction and speed using trigonometric functions.
6. Fuel remaining: This factor gives the amount of fuel that is left when the airspace of an airport is reached. The minimum amount can be calculated based on regulatory requirements and the current conditions.
7. Number of runways at the airport.

Additionally, the airports were given generic names with two letters to avoid biasing the pilots' responses. This was done to ensure that the pilots were not influenced by their own personal preferences or experiences. Within the study the twelve scenarios were designed to be difficult in order to challenge the participants. This meant that the scenarios showcased where more extreme scenarios where the true answers is not clear. In fact, in [7] the idea was to use extreme cases so that more insight could be gathered into the trade-offs that pilots would make in terms of airport characteristics. Additionally, the scenarios were divided into difficult and very difficult categories. In this context difficulty refers to the how extreme the values are for the factors in terms of how much they close to breaking the safety guidelines.

2.2 Data analysis

Figure 1 shows the favored or Rank 1 choice of the pilots for each of the scenarios. It is important to note that although 46 participants took part in the study not all of them evaluated all the scenarios presented, hence there are some discrepancy's in the number of evaluations in figure 1. From the figure one can ascertain that many scenarios have more apparent preferences, and it would seem that the participants were of a similar mind in their choices.However, in some scenarios, such as S7 and S9, we see that the margin between the airport ranked first most often and the second-best ranked one is not as big as it is in the other scenarios where we can see a more decisive clear preference. While the participants did not always agree, the dataset was still compiled from experts in a specific field, making it a valuable treasure trove of information that models can learn from.

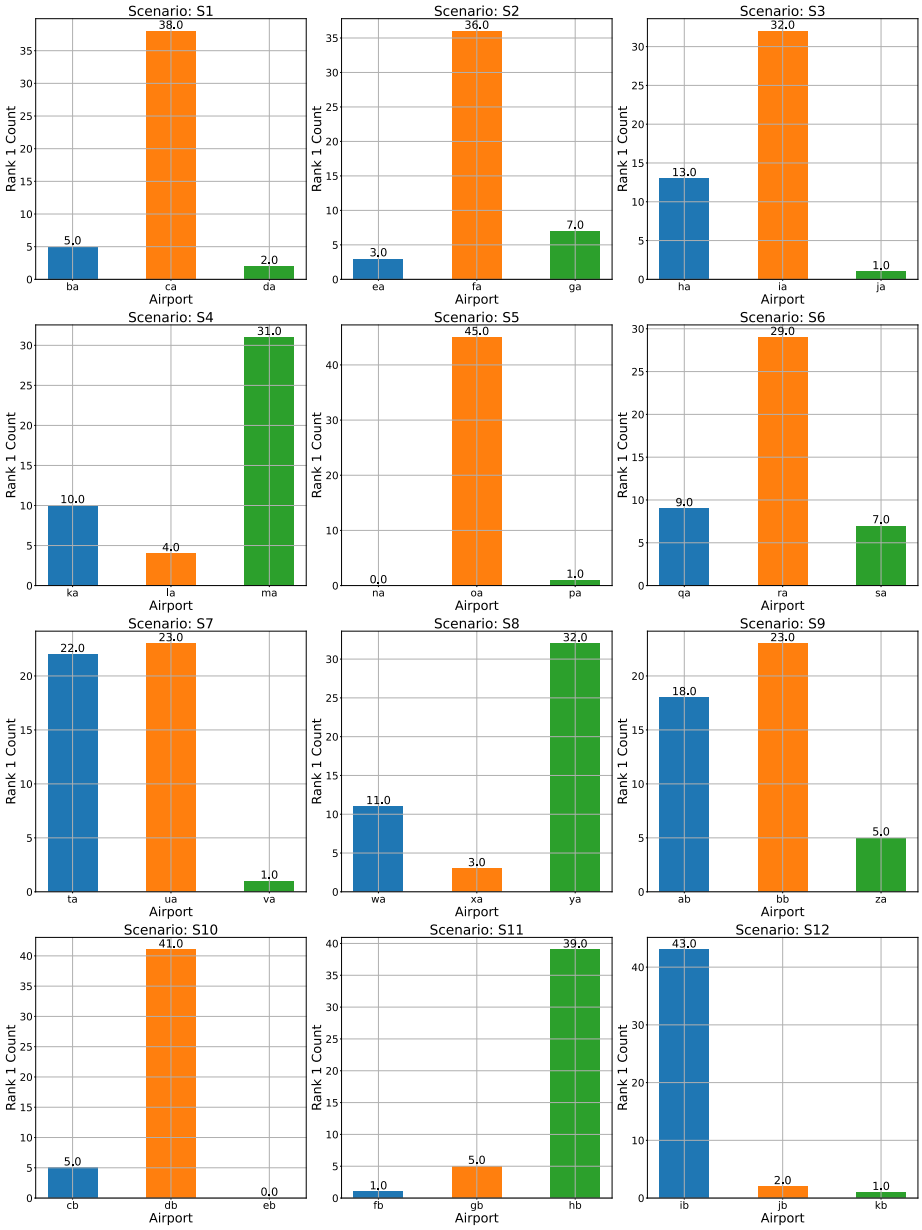

Fig. 1. Analysis of pilot choices

In [7], aside from the wider breath of the investigation analysis the examination of the scenario data was primarily approached by treating all the options/ airports as separate instances of a complete dataset. It is worth mentioning that analysis focusing on the scenarios as complete separate units was included but to a limited degree. To better understand the factors influencing participants' decisions, an additional analysis was conducted. In it each scenario was treated as a separate subdataset of the whole dataset. The initial step taken was to simplify the target variable by transforming the answers from the pilot survey to a single variable instead of having three variables for each of the rankings. To accomplish this, a score for each option/airport was calculated based which incorporates all three ranking. The new score was calculated as:

$$SC(a_i) = R1(a) + \frac{R2(a_i)}{2} + \frac{R3(a_i)}{3} \tag{1}$$

where a refers to the airport selected and Rm refers to the mth ranking from the dataset. This enables the scoring function to take into account the values corresponding to the other rankings and not just focus on the airport which was ranked first. This is needed as figure 1 showcased that in certain scenarios the preferred choice is not entirely clear. This score was then used to calculate the Pearson correlation. The correlations for each scenario were calculated and then averaged. Fisher's z-transform was used for this purpose [9], as it stabilizes the variance of correlation coefficients and makes them more suitable for statistical analysis by normalizing their distribution. Using this transformation the influence of the features across all the scenarios can be examined as depicted in figure 2. Based on Figure 2, the Marginal Landing Distance and Related Landing Performance emerge as the most influential factors when making decisions, exhibiting a correlation of nearly 1. This observation aligns with the logical consideration that pilots prioritize information about available landing distance, recognizing its critical role in preventing runway excursions. A runway excursion refers to an incident in aviation where an aircraft departs from the runway during either takeoff or landing. This can occur due to various factors, such as adverse weather conditions, pilot error, technical malfunctions, or a combination of these elements. Runway excursions may involve the aircraft veering off the side of the runway, overshooting the runway end, or, in more severe cases, going off the runway surface entirely. These incidents can pose safety risks and often prompt investigations to determine the causes and implement preventive measures to enhance aviation safety. Similarly, the substantial correlation for Related Landing Performance is justifiable, given that runway conditions significantly impact the aircraft's braking coefficient, thereby influencing the likelihood of a runway excursion.

Interestingly, crosswind and tailwind exhibit a negative correlation, with crosswind exerting a more pronounced influence. It was initially hypothesized that tailwind might be the more influential factor due to the potential dangers associated with strong tailwinds, making landing precarious or even impossible at certain speeds. However, the data suggests a stronger association with crosswind, challenging the initial assumption and emphasizing the need for a

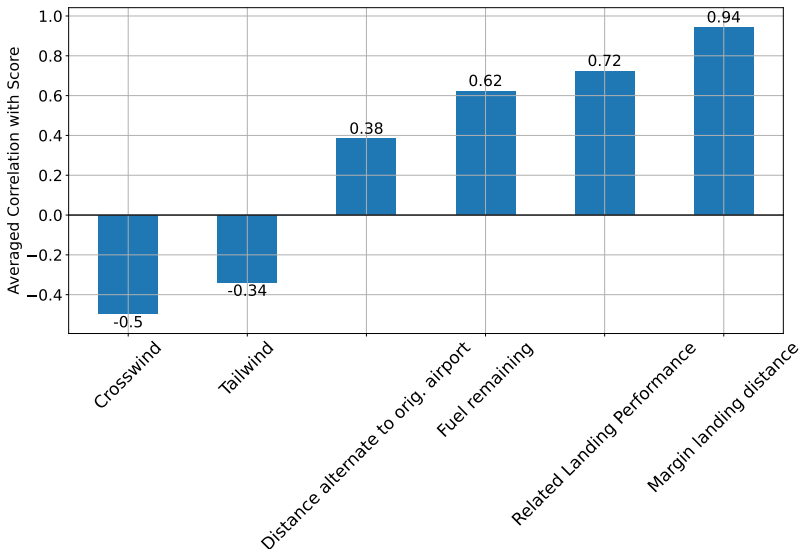

Fig. 2. Feature corelations

nuanced understanding of the interplay between wind conditions and landing performance

2.3 Traning dataset creation

With a better understanding of the dataset, attention can now shift to how it can be utilized for training the model. The scenarios evaluated by participants were designed to reflect a real-world decision-making problem, albeit in a simplified form. Due to time and resource constraints, only this limited subset of scenarios could be assessed. However, training a model on such a small and restricted dataset may not be optimal, as the algorithm would benefit more if it could generalize and handle a broader range of scenarios. Consequently, the model should not only perform well on the smaller pool of evaluated scenarios but also on scenarios that differ from those assessed. To tackle this, a dataset comprising new scenarios was created. These scenarios were generated by sampling airports, including their characteristics, and incorporating them into new scenarios consisting of three airports. The sampling was performed to ensure that no new scenario would contain duplicate airports. Using this method and the fact that $12 * 3 = 36$ airports were available to sample, a total of 988 new scenarios were generated. These new scenarios were combined with the original twelve scenarios to create a dataset of $1,000$ scenarios that could be used to train the LCS. The new dataset, unlike the sampling pool, consists of unlabeled

data. "Unlabeled" refers to the fact that the new scenarios have not been evaluated, and there is no established measure of what constitutes a correct response. Hence, the initial idea was to approach this dataset as a reinforcement learning (RL) problem. In an RL problem, an intelligent agent learns through trial and error by receiving rewards or penalties for its actions [8]. However given that the dataset comprised a single step problem, i.e. only a single action/ classification was needed per scenario the dataset was instead converted to a labeled dataset. This was done by making use of a weighted similarity function for each of the airports/options in the new scenarios. The evaluation for an option a from a scenario S is calculated using:

$$EV(a_i, S_n) = \frac{1}{d_{min}(\vec{x}, \vec{b_j})} * SC(a_i) \tag{2}$$

where n refers to the nth scenario, $d_{min}(\vec{x}, \vec{b_j})$ represent the smallest Eucledian distance between the current input $\vec{S}$ and $\vec{b_j}$ which represents the the jth element of the the created reference matrix b. The reference matrix comprises the original twelve scenarios, flattened so that each scenario is represented by a single row in the reference matrix. The Eucledian distance between any scenario $\vec{S}$ and a scenario from the reference matrix is calculated as:

$$d(S_n, b_j) = \sqrt{(S_1 - b_{j1})^2 + (S_2 - b_{j2})^2 + ...(S_{nk} - b_{jk})^2} \tag{3}$$

where k is the number of factors in our dataset, i.e., the number of airport characteristics being considered. Using the evaluation, we measure the similarity between the scenario from the newly created dataset and an evaluated scenario. Subsequently, each of the available actions in the scenario is evaluated based on the scores from the original twelve scenarios. The closer the scenarios resemble each other, the more the ranking of the options ought to resemble each other. Using this new evaluation for the options, the correct class or label is the airport/option that has the highest value. Thus, the unlabeled dataset, using the original twelve scenarios as a reference guide, is transformed into a labeled dataset that can be used to train the model.

3 Model configuration

An LCS consists of a set of rules known as the population of classifiers. The primary goal is for these classifiers to collectively emulate an intelligent decision maker. This is done through evolution and learning. The genetic algorithm and a problem-specific learning mechanism guide the system toward better rules. Both mechanisms rely on the system's "environment," which, in the context of LCS literature, refers to the source of input data for the algorithm. The information received from the environment depends on the problem being addressed. In this instance, the system will receive decision-making scenarios requiring the selection of one out of three airports. After the interaction with the environment, the LCS will receive feedback in the form of numerical rewards, which

then in turn drive the learning process. While various LCS algorithm implementations exist, a basic framework includes a finite population of classifiers representing system knowledge, a performance component regulating interaction with the environment, a reinforcement component distributing rewards to classifiers, and a discovery component improving rules through different genetic operators. These components serve as the foundation for numerous variations in LCS algorithms. Figure 3 illustrates how specific mechanisms of LCS interact within the context of these major components [18] [19]. Aside from the interpretability and explainability of the model, LCS has also been extensively used for problems characterized by both epistasis, complex interactions between features of a dataset, and heterogeneity, different features or subset of features are important for different instances of the dataset. The same characteristics are present in popular benchmark problems such as the multiplexer problem [20] [18]. Given that these characteristics are also present in the dynamic alternate airport selection problem and that the problem was used as inspiration for the construction of a multi-objective multiplexer benchmark problem [21], using an LCS seems like a suitable model to employ.

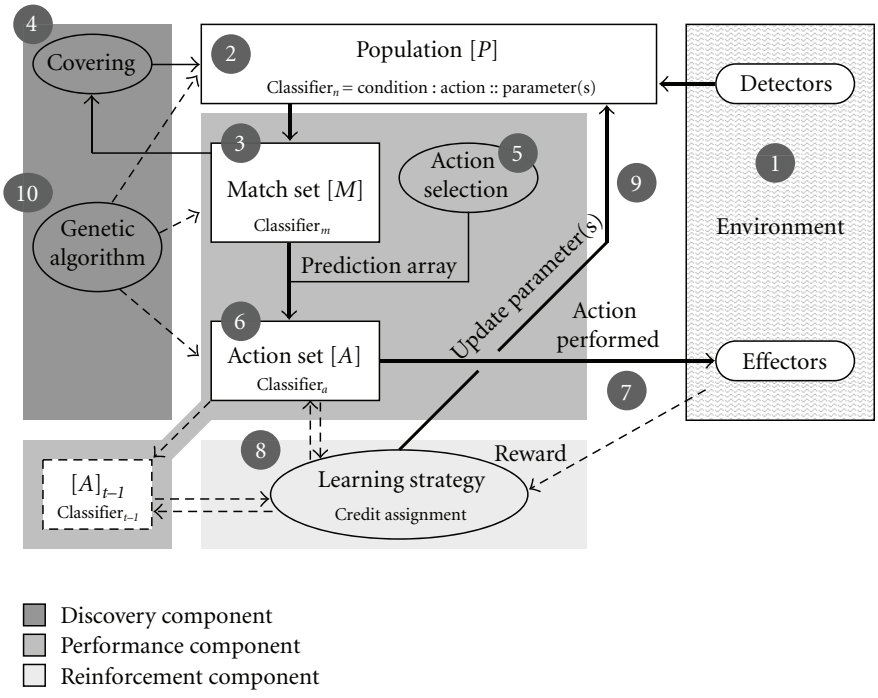

Fig. 3. Visual representation of a LCS, taken from [19]

Due to the large number of LCS variants and the lack of a standardized implementation, here the exact structure of the LCS will be presented. The LCS implemented is loosely based on [24] and is in fact avariation on an enhanced Michigan-style learning classifier, often called XCSF [25] [26]. The LCS is comprised of:

1. The condition C which is a hyperellipsoidal condition and is mathematically represented as:
$$C = (\vec{m}, \Sigma) = ((m_1, m_2, ..., m_n)^T, (\sigma_1, \sigma_2, ..., \sigma_n)), \tag{4}$$
where $\vec{m}$ specifies the center of the ellipsoid, T denotes the transpose of a (column) vector or matrix, and Σ defines the fully weighted Euclidean distance metric of the ellipsoid, also termed Mahalanobis distance [27]. This transformation matrix determines the stretch of the ellipsoid and the rotation in the n -dimensional problem space.
2. The possible action set A, that is, $A \in A$ where $A = \{a1, ..., a_m\}$ represents the set of all possible actions.
3. The linear function prediction R which is specified by the weight vector
$$R = \vec{w} = (w_0, w_1, ..., w_n)^T, \tag{5}$$
where w_o is the offset weight
4. The prediction error ε which estimates the mean absolute deviation of the reward predictions.
5. F which is the fitness of the classifier.

The values are iteratively modified and evolved. The LCS is initialized with an empty population. Each learning iteration t, the LCS receives a instance x_t along with the function value y_t. LCS then forms a match set $[M]$ of all classifiers whose conditions are active. Classifier activity is determined by

$$\text{cl.ac} = \exp\left(-\frac{(\mathbf{x}-\mathbf{m})^T \Sigma^{-1} (\mathbf{x}-\mathbf{m})}{2}\right),$$

which determines the distance from the current input and then applies the Gaussian kernel on the distance. A classifier is active, that is, it matches if its current activity is above the threshold θ_m. For each step within a learning trial the LCS constructs a match set $[M]$ comprised of active classifiers from the population. Each classifier in $[M]$ has its numerocity increased by 1, $cl.num = cl.num + 1$. Should the match set $[M]$ generate fewer than $\theta_m na$ actions a covering mechanism generates new classifiers and adds them to the population $[P]$. The center of the hyperellipsoid ($\mathbf{m}$) fro the newly generated classifiers is set to the current input ($\mathbf{x}$). Only the diagonal entries in the transformation matrix (Σ) are initialized to the squared inverse of the uniformly randomly chosen number between zero and the parameter r_0. All other matrix entries are set to zero. In this way, covering creates axis-parallel hyperellipsoidal condition parts. During learning for each possible action a_k in $[M]$ a classifier prediction p_j is calculated as:

$$cl.P(\vec{x}) = cl.w_0 \times x_0 + \sum_{i>0} cl.w_i \times x_i \tag{6}$$

For each action a_k, prediction p_j, and classifier fitness F_j the expected payoff is computed using:

$$P_k = \frac{\sum_j F_j p_j}{\sum_j F_j} \tag{7}$$

Afterwards a system action is chosen and all of the classifiers in $[M]$ advocating the chosen action are used to create a action set $[A]$. The chosen action is then performed and a scalar reward r is received along with the next input. Upon receiving the reward each classifier in $[A]$ has its weight vector updated according to:

$$cl.w_i \leftarrow cl.w_i + \Delta w_i \tag{8}$$

where Δw_i is calculated as:

$$\Delta w_i = \frac{\eta}{|\vec{x}|^2}(y - cl.P(\vec{x}))x_i \tag{9}$$

where $\eta \in [0, 1]$ denotes the learning rate and y the target output. The error for each classifier is then calculated as:

$$\varepsilon_j \leftarrow \varepsilon_j + \eta\left(|y - p_j| - \varepsilon_j\right) \tag{10}$$

The fitness is updated next using:

$$F \leftarrow F + \eta(\kappa' - F) \tag{11}$$

$$\kappa = \begin{cases} 1 & \text{if } \varepsilon_0 \leq \alpha \left|\frac{\varepsilon}{\varepsilon_0} - \nu\right| \\ 0 & \text{otherwise} \end{cases}$$

$$\kappa' = \frac{\kappa \cdot num}{\sum_{\text{cl}\in[M]} cl.k \cdot cl.num} \tag{12}$$

where $cl.k$ refers to the k value for the specific classifier and $cl.num$ refers to the numerocity of the classifier. Finally the set size estimate of each classifier is updated:

$$a_j \leftarrow a_j + \eta(|[M]| - a_j) \tag{13}$$

The LCS employs a Evolutionary algorithm (EA) for the evolution of classifiers. EAs are a type of algorithm inspired by the natural process of evolution and attempt to solve a problem by evolving and mutating potential solutions to the problem while also applying specific selection pressures. The EA applied to classifiers within $[A]$ it he average time since it's last execution exceeds θ_{EA}. The EA used here employs set-size relative tournament selection [28] based on the fitness estimates of classifiers to choose two parental classifiers from the current match set $[M]$. This means that from the set $[M]$, a random sample is selected to participate in a tournament of varying size. The size of the tournament refers to the number of classifiers that will be compared simultaneously. The classifiers are compared based on their fitness, as this is the mechanism the GA uses to determine how "fit" a classifier is. Subsequently, two offspring are produced

through crossover and mutation operations. Uniform crossover is used in which the corresponding values in the two selected classifier are exchanged with a probability of 0.5. Mutation, with a probability of μ, modifies each entry in the condition part by either randomly shifting the center of the hyperellipsoid within its current interval. During mutation, a matrix entry undergoes a maximal increase or decrease of the value by 50%. If the value is initially set to zero, it is initialized to a randomly selected value, using the same method from covering process for diagonal matrix entries, while taking into account the parameter μ_0. The implemented LCS also has a deletion operator in order to maintain the population size . The deletion mechanism initially establishes the average fitness of all classifiers in the population. Next, each classifier is assigned a deletion vote based on its match count and fitness. The deletion vote initially equals the classifier's estimated set size. If the classifier's match count exceeds a threshold θ_{del} and its fitness falls below a fraction δ, its deletion vote is further increased. This ensures that classifiers with many matches and low fitness are more likely to be selected for deletion. A probability distribution where classifiers with higher deletion votes have a higher probability of being selected for deletion is created by normalizing the deletion votes. Using this probability distribution, a classifier is randomly chosen for deletion and removed from the population. This process is repeated until the population size reaches the maximum allowed limit.

4 Experimental design and results analysis

The experiments were carried out in Python version 3.9.6 and made use of the XCSF Python package [30]. LCS is characterized by a large number of hyperparameters, i.e., parameters that need to be given beforehand. Due to this, hyperparameter tuning was done using Bayesian optimization and made use of the Bayesian Optimization: Open-source constrained global optimization tool for Python package [29]. This method employs a Gaussian process to establish a posterior distribution of functions that accurately represents the function being optimized. As the number of observations increases, the posterior distribution becomes more refined, empowering the algorithm to make more informed decisions about which regions of the parameter space are worth exploring and which are not [31]. The Bayesian optimization was carried out using twenty initial points and fifty additional evaluation points. For each combination of parameters, the model was run thirty one times due to its stochastic nature and had 10% of the dataset used as a validation set. The weighted F1 score was used as a measure for hyperparameter tuning, i.e., the parameters were optimized to obtain the highest average (from the thirty one runs per combination of parameters) weighted F1 score. The Weighted F1 Score is calculated using the following formula:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{14}$$

where TP refers to the number of true positives predictions and FP refers to the number of false positives, additionally FN refers to the number of false negatives and TN refers to the number of true negative predictions.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{15}$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{16}$$

The Weighted F1 Score is then computed as the weighted average of the F1 Scores for each class, where the weights are based on the number of true instances for each class:

$$\text{Weighted F1 Score} = \frac{\sum_i (\text{TP}_i + \text{TN}_i) \times \text{F1 Score}_i}{\sum_i (\text{TP}_i + \text{FN}_i)} \tag{17}$$

The Weighted F1 Score is then computed as the weighted average of the F1 Scores for each class, where the weights are based on the number of true instances for each class:

$$\text{Weighted F1 Score} = \frac{\sum_i \text{TP}_i \times \text{F1 Score}_i}{\sum_i \text{TP}_i + \text{FN}_i} \tag{18}$$

The best-found hyperparameters are illustrated in Table 1.

Hyperparameter	Lower bound	Upper Bound	Optimal Value
population size	100	200	163
α	0.01	1	0.30
β	0.01	1	0.1
δ	0.01	1	0.4646
error reduc.	0.01	1	0.264
fit reduc.	0.01	1	0.265
max nu. trials	1000	10000	7937
ν	1	10	5.383
crossover probability	0.01	1	0.46
patience	10	10000	4540
perf. trials	1	500	142
θ_{del}	20	200	159.873
θ_{EA}	20	200	155.066

Table 1. Hyperparameter values

As depicted in table 1, the population for the LCS was chosen quite conservatively, ranging between 100 and 200 classifiers. This decision was made to exert additional pressure on the LCS to create classifiers that can generalize well. The optimal number found was 163 classifiers, meaning that, on average, one rule would represent $1000/163 = 6.13$ inputs/scenarios from the dataset. However, this is a very rough and simplified estimate, as the classifiers in this configuration of the LCS can overlap. Therefore, it is challenging to determine if solely one classifier from the population represents 6 inputs/scenarios. It is more likely that various subsets of classifiers from the population together are responsible for multiple similar inputs. Another noteworthy hyperparameter is the patience parameter, whose optimal value was estimated to be 4540. This hyperparameter is used as a stopping criterion for the algorithm. If there is no improvement within the last patience number of inputs/scenarios, the algorithm stops training. The chosen number seems quite high compared to the total number of trials. In this context, trials refer to the number of inputs/scenarios given to the algorithm, with a number greater than the size of the dataset indicating multiple passes on the dataset. For the maximum number of trials, it would indicate that the LCS went over the dataset almost 8 times, as the optimal value for this number is 7937.

Using the best-found hyperparameters, an instance of the LCS was run thirty one times, and its precision was measured and compared to that of a random agent. The random agent randomly selects from one of the three available classes for its prediction. The comparison can be seen on figure 4. Examining the figure we an observe that the average accuracy of the LCS lies somewhere between 0.55 and 0.6, which would indicate that it gets the answer right around 60% of the time. Although the LCS has a wider distribution for it's accuracy values even at it's worst it would still outperform the random agent. Although being able to outperform a random agent the LCS still has a mildly impressive accuracy. Additionally, during an exploration into the training of the algorithm it was noticed that in many training sessions of the model there would be a decline of the validation error followed by a plateau or oscillation by the validation error and often, but not always a sudden drop in the validation error. An example of it can be seen in figure 5. This behavior could be caused by the more stochastic nature of the algorithm, as the evolution of classifiers may not be sufficient, and at some point, one or a subset of classifiers is created in the population that drastically improves performance. However, it does seem like a problem, as this oscillation can last for many trials, and in some cases, it was never able to reduce. To this end, greater investigation should be given to the stability and consistency of the training of the algorithm, as this undoubtedly influences the overall performance of the algorithm.

Fig. 4. LCS accuracy versus random agent accuracy

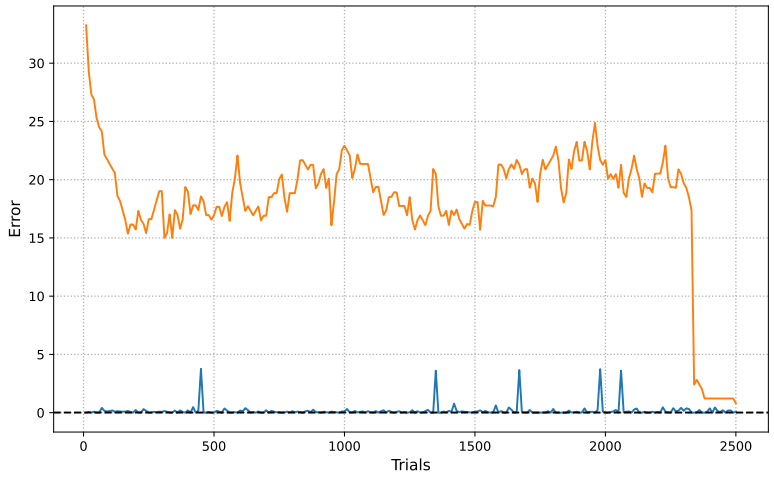

Fig. 5. LCS error across trials

5 Conclusion and future work

This paper explores the utilization of an AI model to assist pilot decision-making in emergency situations. The model was trained on a dataset from an online survey assessing how pilots would decide in twelve scenarios representing a simplified version of midflight emergency alternate airport selection. The dataset was processed using the Learning Classifier System (LCS), a machine learning approach employing a set of classifiers to simulate intelligent decision-making. LCS was chosen for its enhanced explainability and interpretability compared to more contemporary models like neural networks. The LCS demonstrated promise, achieving an accuracy of approximately 0.6 and outperforming a random agent. However, the algorithm exhibited training instability, leading to a lack of reduction in validation error. Moving forward, efforts focus on enhancing LCS performance and constructing more elaborate datasets for model testing. The ultimate goal is to develop a real-world system that enhances aviation safety and earns pilots' trust.

References

1. B. Djartov, S. Mostaghim, A. Papenfuß and M. Wies, "Description and First Evaluation of an Approach for a Pilot Decision Support System Based on Multi-attribute Decision Making," 2022 IEEE Symposium Series on Computational Intelligence (SSCI), Singapore, Singapore, 2022, pp. 141-147, doi: 10.1109/SSCI51031.2022.10022076.
2. Walters, A.J., 2002. Crew resource management is no accident. Aries.
3. Hörmann, H.J., 1994. FOR-DEC-A prescriptive model for aeronautical decision making.
4. Würfel, J., Djartov, B., Papenfuß, A. and Wies, M., 2023. Intelligent pilot advisory system: The journey from ideation to an early system design of an AI-based decision support system for airline flight decks. AHFE 2023.
5. European Union Aviation Safety Agency (2023) EASA-AI-Roadmap 2.0: A human-centric approach to AI in aviation, Cologne.
6. Majumder, M. and Majumder, M., 2015. Multi criteria decision making. Impact of urbanization on water shortage in face of climatic aberrations, pp.35-47.
7. Keul, M. 2023. Are there any factors that make the pilots decision predictable? An analysis about the influence of the conditions on the choice of alternate airports. (Master's Thesis). Hochschule Fresenius, Frankfurt am Main
8. K. Arulkumaran, M. P. Deisenroth, M. Brundage and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," in IEEE Signal Processing Magazine, vol. 34, no. 6, pp. 26-38, Nov. 2017, doi: 10.1109/MSP.2017.2743240.
9. Howell, D. C. ,2014. Statistical methods for psychology (8th ed.). Wadsworth Publishing Company. (pp. 222-224)
10. Mi, X., Tang, M., Liao, H., Shen, W. and Lev, B., 2019. The state-of-the-art survey on integrations and applications of the best worst method in decision making: Why, what, what for and what's next?. Omega, 87, pp.205-225.
11. Lempert, R.J., Groves, D.G., Popper, S.W. and Bankes, S.C., 2006. A general, analytic method for generating robust strategies and narrative scenarios. Management science, 52(4), pp.514-528.

12. Groves, D.G. and Lempert, R.J., 2007. A new analytic method for finding policy-relevant scenarios. Global Environmental Change, 17(1), pp.73-85.
13. Kwakkel, J.H., Haasnoot, M. and Walker, W.E., 2015. Developing dynamic adaptive policy pathways: a computer-assisted approach for developing adaptive strategies for a deeply uncertain world. Climatic Change, 132, pp.373-386.
14. Hamarat, C., Kwakkel, J.H., Pruyt, E. and Loonen, E.T., 2014. An exploratory approach for adaptive policymaking by using multi-objective robust optimization. Simulation Modelling Practice and Theory, 46, pp.25-39.
15. Trindade, B.C., Reed, P.M., Herman, J.D., Zeff, H.B. and Characklis, G.W., 2017. Reducing regional drought vulnerabilities and multi-city robustness conflicts using many-objective optimization under deep uncertainty. Advances in Water Resources, 104, pp.195-209.
16. Watson, A.A. and Kasprzyk, J.R., 2017. Incorporating deeply uncertain factors into the many objective search process. Environmental Modelling and Software, 89, pp.159-171.
17. Bozorg-Haddad, O., Zolghadr-Asli, B. and Loaiciga, H.A., 2021. A handbook on multi-attribute decision-making methods. John Wiley and Sons.
18. Urbanowicz, R.J. and Browne, W.N., 2017. Introduction to learning classifier systems. Springer.
19. Urbanowicz, R.J. and Moore, J.H., 2009. Learning classifier systems: a complete introduction, review, and roadmap. Journal of Artificial Evolution and Applications, 2009.
20. Alvarez, I.M., Browne, W.N. and Zhang, M., 2016, July. Human-inspired scaling in learning classifier systems: Case study on the n-bit multiplexer problem set. In Proceedings of the Genetic and Evolutionary Computation Conference 2016 (pp. 429-436).
21. Djartov, B. and Mostaghim, S., 2023, July. Multi-objective Multiplexer Decision Making Benchmark Problem. In Proceedings of the Companion Conference on Genetic and Evolutionary Computation (pp. 1676-1683).
22. Olive, X., Tanner, A., Strohmeier, M., Schäfer, M., Feridun, M., Tart, A., Martinovic, I. and Lenders, V., 2020, October. OpenSky Report 2020: Analysing in-flight emergencies using big data. In 2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC) (pp. 1-10). IEEE.
23. Eurocontrol. (2024). Aircraft Performance Details: Airbus A320. Retrieved from https://contentzone.eurocontrol.int/aircraftperformance/details.aspx?ICAO=A320
24. Butz, M.V., Lanzi, P.L. and Wilson, S.W., 2006, July. Hyper-ellipsoidal conditions in XCS: rotation, linear approximation, and solution structure. In Proceedings of the 8th annual conference on Genetic and evolutionary computation (pp. 1457-1464).
25. Wilson, S.W., 1999, July. Get real! XCS with continuous-valued inputs. In International Workshop on Learning Classifier Systems (pp. 209-219). Berlin, Heidelberg: Springer Berlin Heidelberg.
26. Wilson, S.W., 2002. Classifiers that approximate functions. Natural Computing, 1(2-3), pp.211-234.
27. Atkeson, C.G., Moore, A.W. and Schaal, S., 1997. Locally weighted learning. Lazy learning, pp.11-73.
28. Butz, M.V., Goldberg, D.E. and Tharakunnel, K., 2003. Analysis and improvement of fitness exploitation in XCS: Bounding models, tournament selection, and bilateral accuracy. Evolutionary computation, 11(3), pp.239-277.
29. Fernando Nogueira, Bayesian Optimization: Open source constrained global optimization tool for Python, 2014, url = https://github.com/fmfn/BayesianOptimization

30. Preen, Richard John and Pätzel, David Pätzel, XCSF, DOI=10.5281/zenodo.10699246, url = https://github.com/xcsf-dev/xcsf/wiki
31. Snoek, J., Larochelle, H. and Adams, R.P., 2012. Practical bayesian optimization of machine learning algorithms. Advances in neural information processing systems, 25.