Scilight

Applied Mathematics and Statistics
https://www.sciltp.com/journals/ams

Article

Adaptive Resilience via Probabilistic Automaton: Safeguarding Multi-Agent Systems from Leader Missing Attacks

Kuanxiang Wang [1] and Xin Gong [2,*]

[1] The School of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237, China
[2] The School of Cyber Science and Engineering, Southeast University, Nanjing 210096, China
* Correspondence: xingong@seu.edu.cn; Tel.: +86-185-9806-0508

How To Cite: Wang, K.; Gong, X. Adaptive Resilience via Probabilistic Automaton: Safeguarding Multi-Agent Systems from Leader Missing Attacks. *Applied Mathematics and Statistics* **2024**, *1*(1), 4. https://doi.org/10.53941/ams.2024.100004.

Received: 26 August 2024
Revised: 3 October 2024
Accepted: 18 October 2024
Published: 25 October 2024

Abstract: The resilience of leader-following structures has been a hotspot in both academic and industrial research. Existing studies mainly focus on maintaining follower coherence, usually assuming that the leader can always function properly. However, these studies neglect the risk of system paralysis if the leader is compromised. To resolve this problem, this paper leverages probabilistic automata grammar reasoning to investigate how followers can gradually infer their operational rules within the system over time. First, a grammatical inference module is implemented on the followers to enable them to deduce their rules once they receive commands from the leader. Then, this paper proposes three probabilistic automata reasoning methods for this inference: the Algorithm for Learning Regular Grammars with Inference Assistance (ALERGIA), Distinguished String Automata Inference (DSAI), and Minimum Divergent Inference (MDI). By using these methods, a follower can reason about deterministic finite automata from multiple commands issued by the leader, which are then utilized to construct deterministic probabilistic finite automata for representing the follower's rules. Finally, several examples are provided to validate the correctness of these reasoning methods and compare their efficiency in learning probabilistic automata. The results indicate that all three methods achieve an accuracy of 98.535% in learning the correct automata transformation function, and ALERGIA runs slightly faster. These findings suggest that even if the leader is compromised, the agent can still perform tasks autonomously using the inferred rules, thereby avoiding system paralysis.

Keywords: adaptive resilience; probabilistic automaton; leader-following multi-agent systems; leader missing attacks

1. Introduction

In recent years, multi-agent systems (MASs) have been widely used in transportation, energy, healthcare, and other critical domains [1–5], and the problem of system paralysis caused by leader missing attacks in this system has received much attention [6]. In the current highly interconnected and distributed environment, it is increasingly important to improve the resilience of MASs [7]. Compared to single-agent systems, MASs have advantages in robustness, parallel processing, and adaptability to dynamic task requirements and environmental changes, achieving superior performance in complex and evolving scenarios [8–10]. The leader-following architecture of MASs has become a research hotspot, particularly the consensus issues between the leader and followers caused by agent failures or communication link disruptions. Several state feedback protocols based on adaptive control and H_∞ control have been proposed to address sensor and actuator failures in leader-following MASs, where sensor failure refers to deviations in transmitted data from the ideal [11]. Meanwhile, a distributed observer and a proportional-integral control strategy have been developed to resolve multi-Euler-Lagrange control problems in leader-following MASs, particularly under communication link failures and uncertainties in external disturbances [12]. Additionally, a two-layer control approach has been devised, which adopts a trust node policy and a flexible network topology to

realize flexible output time-varying formation tracking in MASs, especially in the face of Byzantine attacks on followers, and this approach also integrates virtual and cyber-physical layers [13]. However, these studies do not consider scenarios where the leader is completely disabled. If the leader fails to function properly due to an attack, the entire system may crash. Therefore, it is necessary to design resilient systems that can maintain overall situational awareness and functionality, even under the condition of individual component failures and partial knowledge distribution. As shown in Figure 1, in an air-sea coordinated combat system, if the command aircraft, as the highest leader of the coordinated system, is compromised by a malicious attack and loses its capability to make decisions, coordinate, and communicate, the entire system is at risk of paralysis. This leads to a critical question: How should the followers operate without guidance from the leaders and how can they continue to complete the remaining tasks?

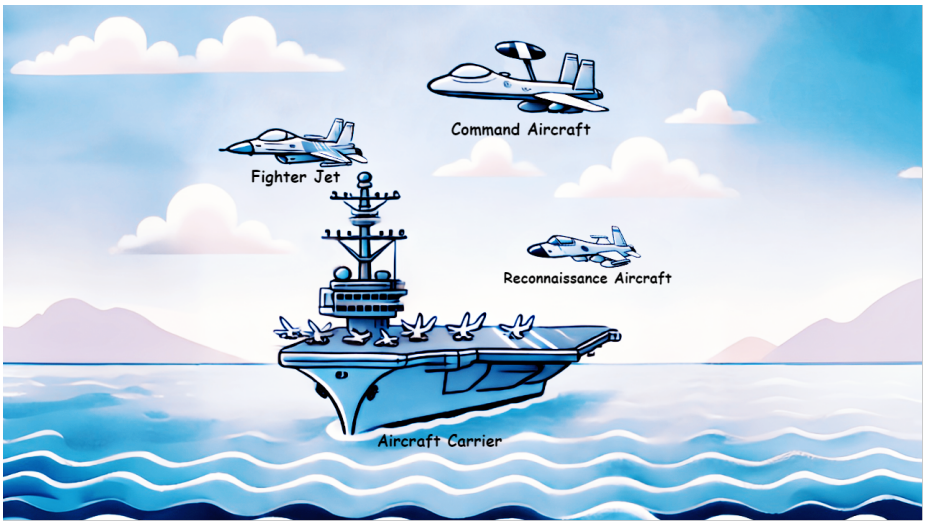

Figure 1. The multi-agent system for air-sea coordinated combat is based on a leader-following structure, comprising command aircraft, reconnaissance aircraft, fighter jet, and aircraft carrier.

There are two main methods to address the issue of leader missing attacks in MASs: Followers either reason about their own rules based on leader commands or re-elect the leader. A secure and resilient supervisory control framework that integrates a learning module and a syntactic reasoning mechanism has been developed for cyber-physical systems. This framework allows followers to gradually infer their own rules over time as leaders transmit commands, thereby ensuring the resumption of operations in the case of coordinator failure [6]. Meanwhile, the Dyna-Delayed Q-learning algorithm has been proposed, which, when combined with distributed reinforcement learning and syntactic reasoning, can improve the resilience of MASs and optimize decision-making after coordinator failure [14]. Additionally, a leader-following cycle strategy has been proposed to resolve the leader error scoring problem, and it assigns an equal distribution of time slices to each follower without qualification management [15]. These studies suggest that the effectiveness of rule inference relies on efficient information exchange and it is crucial for followers to quickly deduce their own rules with limited commands in real-world environments.

In recent years, the rapid development of natural language rule-based reasoning has made probabilistic automata a key method for dealing with linguistic patterns and uncertainty [16–18]. Probabilistic automata enable researchers to effectively capture and manage language randomness during syntactic analysis, semantic understanding, and language generation [19, 20]. This method is particularly suitable for reasoning about and predicting complex linguistic structures, and it allows for more accurate modeling and inference of linguistic rules and their variations, thereby substantially improving the accuracy of comprehension and generation in natural language processing tasks. In MASs where the leader issues commands to followers through feature language [14], sufficient transmission commands allow followers to efficiently infer the rules they follow within the system. As previously mentioned, ensuring a MAS to properly function after the sudden and complete failure of the leader in a leader-following structure remains a significant challenge. Drawing on existing research, our study uses the deterministic probabilistic finite state automaton (DPFA) syntactic reasoning to infer the rules followed by the followers. Meanwhile, the follower is allowed to infer its own rules by reasoning from multiple commands issued by the leader. This is realized by constructing a deterministic finite frequency automaton (DFFA) that represents the follower's rules using folding and merging techniques, as well as the Algorithm for Learning Regular Grammars with Inference Assistance (ALERGIA) [21], Distinguishing Strings Automata Inference (DSAI), and Minimum Divergence Inference (MDI) inference methods. Then, these frequency automata are employed to construct probabilistic automata.

The above discussion indicates that the system collapse problem due to leader missing attacks has not been well addressed in previous studies. In this study, this problem is resolved by establishing the agent's rules as a DPFA and applying various probabilistic automata reasoning methods to infer the agent's own rules. The main contributions of this study are as follows:

(1) Construction of a probabilistic finite state automaton for follower rules: Based on the insights from reference [14], the follower's rule model is established and further extended to a probabilistic finite state automaton to better accommodate dynamic changes. This extension improves the system's ability to manage uncertainty and makes it more stable in complex environments.

(2) Three inference methods for leader missing attacks: This study proposes three inference methods, namely ALERGIA, DSAI, and MDI, which enable followers to autonomously infer and execute operational rules when faced with leader-missing attacks. These methods ensure that the system maintains continuity and stability even in the absence of a fully functional leader.

(3) Comparison of inference speeds among three methods: The inference speeds of ALERGIA, DSAI, and MDI are compared comprehensively to demonstrate the performance differences of these methods across various tasks. The findings provide a basis for selecting the optimal inference method in practical applications.

2. The Establishment of DFFA and DPFA Models

The DFFA models system behaviors by using precise frequency data to define state transitions, ensuring deterministic results in well-defined situations. In contrast, DPFA integrates deterministic transitions with probabilistic distributions, enabling the system to deal with uncertainty and randomness. By combining these two automata, the system can benefit from decision-making stability and the ability to manage events with statistical changes. Therefore, this study models the deterministic behavior of an agent by defining its rules as a DPFA transformed from a DFFA. This integration of precise frequency data of DFFA with the probabilistic distribution of DPFA ensures stability and predictability in decision-making and provides accurate quantification for handling events with statistical and stochastic properties. The notations used in the previous and following sections are listed in Table 1.

Table 1. Notations.

Notations	Description
$\mathcal{A}_f$	Symbol representation of DFFA
Σ_f	Alphabet of DFFA, representing all input symbols the automaton can process
Q_f	Set of states of DFFA, including all possible states the automaton may occupy
I_{fr}	Initial state frequency function, describing the frequency at which the system starts in a specific state
F_{fr}	Final state frequency function, describing the frequency at which the automaton terminates in each state
δ_{fr}	Transition frequency function, defining the frequency of transitioning from one state to another through a specific input symbol
δ_A	Associated transition function, determining the unique path of state transitions given a specific input
w	Observed event sequence in sample S
θ	Transition path in an automaton for a given sequence of events
$\mathcal{B}_p$	Symbol representation of DPFA
Σ_p	Alphabet of DPFA, representing all input symbols the automaton can process
Q_p	Set of states of DPFA, containing all possible states the system may occupy
I_P	Initial state probability function, representing the probability distribution for the system to start in each state

Table 1. *Cont.*

Notations	Description
F_P	Final state probability function, describing the probability distribution for the system to terminate in each state
δ_P	Transition probability function, defining the probability of transitioning from one state to another through a specific input symbol
n	Value of the transition frequency function
P	Transition probability between states in a probabilistic automaton
γ	Probability or frequency difference between two states
$Pr_{\mathcal{A},q}(x)$	Transition probability of state q under input sequence x
$F_{fr}(q)$	Initial frequency of state q
FREQ[q]	Array storing the frequency information for state q
$F_{\mathcal{P}}(q)$	Probability distribution of state q
$\delta_{\mathcal{P}}$	Transition probability distribution
S	Sample set
$\text{cnt}_S(w)$	The number of times string w appears in sample S
$\|\mathcal{B}_p\|$	Size or complexity of automaton $\mathcal{B}_p$
$\text{score}(S, \mathcal{B}_p)$	Match score between sample set S and automaton $\mathcal{B}_p$
P_f	Inferred frequency
P_p	Inferred probability

2.1. DFFA Models

Since the agent cannot obtain the DPFA directly from the command, the DFFA needs to be defined first to make initial reasoning about the rules in the command. A DFFA $\mathcal{A}_f$ is a tuple, and $\mathcal{A}_f = \langle \Sigma_f, Q_f, I_{fr}, F_{fr}, \delta_{fr} \rangle$, where

- Σ_f represents the alphabet, including all possible input symbols that the automaton can process.
- Q_f denotes a finite set of states, containing all the possible states the automaton can occupy during its operation.
- $I_{fr} : Q_f \to \mathbb{N}$ is the initial-state frequency function; In a deterministic automaton, there is exactly one state q_l for which $I_{fr}(q) \neq 0$, indicating the state from which the system starts.
- $F_{fr} : Q_f \to \mathbb{N}$ is the final-state frequency function, describing the frequency at which the automaton terminates in each state.
- $\delta_{fr} : Q_f \times \Sigma_f \times Q_f \to \mathbb{N}$ is the transition frequency function, defining the frequency at which the automaton transitions from one state to another through a specific input symbol.
- $\delta_A : Q_f \times \Sigma_f \to Q_f$ is the associated transition function, which determines the unique path of state transitions in the automaton given a specific input.

The above DFFA satisfies frequency conservation if and only if, for any $q_u \in Q_f$, the initial frequency $I_{fr}(q)$ equals the sum of the frequencies of all possible transitions and termination states from state q_u. That is, for each state q_u, the following relation is satisfied:

$$I_{fr}(q_u) + \sum_{q'_u \in Q_f} \sum_{a \in \Sigma_f} \delta_{fr}(q'_u, a, q_u) = F_{fr}(q_u) + \sum_{q'_u \in Q_f} \sum_{a \in \Sigma_f} \delta_{fr}(q_u, a, q'_u). \tag{1}$$

Based on Equation (1), the components of the DFFA model are as follows:

- $a \in \Sigma_f$ represents the specific action performed during the activity; The transition from that state represents the frequency of transitioning to another state after the activity is performed.
- $q_\lambda \in Q_f$ denotes the initial state, as the initial condition of the activity, at which time no activity is performed.

- $q_u \in Q_f$ indicates the state of the agent at the current time.
- A transition $(q'_u, a, q_u) \in \delta_{A_f}$ from state q'_u and event a and activity A_f to reach state q_u.
- $F_{A_f}(q'_u, a, q_u)$ represents the frequency of observing event a and destination state q_u starting from state q'_u in activity A_f.
- $w = a_1 a_2 \dots a_i$ denotes a sequence of observed events, with its length (denoted as $|w|$) corresponding to the number of events in the sequence.
- $\theta = (q_\lambda, a_1, q_j, a_2, q_k, \dots, q_l, a_i, q_u)$ is a transition path for w in A_f, i.e., the sequence of transitions (q_λ, a_1, q_j), $(q_j, a_2, q_k), \dots, (q_l, a_i, q_u) \in \delta_{A_f}$ consistent with $w = a_1 a_2 \dots a_i$.

2.2. DPFA Models

The DPFA ensures system stability and predictability by combining deterministic state transitions with probabilistic distribution properties, and at the same time, it provides precise quantification for dealing with random events. A DPFA $\mathcal{B}_p$ is a tuple $\mathcal{B}_p = \langle \Sigma_p, Q_p, I_P, F_P, \delta_p \rangle$, where

- Q_p is a finite set of *states*, including all possible states (labeled as $q_1, \dots, q_{|Q|}$) the system may occupy.
- Σ_p is the alphabet, encompassing all possible input symbols that the automaton can process.
- $I_P : Q_p \to \mathbb{Q}_{\text{I}}{}^+ \cap [0, 1]$ is the initial-state probability function, representing the probability distribution for the system to start in each state.
- $F_P : Q_p \to \mathbb{Q}_{\text{I}}{}^+ \cap [0, 1]$ is the final-state probability function, describing the probability distribution for the system to terminate in each state.
- $\delta_P : Q_p \times (\Sigma \cup \{\lambda\}) \times Q_p \to \mathbb{Q}_{\text{I}}{}^+$ is a transition function, defining the probability of transitioning from one state to another given a specific input symbol; the function is complete: $\delta_P(q_v, a, q'_v) = 0$ indicates no transition from q_v to q'_v labeled with a. This paper also denotes (q_v, a, q', P_v) instead of $\delta_P(q_v, a, q'_v) = P$ where P stands for probability.

I_P, δ_P, and F_P are functions such that:

$$\sum_{q_v \in Q_p} \mathbb{I}_{\mathbb{P}}(q_v) = 1, \tag{2}$$

and

$$\forall q_v \in Q_p, \mathbb{F}_{\mathbb{P}}(q_v) + \sum_{a \in \Sigma_p \cup \{\lambda\}, q'_v \in Q_p} \delta_{\mathbb{P}}(q_v, a, q'_v) = 1. \tag{3}$$

Since the DPFA is derived from the DFFA, its specific components correspond directly to those of the DFFA and will not be detailed here. As depicted in Figure 2a, for the DPFA, all initial states, where q_λ and q_b serve as the transition states, satisfy the transition probability given by Equation (3). For q_λ, its probability is satisfied: $P_{q_\lambda} + P_a + P_b = \frac{1}{2} + \frac{1}{4} + \frac{1}{4} = 1$; For q_b, its probability is satisfied: $P_{q_b} + P_a + P_b = \frac{1}{2} + \frac{1}{6} + \frac{1}{3} = 1$. Similarly, as shown in Figure 2b, for the DFFA, all initial states, where q_λ and q_b serve as the transition states, satisfy the transition probability given by Equation (1). For q_λ, its frequency is satisfied: $F_\lambda + F_b = 65 + 10 = F_{q_\lambda} + F_a = 50 + 25$; For q_b, its frequency is satisfied: $F_{q_\lambda} + F_b = 15 + 10 = F_a = 25$.

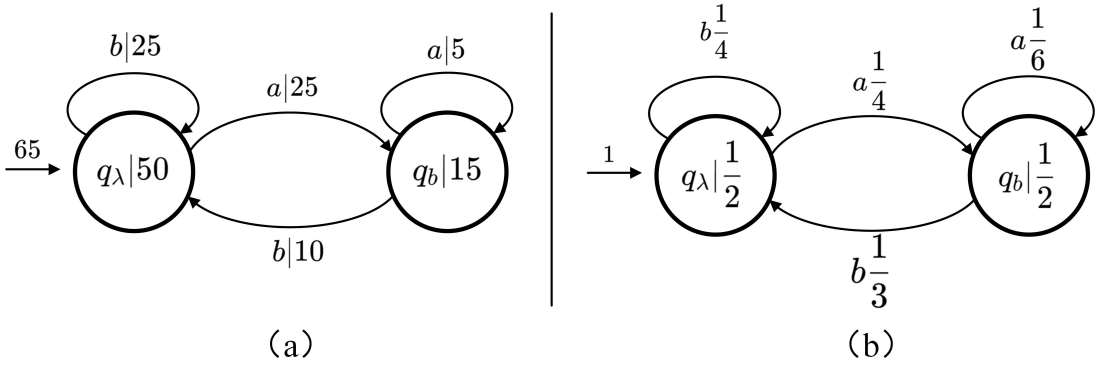

Figure 2. (**a**) State transitions of a DFFA; (**b**) State transitions of a DPFA.

2.3. Construction of DPFA from DFFA

The DFFA is constructed by manually giving the frequencies and calculating them. The DPFA is constructed by transforming it from DFFA, and this transformation involves calculating the frequencies of state transitions in the DFFA and normalizing them to obtain the corresponding probabilities, which are used to define the DPFA.

The detailed steps for the transformation process are as follows:

(1) Initialize DFFA: Start with the DFFA $\mathcal{A}_f = (\Sigma_f, Q_f, I_{fr}, F_{fr}, \delta_{fr}, \delta_{\mathcal{A}})$.

(2) Initialize frequency counts: For each state q in Q, initialize the frequency of state q in the array FREQ$[q]$ to store the frequency information for each state as follows:

$$\text{FREQ}[q] = F_{fr}(q). \tag{4}$$

(3) Update the frequencies and calculate the final probabilities: Iterate over all possible input symbols a. Accumulate the total frequency FREQ$[q]$ for state q. Then, calculate the probability distribution $F_{\mathcal{P}}(q)$ by dividing the initial frequency $F_{fr}(q)$ of state q by the accumulated total frequency FREQ$[q]$. This process is given by:

$$\text{FREQ}[q] = \text{FREQ}[q] + \delta_{fr}(q, a, \delta_{\mathcal{A}}(q, a)). \tag{5}$$

$$F_{\mathcal{P}}(q) = \frac{F_{fr}(q)}{\text{FREQ}[q]}. \tag{6}$$

(4) Calculate the transition probabilities: The frequency $\delta_{fr}(q, a, q')$ of transitioning from state q to state q' through input a is divided by the total frequency of state q FREQ$[q]$ to obtain the corresponding transition probability:

$$\delta_{\mathcal{P}}(q_f, a, q'_f) = \frac{\delta_{fr}(q_f, a, q'_f)}{\text{FREQ}[q]}. \tag{7}$$

When state q_f receives input a, the automaton transitions according to the original transition function δ_{A_f}. The state transition function δ_{A_f} is assigned to the new one $\delta_{\mathcal{B}_p}$ as follows:

$$\delta_P(q_v, a) = \delta_{\mathcal{A}}(q_f, a). \tag{8}$$

(5) Construct the DPFA: After computing all the necessary probabilities and updating the transition function $\mathcal{B}_p = (\Sigma_v, Q_v, q_\lambda, F_{\mathcal{P}}, \delta_{\mathcal{P}}, \delta_{\mathcal{B}_p})$, output the deterministic probabilistic finite automaton according to Equations (4)–(8).

3. Three Methods for Agents to Reason about Their Own Rules

3.1. Folding and Merging

All three methods for reasoning about agents' own rules involve two steps: Enumerate all states to form an initial automaton and merge compatible states in this initial automaton. The merging process is conducted by recursively combining two states q_f and q' and their substructures to simplify and optimize the overall automaton structure. After receiving input a, state q_f is replaced by the target state q', and the related state transition function δ_A and frequency information δ_{fr} are updated. The main modification involves changing the transition target from q' to q for all states q_f and adjusting the frequency value $n = \delta_{fr}(q_f, a, q')$. At last, the substructures of q and q' are merged by iterating over all possible input symbols $a \in \Sigma_f$. If there is already a transition for state q' for a given input a, then the corresponding subtrees are merged, the frequency of both is increased, and the transformation relation is updated. The relationship between folding and merging and their respective processes are demonstrated in Figure 3. The details of the merging process are as follows:

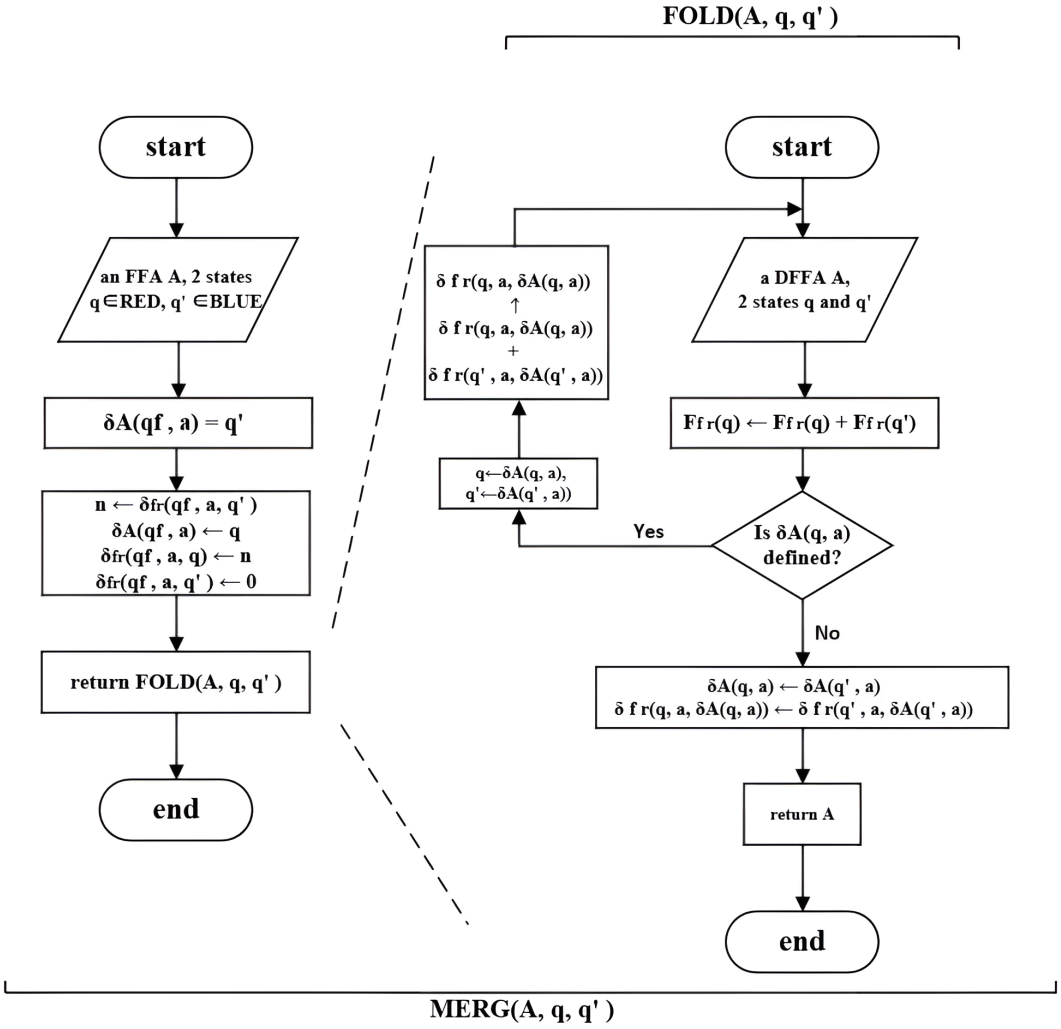

Figure 3. The folding and merging process.

First, given a finite state automaton $\mathcal{A}$ and two states q and q', where $q \in$ RED and $q' \in$ BLUE, consider a state q_f and an input a such that:

$$\delta_A(q_f, a) = q'. \tag{9}$$

In this case, let n be the value of the frequency transition function δ_{fr}:

$$n = \delta_{fr}(q_f, a, q'). \tag{10}$$

Then, the state transition function δ_A is updated by changing the target state from q' to q when q_f receives the input a:

$$\delta_A(q_f, a) = q. \tag{11}$$

Meanwhile, the frequency transition function δ_{fr} is updated by assigning the value n to the pair (q_f, a, q):

$$\delta_{fr}(q_f, a, q) = n. \tag{12}$$

and assigning $\delta_{fr}(q_f, a, q')$ to 0:

$$\delta_{fr}(q_f, a, q') = 0. \tag{13}$$

Folding is utilized to optimize the state transition system of an automaton by recursively merging the subtree structures of two states, q and q'. The detailed process is as follows:

First, the frequency distributions of states q and q' are merged according to the following equation:

$$F_{fr}(q) = F_{fr}(q) + F_{fr}(q'). \tag{14}$$

Then, iterate over all possible input symbols $a \in \Sigma_f$. For each input symbol a that the automaton can receive, determine whether there is a defined state transition $\delta_A(q', a)$ for state q' upon receiving a. If $\delta_A(q', a)$ exists, further determine whether there is a defined state transition $\delta_A(q, a)$ for state q upon receiving the same input a. If there is a transition for state q, the algorithm merges the frequency information of the two states' transitions according to the following equation:

$$\delta_{fr}(q, a, \delta_A(q, a)) = \delta_{fr}(q, a, \delta_A(q, a)) + \delta_{fr}(q', a, \delta_A(q', a)). \tag{15}$$

This step guarantees that the merged state can accurately reflect the transition frequencies of both original states q and q' while preserving the transition logic under input a.

Subsequently, the algorithm continues to recursively merge the subtree structures pointed to by $\delta_A(q, a)$ and $\delta_A(q', a)$, thereby ensuring the integrity and correctness of the entire structure.

If $\delta_A(q, a)$ is not yet defined, the algorithm directly assigns the state transition $\delta_A(q', a)$ from state q' to the corresponding transition for state q upon receiving input a, and then it updates the associated transition frequency as follows:

$$\delta_A(q, a) = \delta_A(q', a). \tag{16}$$

$$\delta_{fr}(q, a, \delta_A(q, a)) = \delta_{fr}(q', a, \delta_A(q', a)). \tag{17}$$

Finally, through Equations (14) and (17), the merging of states q and q' is completed for all input symbols a, and the state transition functions and frequency information of the automaton are updated accordingly.

3.2. ALERGIA

ALERGIA is a probabilistic inference method for inferring DFFA from limited data samples by exploiting state frequency similarities. This method has been widely used in fields such as natural language processing, speech recognition, and bioinformatics, and it is employed to infer automata models that can identify underlying rules from limited data, thus improving the precision of data modeling. The core idea of ALERGIA is to evaluate the frequency differences between two states for all input symbols and determine their compatibility based on a predefined threshold. If the states are highly similar, they are merged to simplify and optimize the structure of the automaton. This process is carried out incrementally over the sample set to ensure that the automaton can accurately capture the statistical properties of the data. The specific steps are as follows:

(1) Frequency proximity test: Given a finite-state automaton $\mathcal{A}$, along with the frequencies f_1, f_2, and their corresponding sample sizes n_1, n_2, the process is designed to check whether the two frequencies are sufficiently similar. First, the difference between the frequencies, denoted as γ, is computed as follows:

$$\gamma = \left| \frac{f_1}{n_1} - \frac{f_2}{n_2} \right|. \tag{18}$$

γ is compared with a threshold value to check whether the frequencies are sufficiently similar.

$$\gamma < (\sqrt{\frac{1}{n_1}} + \sqrt{\frac{1}{n_2}}) \cdot \sqrt{\frac{1}{2} \ln \frac{2}{a}}. \tag{19}$$

If γ is less than this threshold, then $\frac{f_1}{n_1}$ and $\frac{f_2}{n_2}$ are considered sufficiently similar, and a boolean value is returned to indicate this result.

(2) State compatibility test: The process begins with a finite-state automaton $\mathcal{A}$ and two states, q_u and q_v, that need to be compared. First, these states are assumed to be compatible, and the variable $Correct$ is initialized to true.

Then, a frequency proximity test is conducted to compare the overall frequency distributions of q_u and q_v. If this test confirms significant differences between the frequency distributions of the two states, $Correct$ is set to false to indicate that the states are incompatible.

Meanwhile, the process checks the state transition frequencies for each input symbol $a \in \Sigma$. If the transition frequencies between q_u and q_v are incompatible for any input symbol a, then $Correct$ is again set to false.

Finally, the value of $Correct$ is returned to indicate whether q_u and q_v are compatible based on the comparison result.

(3) Finite-state automaton construction: In the iterative process, a state q_b with a frequency greater than or equal to t_0 is chosen from the set blue. For each iteration, it is determined whether there exists a state q_r in the set red such that q_r and q_b demonstrate compatibility in the frequency proximity test. If q_r and q_b are compatible, they are merged; otherwise, q_b is added to the set red. After each iteration, the set blue is updated by eliminating the processed state and adding prefix states that are different from those in the set red. This process continues until no further merging operations can be performed. Finally, by iteratively adjusting the states in the sets red and blue, a new finite-state automaton $\mathcal{A}$ is constructed and returned, which can perform effectively on the given sample set.

3.3. DSAI

The DSAI iteratively compares and merges the transition probabilities of two states to construct a finite-state structure optimized for a given sample set. The core idea of this method is to compare the probability differences between states to determine whether they are sufficiently similar to be merged into a single state. If their behaviors are highly similar, these states are merged to simplify the structure; otherwise, they are retained in different sets for further examination and optimization. This process is repeated iteratively, and finally, a system that is statistically significant and structurally optimized is obtained. The specific steps are as follows:

First, for any input sequence $x \in \Sigma_f$, the transition probabilities $Pr_{\mathcal{A},q}(x)$ and $Pr_{\mathcal{A},q'}(x)$ for states q and q' under input x are calculated. The difference between these probabilities should be evaluated to determine its significance, and it is computed as follows:

$$\gamma = \left| Pr_{\mathcal{A},q}(x) - Pr_{\mathcal{A},q'}(x) \right|. \tag{20}$$

Next, this difference γ is compared with a predefined threshold $\frac{\mu}{2}$:

$$\gamma > \frac{\mu}{2}. \tag{21}$$

If the above inequality holds, it suggests that there is a significant behavioral difference between states q and q' under input sequence x and that the transition characteristics of states q and q' are incompatible. In this case, the algorithm should immediately return a result of incompatibility.

The next step is to further verify the compatibility of states q and q' for specific input sequences. Particularly, for each input sequence x belonging to the set $\text{mps}(\mathcal{A}qr, k+1)$, the transition probability $Pr\mathcal{A}, q'(x)$ for state q' under input x is calculated. Again, this transition probability is compared with $\frac{\mu}{2}$:

$$Pr_{\mathcal{A},q'}(x) > \frac{\mu}{2}. \tag{22}$$

If this inequality holds, it indicates that the transition probability of state q' under the specific input sequence x far exceeds the threshold. Therefore, states q and q' are considered incompatible, and the algorithm should immediately return a result of incompatibility.

If none of the inequalities hold for all input sequences and specific input sequences, it indicates that all differences do not exceed the allowable threshold, then states q and q' can be considered compatible. In this case, the algorithm should return a result of compatibility.

This method for compatibility determination realizes precise quantification of behavioral differences between states through rigorous mathematical analysis, providing a reliable basis for further merging states and optimizing automaton structures. It not only considers the overall transition probability differences but also pays particular attention to behaviors under critical input sequences, thereby ensuring the accuracy and comprehensiveness of the determination.

3.4. MDI

The MDI iteratively merges states in a probabilistic automaton by comparing their transition probabilities to construct an optimized finite-state structure for the given data sample. Its core idea is to calculate a match score after each potential state merge to measure how well the merged states fit the data, and then it compares this score with a predefined threshold to determine their compatibility. If the score does not exceed this threshold, the states are merged to reduce the automaton's complexity while maintaining its accuracy in modeling the data. This process continues until no compatible states can be merged. MDI is particularly useful for applications where balancing

model complexity with data fidelity is crucial, such as in language modeling, system diagnostics, and probabilistic reasoning, because it ensures that the resulting automaton can deal with uncertainty both efficiently and robustly. The specific steps are as follows:

First, given a finite-state automaton $\mathcal{B}_p$, the threshold α is calculated from the sample set S, and it is employed to evaluate the statistical significance of frequencies. Then, the prefix tree acceptor is constructed from the sample set S, and two state sets, red and blue, are initialized, where the former contains the initial state q_λ and the latter contains all prefix states in the sample set.

Next, a state q_b with a frequency greater than or equal to α is chosen from the blue set, and it is determined whether there exists a state q_r in the red set such that these two states are compatible in the match score test. If q_r and q_b are compatible, they are merged into a new automaton $\mathcal{B}_p$, and the match score $(S, \mathcal{B}_p)$ of the merged automaton with the sample set S is calculated. Subsequently, this score is compared with the threshold α:

$$\text{score}(S, \mathcal{B}_p) < \alpha. \tag{23}$$

$$\text{score}(S, \mathcal{B}_p) = \frac{\sum_{w \in S} \text{cnt}_S(w) \log \Pr_{\mathcal{B}}(w)}{\|\mathcal{B}_p\|}. \tag{24}$$

where:

- S: The sample of strings is being analyzed.
- $\mathcal{B}_p$: The automaton is being evaluated.
- $w \in S$: A string in the sample S.
- $\text{cnt}_S(w)$: The number of times the string w appears in the sample S.
- $\Pr_{\mathcal{B}}(w)$: The probability assigned by the automaton $\mathcal{B}_p$ to the string w.
- $\|\mathcal{B}_p\|$: The size or complexity of the automaton $\mathcal{B}_p$. This can refer to the number of states, transitions, or parameters in a probabilistic automaton.

The score function is utilized to evaluate how well the automaton $\mathcal{B}_p$ models the sample S. Specifically, it sums over all strings in the sample, weighted by their frequency $\text{cnt}_S(w)$ and the logarithm of the probability $\log \Pr_A(w)$ assigned by the automaton. Then, this sum is normalized by the size of the automaton $\|\mathcal{B}_p\|$ to strike a balance between goodness-of-fit and model complexity.

If the match score is less than the threshold, q and q' are considered compatible; otherwise, q_b is added to the red set to indicate that the states are incompatible.

Finally, the blue set is updated by eliminating the processed state and adding all prefix states that are different from those in the red set. This process continues repeatedly until no further merging operations can be performed. Finally, the optimized finite-state automaton $\mathcal{B}_p$ that can perform best on the given sample set is returned.

4. Results

In this paper, a reconnaissance aircraft follower in the leader-following structure is taken as an example to demonstrate how followers infer their own rules. As illustrated in Figure 4, the follower's rules are defined by an automaton. First, the leader knows these rules, but the followers do not. The leader sends several commands to the follower according to the predefined rules, and the follower records the commands it receives.

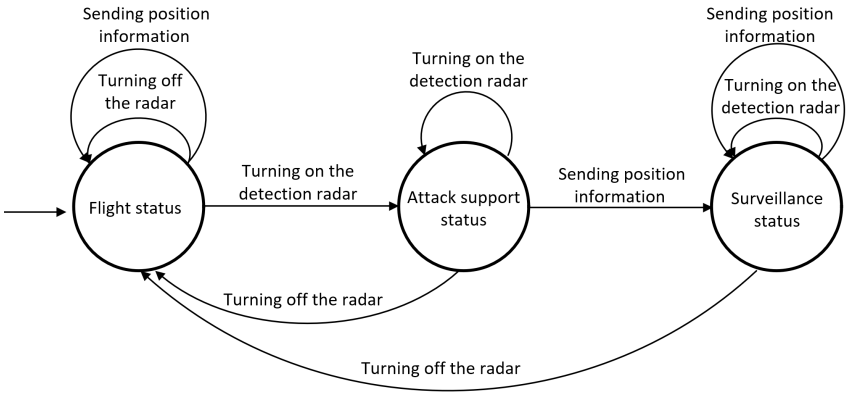

Figure 4. State transitions of the scout.

As listed in Table 2, the follower calculates the frequency of each command sent by the leader, where a stands for turning on the detection radar, b stands for turning off the radar, c denotes sending position information, q_λ represents the flight status, q_a stands for the attack support status, and q_{ac} denotes the surveillance status. Based on these command frequencies and using a combination of merging, folding, and the three different inference methods, the follower can infer a frequency automaton, as shown in Figure 5.

Subsequently, the frequency automaton is utilized to construct a probabilistic automaton representing the follower's rules, as demonstrated in Figure 5. To sum up, the follower can infer the rules it follows within the system by analyzing the commands sent by the leader.

This paper takes a reconnaissance drone as an example to demonstrate how a follower in a leader-following structure infers its operational rules. As demonstrated in Figure 4, the follower's rules are defined by an automaton. First, these rules are known to the leader but unknown to the follower. The leader sends several commands to the follower according to predefined rules, and the follower records these commands.

As listed in Table 2, the follower calculates the frequency of each command sent by the leader. Here, a denotes activating the detection radar, b denotes deactivating the radar, c represents transmitting location information, q_λ stands for the flight state, q_a denotes the attack support state, and q_{ac} represents the surveillance state. Based on these command frequencies, the follower can infer a frequency automaton by using a combination of merging, folding, and the three distinct inference methods, as illustrated in Figure 5.

Table 2. The frequency of each order given by the leader.

Action	Frequency	Action	Frequency	Action	Frequency
λ	3730	aab	60	acba	10
a	1370	aac	220	acbb	20
b	550	aba	130	acbc	40
c	550	abb	50	acca	100
aa	340	abc	60	accb	20
ab	260	aca	400	accc	90
ac	750	acb	80	baac	20
ba	200	acc	250	baba	10
bb	60	baa	10	babb	10
bc	80	bab	40	baca	40
ca	300	bac	150	bbaa	10
cb	70	bba	30	bbab	10
cc	100	bbb	0	bcab	10
aaa	40	bbc	10	bcac	10
aba	130	bca	30	bcba	10
abb	50	bcb	20	bccc	10
abc	60	bcc	20	caaa	10
aca	400	caa	30	caab	10
acb	80	cab	20	cac	190
aca	400	cac	190	caca	50
acb	80	cba	20	cacb	10
aacc	70	cbb	20	cacc	60
aaaa	10	cbc	10	cbac	10
aaab	10	cca	60	cbba	10
aaac	20	ccb	10	ccac	20
aaba	30	abca	20		
aabc	20	abcb	10		

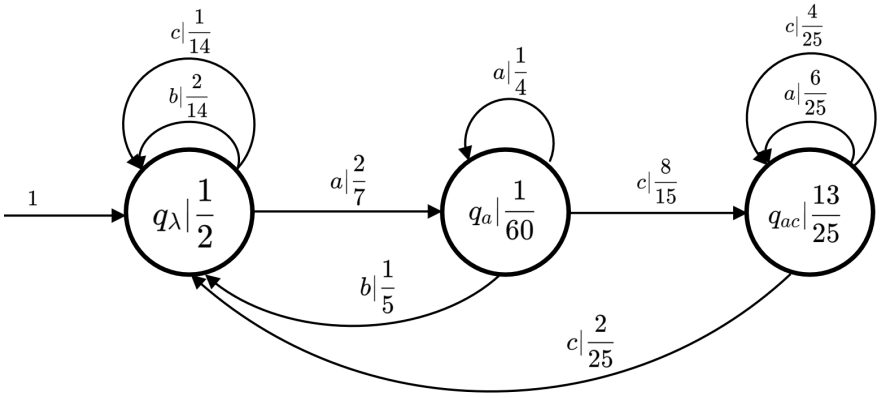

Figure 5. DPFA state transition, where the score in the circle represents the possibility of staying in this state, and the score on the curve represents the possibility of performing the action to realize the state transition.

5. Simulation

This section provides a simulation example to verify the effectiveness of the aforementioned theoretical results.

Figures 6–9 show the results produced by the initial automaton constructed based on Table 2. The frequencies corresponding to each action are provided, and the frequency for each state is calculable from the actions associated with that state. For instance, the frequency of staying in state q_λ in Figure 6 is $F_{q\lambda} = F_\lambda - F_a - F_b - F_c = 3730 - 1370 - 550 - 550 = 1260$, the frequency of staying in state q_a in Figure 7 is $F_{qa} = F_a - F_{aa} - F_{ab} - F_{ac} = 1370 - 340 - 260 - 750 = 20$, the frequency of staying in state q_b in Figure 8 is $550 - 200 - 60 - 80 = 210$, and the frequency of staying in state q_c in Figure 9 is $F_{qb} = F_b - F_{ba} - F_{bb} - F_{bc} = 550 - 300 - 70 - 100 = 80$. Following the methodology introduced in the previous section, the final probabilistic automaton representing the follower's rules is inferred through folding and applying the three inference methods, as demonstrated in Figure 10.

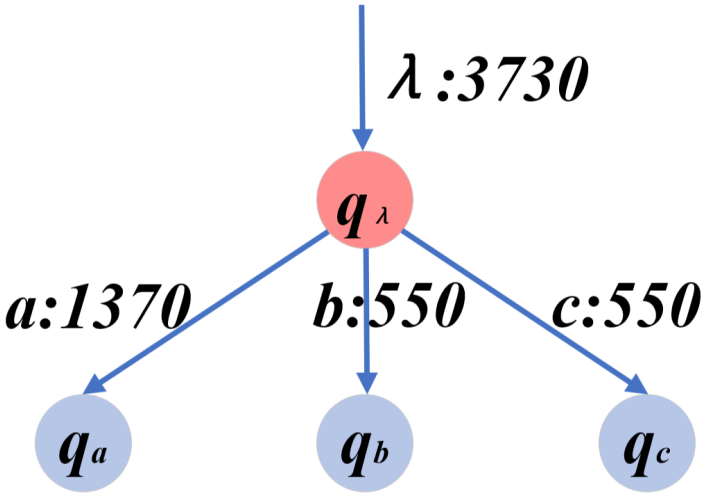

Figure 6. State transitions of q_λ.

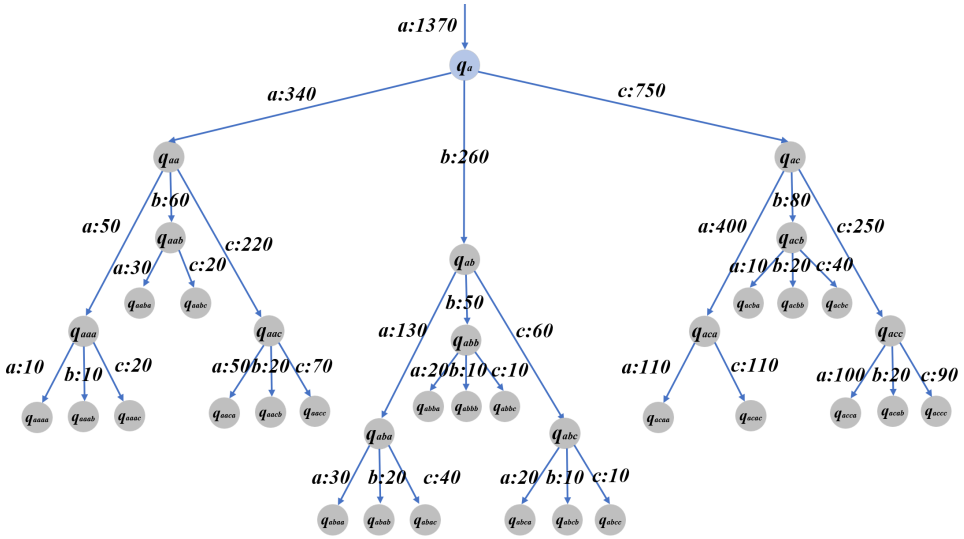

Figure 7. State transitions of q_a.

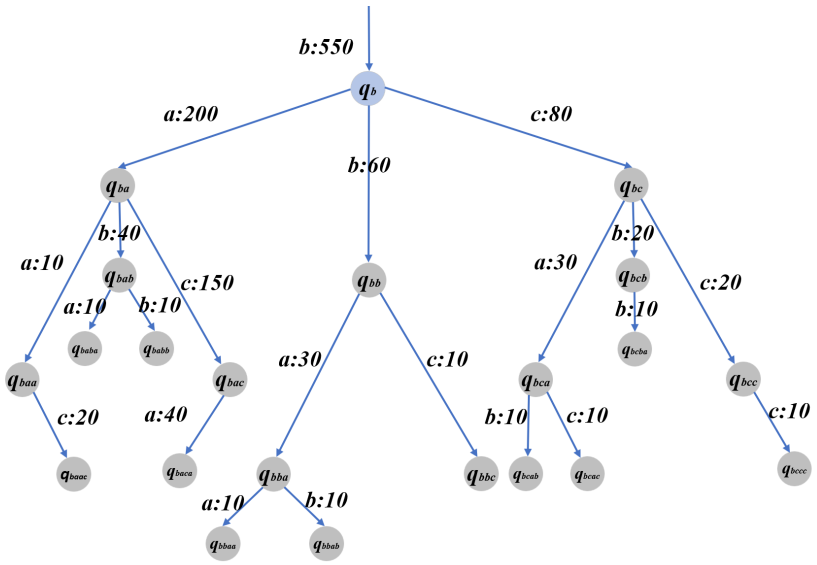

Figure 8. The state transitions of q_b.

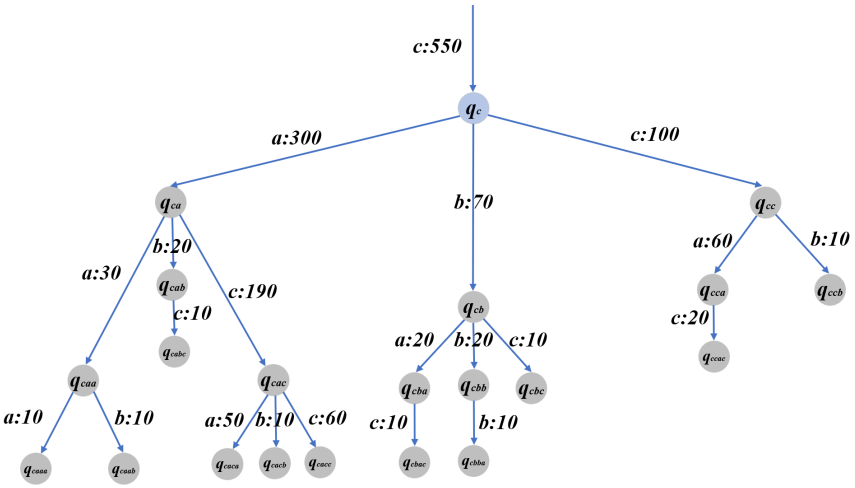

Figure 9. The state transitions of q_c.

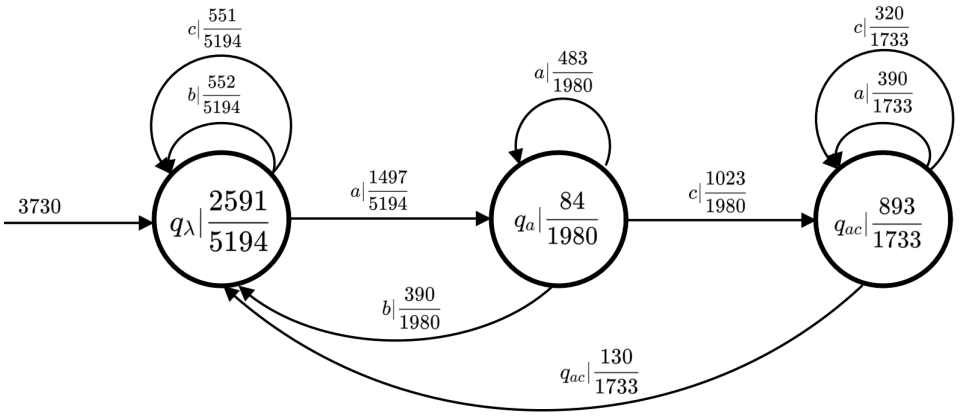

Figure 10. The deterministic probabilistic finite state automaton reasoned by ALERGIA, DSAI, and MDI.

In this study, the inference accuracy is defined as the absolute difference between the inferred frequency of the corresponding state and the inferred probability:

$$(1- \mid P_f - P_p \mid) \times 100\%. \tag{25}$$

Using Equation (25) and the three algorithms designed in the previous section, the results listed in Table 3 are obtained. Owing to a large amount of experimental data, it can be observed from Table 3 that the final inference accuracy for each state and action is very high. The calculation of the average value for each state and action indicates that the final DPFA inference accuracy is 98.535%.

Table 3. The error and accuracy of DFFA and DPFA in different states.

State and Action	Probability of DFFA	Probability of DPFA	Probability Error	Accuracy
$q\lambda$	2591/5194	1/2	−0.001155	99.88%
qa	84/1980	1/60	0.025758	97.42%
qac	893/1733	13/25	−0.004709	99.53%
$q\lambda\|a$	1497/5194	2/7	0.002503	99.75%
$q\lambda\|b$	552/5194	2/14	−0.036581	96.34%
$q\lambda\|c$	551/5194	1/14	0.034655	96.53%
$qa\|a$	483/1980	1/4	−0.006061	99.39%
$qa\|b$	390/1980	1/5	−0.003030	99.70%
$qa\|c$	1023/1980	8/15	−0.016667	98.33%
$qac\|a$	390/1733	6/25	−0.014957	98.50%
$qac\|b$	130/1733	2/25	−0.004986	99.50%
$qac\|c$	320/1733	4/25	0.024651	97.53%

When merging states, if there is excessively high similarity between two states, it may hinder the learning of a correct probabilistic automaton, and if the similarity is excessively low, the states cannot be properly merged. After multiple attempts, the parameter α for ALERGIA is set to 0.05, $\frac{\mu}{2}$ for DSAI is set to 0.1, and α for MDI is set to 0.15. For commands with frequencies lower than 30, merging them into a state after calculation has the least impact and will consume a significant amount of computational resources, so they are directly merged with the previous state without further computation. The inference process of the three different methods is depicted in Figure 11. Overall, all three methods achieve an accuracy of 98.535% within 6 s, and ALERGIA runs slightly faster than DSAI and MDI. For the ALERGIA method, its accuracy increases in several short bursts, and this is because, when calculating the similarity between two states, the states to merge include both the current state and the subsequent state that can be reached from it. Therefore, when two states are determined to be mergeable, multiple states will actually be merged, resulting in a sharp improvement in the inferred automaton's accuracy. For the DSAI method, the accuracy increases evenly in a wave-like pattern over time. This is because this method determines whether there is sufficient similarity between actions to be merged, so it sequentially computes whether each action meets the merging condition, leading to a steady increase in accuracy. For the MDI method, the accuracy first rises

slowly and then faster. This is because, at the beginning of the inference process, the score between the two states includes information about the automaton's states, action transition functions, etc. At first, the automaton contains much information and takes a longer time to compute, so fewer states are merged, and there is a slower increase in accuracy. Over time, more states are merged, the automaton's computation time decreases, and its accuracy rises quickly. Towards the end, the improvement in accuracy slows down, and this is because merging additional states has little impact on accuracy. The comparison of ALERGIA, DSAI, and MDI methods in terms of accuracy and speed of increase is presented in Table 4.

Table 4. Comparison of ALERGIA, DSAI, and MDI methods.

Method	Accuracy Trend	Speed of Increase
ALERGIA	Sharp increase, then stabilizes	Fastest
DSAI	Steady rise with fluctuations	Medium
MDI	Slow initially, faster later	Slow initially, then accelerates

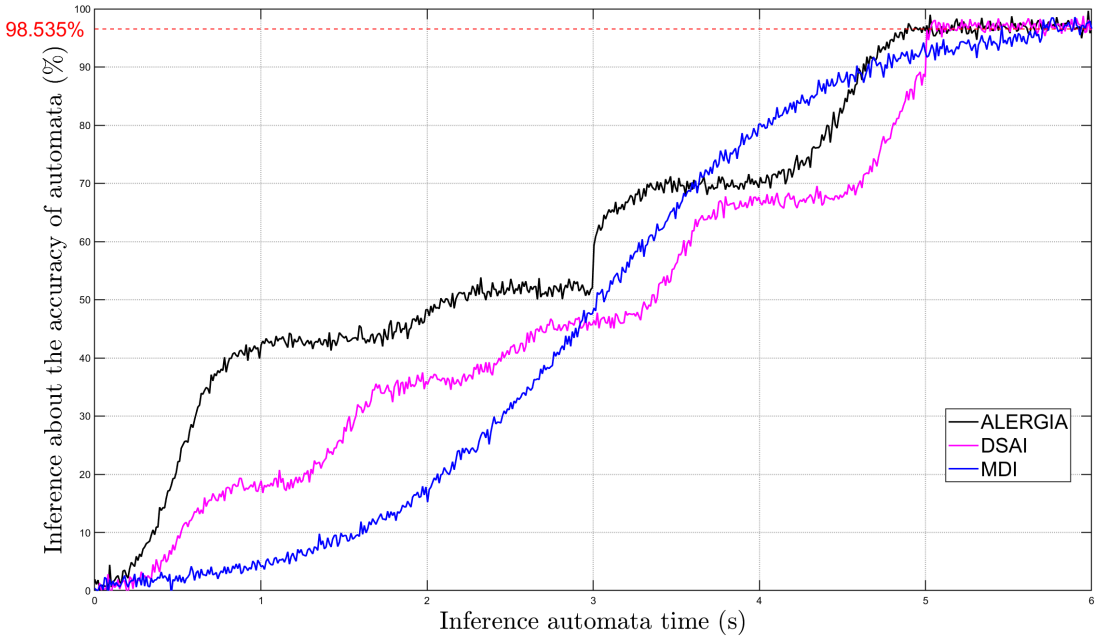

Figure 11. Inference accuracy over time for three methods.

6. Discussion

This study addresses the issue of maintaining the functionality of a MAS in the absence of a leader. Traditional leader-following models assume that the leader can always function but neglect the risk of system paralysis when the leader is attacked. Through an analysis of experimental data, it is found that probabilistic automata inference methods (ALERGIA, DSA, and MDI) effectively enable followers to independently infer operating rules. For instance, followers can employ the ALERGIA method to infer their own rules and continue executing tasks without leader commands, demonstrating a high level of self-recovery.

Compared to previous studies, this research utilizes probabilistic automata to improve the autonomy and robustness of systems in the absence of a leader. This method is different from earlier work that proposed four state feedback protocols based on adaptive control and H_∞ control to manage sensor and actuator failures in leader-following MASs, without considering total leader-missing attacks [11]. Meanwhile, the findings differ from those of other studies, which developed a distributed observer and a proportional-integral control strategy to resolve control problems in leader-following multi-Euler-Lagrange systems under communication link failures and external disturbances [12]. Unlike these studies, this research illustrates how followers can maintain actions without a leader by inferring operating rules from historical command data, providing a novel solution for improving the autonomy and adaptability of MASs.

One limitation of this research is that all inference methods assume that followers can access and process the leader's historical command data in a timely manner. In extreme cases where the communication between followers and the leader is completely broken down, the effectiveness of these methods may be greatly reduced. However,

this research demonstrates that probabilistic automata methods can maintain system continuity and effectiveness even in the absence of a leader. These findings provide new perspectives for future theoretical research and valuable guidance for practical system design.

7. Conclusions

This study proposes three probabilistic automata inference methods, namely ALERGIA, DSAI, and MDI, and compares their efficiency in rule-based reasoning. The construction of a probabilistic finite state automaton improves the system's ability to manage uncertainty and its stability in dynamic environments. The three methods enable followers to autonomously infer and execute operational rules, thereby maintaining system continuity even under leader-missing attacks. Comparative analysis indicates that ALERGIA runs slightly faster than DSAI and MDI, providing a robust basis for selecting the optimal method in practical applications. The results suggest that even in the absence of a leader, followers can independently infer their operational rules, ensuring the continuity and effectiveness of MASs. Future research will investigate integrating adaptive learning mechanisms, such as reinforcement learning or neural networks, into the leader-follower model to improve decision-making and further enhance the resilience and autonomy of MASs.

Author Contributions

K.W.: data curation, writing—original draft preparation; X.G.: conceptualization, methodology, software. All authors have read and agreed to the published version of the manuscript.

Funding

This work was supported by the National Natural Science Foundation of China under Grants 62003374 and 62403128, the Fundamental Research Funds for the Central Universities under Grant 08002150138, the Hunan Provincial Natural Science Foundation of China under Grant 2020JJ5765, the Open Foundation of Key Laboratory of Cyberspace Security, Ministry of Education (No. KLCS20240403), the Jiangsu Provincial Natural Science Foundation of China under Grants No. BK20210223 and BK20241284, the Nanjing Science and Technology Innovation Project for Overseas Scholars under Grant 4209012304, and Start-up Research Fund of Southeast University under Grant RF1028623260.

Institutional Review Board Statement

Not applicable.

Informed Consent Statement

Not applicable.

Data Availability Statement

Not applicable.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Kim, K.-D.; Kumar, P.R. Cyber–physical systems: A perspective at the centennial. *Proc. IEEE* **2012**, *100*, 1287–1308.
2. González-Briones, A.; De La Prieta, F.; Mohamad, M.S.; et al. Multi-agent systems applications in energy optimization problems: A state-of-the-art review. *Energies* **2018**, *11*, 1928.
3. Sharma, M.K.; Zappone, A.; Assaad, M.; et al. Distributed power control for large energy harvesting networks: A multi-agent deep reinforcement learning approach. *IEEE Trans. Cogn. Commun. Netw.* **2019**, *5*, 1140–1154.
4. Iqbal, S.; Altaf, W.; Aslam, M.; et al. Application of intelligent agents in health-care. *Artif. Intell. Rev.* **2016**, *46*, 83–112.
5. Radhakrishnan, B.M.; Srinivasan, D. A multi-agent based distributed energy management scheme for smart grid applications. *Energy* **2016**, *103*, 192–204.
6. Karydis, K.; Kannappan, P.; Tanner, H.G.; et al. Resilience through learning in multi-agent cyber-physical systems. *Front. Robot. AI* **2016**, *3*, 36.
7. Khaitan, S.K.; McCalley, J.D. Design techniques and applications of cyberphysical systems: A survey. *IEEE Syst. J.* **2014**, *9*, 350–365.

8. Mahela, O.P.; Khosravy, M.; Gupta, N.; et al. Comprehensive overview of multi-agent systems for controlling smart grids. *CSEE J. Power Energy Syst.* **2020**, *8*, 115–131.
9. Jiao, W.; Sun, Y. Self-adaptation of multi-agent systems in dynamic environments based on experience exchanges. *J. Syst. Softw.* **2016**, *122*, 165–179.
10. Binyamin, S.S.; Ben Slama, S. Multi-agent Systems for Resource Allocation and Scheduling in a smart grid. *Sensors* **2022**, *22*, 8099.
11. Chen, C.; Lewis, F.L.; Xie, S.; et al. Resilient adaptive and H_∞ controls of multi-agent systems under sensor and actuator faults. *Automatica* **2019**, *102*, 19–26.
12. Long, M.; Su, H.; Zeng, Z. Distributed observer-based leader–follower consensus of multiple Euler–Lagrange systems. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *35*, 157–168.
13. Gong, X.; Li, X.; Shu, Z.; et al. Resilient output formation-tracking of heterogeneous multiagent systems against general Byzantine attacks: A twin-layer approach. *IEEE Trans. Cybern.* **2023**, *54*, 2566–2578.
14. Baxevani, K.; Zehfroosh, A.; Tanner, H.G. Resilient Supervisory Multiagent Systems. *IEEE Trans. Robot.* **2021**, *38*, 229–243.
15. Mahfouz, M.; Hafez, A.T.; Ashry, M.M.; et al. Cyclic leader-following Strategy For Cooperative Unmanned Aerial Vehicles. In Proceedings of the IEEE International Conference on Vehicular Electronics and Safety, Cairo, Egypt, 4–6 September 2019; IEEE: New York, NY, USA, 2019; pp. 1–6.
16. Wei, Z.; Zhang, X.; Zhang, Y.; et al. Weighted automata extraction and explanation of recurrent neural networks for natural language tasks. *J. Log. Algebr. Methods Program.* **2024**, *136*, 100907.
17. Bates, M. Models of natural language understanding. *Proc. Natl. Acad. Sci. USA* **1995**, *92*, 9977–9982.
18. Collins, M. Head-driven statistical models for natural language parsing. *Comput. Linguist.* **2003**, *29*, 589–637.
19. Maletti, A. Survey: Finite-state technology in natural language processing. *Theor. Comput. Sci.* **2017**, *679*, 2–17.
20. Jaf, S.; Calder, C. Deep learning for natural language parsing. *IEEE Access* **2019**, *7*, 131363–131373.
21. Carrasco, R.C.; Oncina, J. Learning stochastic regular grammars by means of a state merging method. In Proceedings of the International Colloquium on Grammatical Inference, Alicante, Spain, 21–23 September 1994; Springer: Berlin/Heidelberg, Germany, 1994; pp. 139–152.