Ingénierie des Systèmes d'Information
Vol. 30, No. 9, September, 2025, pp. 2233-2252
Journal homepage: http://iieta.org/journals/isi

A Mutation Analysis-Based Approach for Testing Coordination Mechanisms in Multi-Agent Systems: Application to Air Traffic Control Systems

Soufiene Boukelloul[1*], Nour El Houda Dehimi[2], Stéphane Galland[2,3], Sofiane Zaidi[4]

[1] LIAOA Laboratory, Department of Mathematics and Computer Science, University of Oum El Bouaghi, Oum El Bouaghi 04000, Algeria
[2] Université de Technologie de Belfort Montbéliard, UTBM, CIAD UR 7533, Belfort 90010, France
[3] Université Bourgogne Europe, UBE, CIAD UR 7533, Dijon 21000, France
[4] Department of Mathematics and Computer Science, Research Laboratory on Computer Science's Complex Systems (RELA(CS)2), University of Oum El Bouaghi, Oum El Bouaghi 04000, Algeria

Corresponding Author Email: soufiene.boukelloul@univ-oeb.dz

https://doi.org/10.18280/isi.300903

Received: 19 July 2025
Revised: 15 September 2025
Accepted: 21 September 2025
Available online: 30 September 2025

Keywords:
multi-agent systems, system level testing, mutation analysis, parallel genetic algorithms, test case generation, coordination mechanisms, conflict detection, air traffic control

ABSTRACT

In this work, we propose an innovative approach that combines mutation analysis with parallel genetic algorithms to evaluate coordination mechanisms in multi-agent systems (MAS), particularly in critical contexts where effective conflict management is essential. The approach involves temporarily disabling coordination, injecting simulated errors (mutants) into concurrent access to shared resources, and then automatically generating test cases that intensify potential conflicts. Coordination is then reactivated to observe whether it successfully resolves these conflicts, thereby assessing its robustness. The approach is applied to a concrete case study, the En-route air traffic control system (EATC), in order to test the ability of the adopted coordination mechanism to manage trajectory conflicts between aircraft, while ensuring compliance with minimum separation standards (vertical, lateral, and longitudinal). The experimental results validate the effectiveness of the proposed approach. Indeed, after 150 generations, the best test case quality achieved an average of 97.6064%, with a maximum fitness of 98.1281%. Among 428 conflict cases, the adopted coordination mechanism successfully resolved 417 (97.43%), failing in only 2.57% of the cases. These outcomes underscore both the robustness and scalability of the proposed approach for testing MAS coordination in critical domains.

1. INTRODUCTION

Multi-agent systems (MAS) [1-3] consist of autonomous and self-organizing agents that collaborate within a shared environment to achieve individual or shared goals. These systems have widespread applications in such vital domains as cooperative robotics [4, 5], intelligent transportation systems [6], control of energy grid [7, 8], financial markets [9], and social simulation [10], where effective coordination among agents is required to ensure optimal performance and harmonious coexistence with minimal conflicts. In an MAS, conflict arises when a group of agents pursues incompatible goals, executes conflicting actions, or competes for access to limited and non-shareable resources. Such conflicts are common in distributed and dynamic environments and represent a major danger for the proper functioning of the system [11]. Their occurrence reveals the MAS's actual ability to manage agent interactions and maintain overall system coherence despite interference situations. However, in dynamic and competitive environments [12, 13], these conflicts may emerge due to various factors, including competition for scarce resources, divergence of agents' objectives, disagreements in decision-making, or uncertainties related to complex agent interactions. These conflicts can significantly impact the system's stability and efficiency, leading to deadlocks, operational inefficiencies, or even breakdowns in cooperation among agents. Therefore, the ability of a coordination mechanism to handle such conflicts turns into a critical parameter for the robustness and responsiveness of MAS. Several coordination approaches have been put forward, such as collective planning [14, 15], negotiation, auctions [16-18], and resource allocation protocols [19]. However, it is a challenging task to analyze these mechanisms as conflicts tend to occur under unanticipated and varying situations. Traditional methods rely primarily on scenario simulation and performance measurement with tangible parameters such as conflict resolution time [20], success rate of negotiations [21, 22] and resource optimization [23]. However, traditional methods have their limitations: they cannot always anticipate failure of coordination mechanisms while facing unexpected disturbances or identifying the structural vulnerabilities of the system. A hidden failure can undermine a MAS's capacity to solve conflicts in an effective manner, to the point that its overall performance and reliability are negatively affected. In order to surpass these challenges, we suggest applying an

approach based on mutation analysis [24-27], a software testing-derived technique [28-32] that entails introducing controlled perturbations into the system under test by design. By seeing how the system reacts to these disruptions, one can identify instances where coordination mechanisms malfunction or lose effectiveness in managing conflicts. This approach thus enables the identification of weaknesses in existing strategies and suggests adjustments to enhance their robustness. Unlike traditional approaches, mutation analysis provides a more rigorous and objective evaluation framework capable of testing coordination mechanisms under adverse and unpredictable conditions [33]. The main objective of this study is to propose a systematic and reproducible approach for evaluating and improving conflict management in MAS by identifying vulnerabilities in coordination mechanisms and suggesting adjustments to optimize their resilience. Our approach also allows for an objective comparison of different coordination strategies and guides the design of more efficient and adaptive mechanisms. Thus, mutation analysis represents a significant advancement in the evaluation and enhancement of MAS, making them more effective and robust in addressing the challenges posed by dynamic and conflict-prone environments. Given that our testing approach specifically targets conflicts arising from concurrent access to resources, it proves particularly relevant for multi-agent systems developed according to the Agent & Artifact (A&A) paradigm [34], where artifacts play a central role in agent coordination. In this framework, artifacts represent functional and observable entities that serve as mediators for agent interactions and coordination. Since the concept of resources aligns with that of artifacts both functionally and conceptually, our methodology not only enables the evaluation of the robustness of proposed coordination mechanisms but also assesses their ability to effectively manage resource-related conflicts that emerge during agent interactions via these artifacts. Thus, our approach provides a rigorous and systematic framework for testing and validating coordination mechanisms within multi-agent systems based on the A&A model.

The remainder of this work is structured as follows: Section 2 provides an overview of the state-of-the-art approaches for testing coordination mechanisms in MAS, highlighting their limitations. Section 3 presents our methodology based on mutation analysis and details the process of generating and applying mutations. Section 4 presents the experimental results obtained after applying our method to a real case study. Finally, Section 5 concludes this paper by summarizing the main contributions and outlining future work on improving MAS testing.

2. SIMILAR WORKS

In the literature, a limited number of approaches have been proposed for testing MAS in recent years. Subsequently, we outline a selection of these approaches in Table 1.

Table 1. Comparative analysis of testing approaches for multi-agent systems: main contributions and limitations

Reference	Approach	Main Contribution	Limitations (in Light of MAS Global Behaviour & Coordination)
Dehimi et al. [35]	model-based testing using genetic algorithms for holonic agents	iterative testing across evolving versions; focuses on newly introduced behaviors	incremental but limited to individual agent evolution; lacks integrated view of MAS coordination
Dehimi & Mokhati [36]	sequence diagrams with OCL "plugs"	formal test case generation capturing constraints on interactions	focuses on single interactions; does not evaluate coordination/conflict mechanisms
Winikoff [37]	testability of BDI agents (all-edges vs all-paths criteria)	adequacy metrics for BDI programs	structural coverage only; no assessment of coordination or conflict resolution
Gonçalves et al. [38]	Moise+ → CPN4M transformation for testing social behaviour	organisational-level testing with path/state-transition adequacy	restricted to Moise+ model; coordination assumed from org. model, no dynamic adaptability
Rehman et al. [39]	model-based methodology with Prometheus artefacts	fault model for goals, plans, and interactions; new coverage criteria	limited to goal/plan execution; does not consider emergent conflicts or adaptive coordination
Huang et al. [40]	semantic Mutation Testing (SMT) for Jason, GOAL, 2APL	semantic operators to assess robustness of rule-based agents	language-specific; focuses on robustness of logic, not on coordination or conflict handling
Dehimi et al. [41]	mutation + parallel GA for isolating scenarios	detects errors within concurrent scenario executions	only targets scenario isolation; does not ensure robustness of coordination mechanisms
Savarimuthu and Winikoff [42]	mutation operators for Goal language	empirical validation of mutation hypotheses in Goal programs	restricted to Goal language; ignores global MAS interactions and coordination
Dehimi et al. [43]	deep learning for error detection across MAS versions	automated scalable error detection	strong for error prediction but not designed to assess coordination or conflict resilience

Although significant progress has been made in MAS testing through the introduction of new strategies, many of these approaches reveal important limitations. They often lack a global and integrated view of MAS behaviour, focusing instead on specific aspects such as communication, plan execution, or individual agent logic. Moreover, conflict management and coordination mechanisms are often overlooked or assumed to function correctly without evaluation. These shortcomings limit the capacity of existing methods to ensure the overall robustness and adaptability of MAS, particularly in dynamic or unpredictable environments. To address these limitations, we propose a novel approach capable of identifying vulnerabilities in coordination mechanisms and suggesting targeted improvements to enhance their resilience. Additionally, it enables objective comparisons between coordination strategies and supports the development of more robust and adaptive mechanisms. What further strengthens the relevance of our approach compared to

existing ones is its applicability to MAS based on the A&A model, where artifacts play a central role in coordination and interaction management. This compatibility ensures that our method can be directly integrated into systems developed under this paradigm, providing a comprehensive and systematic framework for testing and validating coordination mechanisms in MAS.

3. THE PROPOSED APPROACH

The proposed approach aims to evaluate the effectiveness of the coordination mechanism implemented within a given MAS. It relies on the generation of a series of test cases whose inputs are specifically designed to force the maximum number of agents in the system under test to simultaneously access or occupy the same resource. This scenario inevitably leads to interference among the agents and consequently generates conflict situations around that resource.

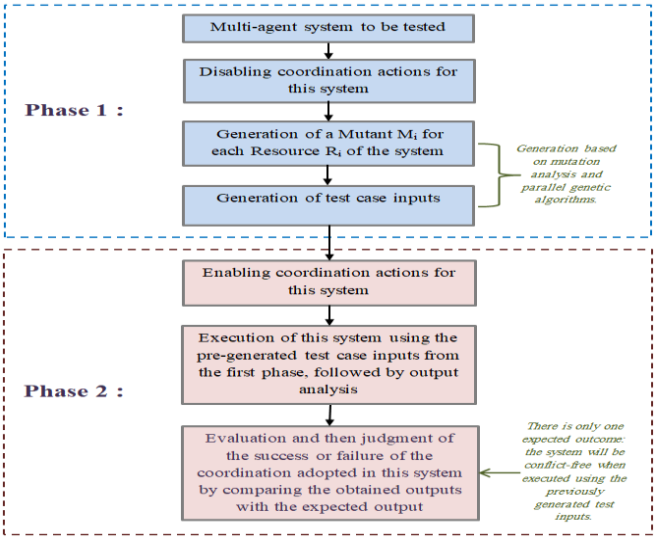


Figure 1. The Phases of the proposed approach for testing coordination in MASs

In cases where the system under test lacks a coordination mechanism, or if such a mechanism is insufficient or faulty, executing the system with these generated test inputs will inevitably lead to system failure. Conversely, if the system integrates a robust and effective coordination mechanism, it will enable the agents to resolve their conflicts, thereby ensuring the system's stability and overall performance. The proposed approach provides a concrete and operational means of evaluating the effectiveness of the coordination mechanism adopted, regardless of the type of coordination implemented (e.g., cooperation, collaboration, negotiation, protocols, planning, etc.). It offers the ability to assess the system's capacity to resolve or manage conflict situations generated by agents competing for access to shared resources. It is essential to emphasize that the emergence and detection of such conflicts depend directly on the quality of the test case inputs generated and applied to the system. Indeed, the more relevant and rigorously designed these inputs are, the higher the likelihood of detecting conflicts, including those that occur rarely and might otherwise go unnoticed in standard testing conditions. Ultimately, this confirms the relevance and robustness of the proposed approach in evaluating whether the adopted coordination mechanism is capable of effectively resolving conflicts, or conversely, fails to manage them. The proposed approach relies on the combined use of mutation analysis techniques and parallel genetic algorithms [44]. It also involves the temporary deactivation of the coordination mechanism to prevent it from masking the conflict situations that form the basis for generating test case inputs. The approach is structured into two main phases, illustrated in Figure 1 and detailed in the following subsection.

3.1 Description of the first phase

First of all, during this initial phase, before starting the procedures for generating test case inputs, it is essential to ensure that the MAS under test is completely free of any coordination mechanism. It is important to remember that coordination includes a set of rules and additional actions necessary to enable interactions and actions among the various agents with a minimum of conflict. These coordination actions can be implemented either in a centralized manner by a single coordinating agent or in a distributed manner by all the agents in the system. Consequently, it is necessary to deactivate all coordination actions, as illustrated in Figures 2 and 3 for the centralized and distributed coordination configurations, respectively. Additional details are provided in Algorithm 1.

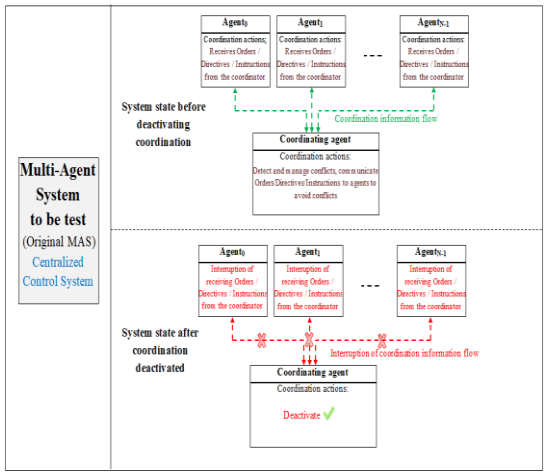


Figure 2. Illustration of the process for disabling a centralized coordination in the multi-agent system to be tested

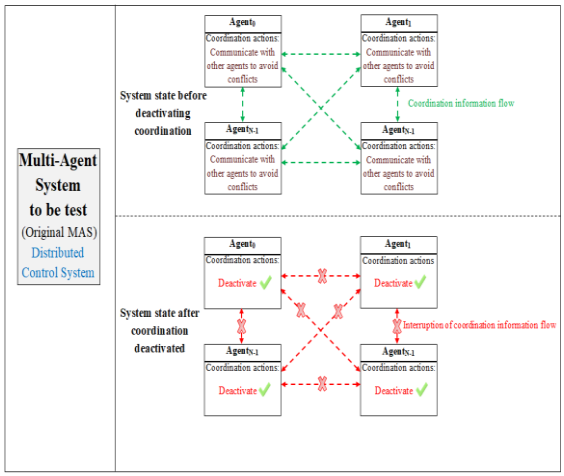


Figure 3. Illustration of the process for disabling distributed coordination in the multi-agent system to be tested

The purpose of this deactivation is to avoid any disruption in the process of generating test case inputs and to ensure the production of high-quality test cases. This guarantees that the MAS is tested exhaustively, taking into account all possible conflict situations between its agents. Indeed, if coordination actions are not deactivated, the system remains controlled and prevents the occurrence of conflict situations. However, these conflict situations are essential for generating test case inputs, as the selection of these inputs depends directly on the number and nature of the conflict situations in which the system under test can find itself.

Algorithm 1: Deactivate_Coordination_for_a_Multi-Agent System

Inputs: Coordinated system **S**;
Outputs: Uncoordinated system **S'**;
1 **Begin**
2 *// Step 1: Identify agents responsible for coordination*
3 CoordinationAgents ← Identify_Coordination_Agents(S);
4 *// Step 2: For each coordination agent, disable its coordination actions*
5 **for** each Agent Aj ∈ CoordinationAgents **do**
6 **for** each coordination action *ac* ∈ Aj **do**
7 /*
8 *Formal deactivation of a coordination action ac:*
9 *Let ac ∈ Aj be a coordination mechanism (e.g., protocol, method, or rule)*
10 *used to manage shared resources (e.g., message sending, negotiation, etc.).*
11 *The operation: Deactivate(ac) ≡ ac.disabled := true*
12 *implies disabling ac at runtime, which may involve:*
13 *- Cancelling or removing its execution from system code,*
14 *- Suspending or deleting the related behaviour (e.g., in JADE),*
15 *- Preventing agent Aj from initiating coordination via ac.*
16 */
17 Deactivate *ac*;
18 **end**;
19 **end**;
20 **End**.

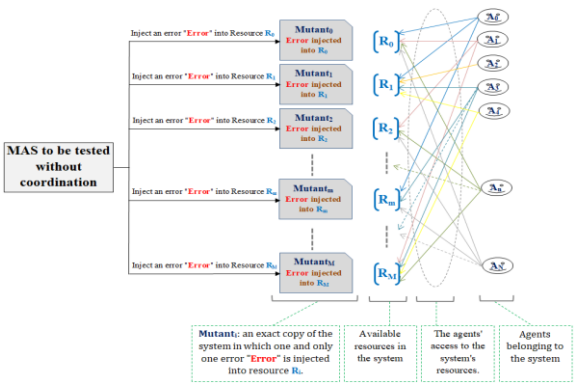


Figure 4. Illustration of the process for generating mutants

The next step in this phase describes how to generate the test case inputs **Inputs$_h$** for the tested MAS, with which we can provoke or direct as many system agents as possible to either access, use, or occupy one of these resources at the same time. This step is based on the mutation analysis technique and parallel genetic algorithms. It consists of producing, for each resource Ri belonging to the system, a mutant. Each produced mutant, called **Mutant$_i$,** represents a copy similar to the system under test. The only difference being that a single error (as formally defined in Eq. (1)) has been injected at the level of instructions that allow an agent $\mathbf{A}_j$ of the system to access, use, or occupy the resource R_i. Here, j is the agent's identifier, with a value between 0 and N-1, where N is the total number of agents in the system. Figure 4 shows how the mutants are generated, highlighting the different key steps of the process. Additionally, Algorithm 2, in pseudo-code form, explains the logic of mutant generation for the test MAS. These two illustrations provide a comprehensive and technical overview of this approach.

Algorithm 2: Mutants_Generation

Inputs: Uncoordinated system **S'**;
Set of resources R = {R_0, R_1, ..., R_{N-1}}; *// Shared resources in the system.*
Outputs: Set of mutants; *// Mutants containing injected errors.*
1 **Begin**

2 **for** each resource R_i ∈ R **do**
3 Mutant$_i$ ← exact copy of system S′; *// Duplicate the uncoordinated system.*
4 Inject R_i.Error into Mutant$_i$; *// Inject an error related to Ri (see the formula in Eq. (1)).*
5 **end**;

6 **End**.

Eq. (1) below describes the error we introduced to obtain a mutant **Mutant$_i$**. This means that the agent A_j is in conflict with the agent $A_{j'}$ (with $j' \neq j$) regarding the use of the resource R_i at the same time t. It should be noted that the symbols "+" in this formula indicate a simple concatenation operation:

$$\{A_{C0}\,(EW_{A_{C0}}), A_{C1}\,(EW_{A_{C1}}), \ldots, A_{Ck-1}\,(EW_{A_{Ck-1}})\}+ R_i + t + V_{dd'} + ED_i + EW_i + \text{"// ERROR "} \quad (1)$$

To better understand the structure of this error, it is important to explain each parameter composing Eq. (1). Here is a detailed explanation of each parameter:

- **$\{A_{C0}\,(EW_{A_{C0}}), A_{C1}\,(EW_{A_{C1}}), \ldots, A_{Ck-1}\,(EW_{A_{Ck-1}})\}$:** A list representing a subset of agents involved in a specific conflict over access to resource R_i. where each element corresponds to an agent competing for the use of this resource. Each agent A_j, is associated with an error weight, denoted EW_{A_j} (Error Weight), which indicates the importance or weight of agent A_j in the conflict. The indices c_0, c_2, …, c_{k-1} are unique identifiers assigned to these agents, with the condition that all indices are distinct (i.e., $c_0 \neq c_1 \neq \ldots \neq c_{k-1}$). These indices refer to agents from the complete set of system agents, denoted $\{A_0, A_1, \ldots, A_{N-1}\}$. In other words, the list $\{A_{C0}\,(EW_{A_{C0}}), A_{C1}\,(EW_{A_{C1}}), \ldots, A_{Ck-1}\,(EW_{A_{Ck-1}})\}$ includes only those agents who, at a given time t, are in conflict due to concurrent access to resource R_i.
- **R_i:** Represents the identifier of the resource over which the agents in the list $\{A_{C0}\,(EW_{A_{C0}}), A_{C1}\,(EW_{A_{C1}}), \ldots, A_{Ck-1}\,(EW_{A_{Ck-1}})\}$ are in conflict. The parameter i refers to the index of this resource, where i ranges from 0 to M−1, with M being the total number of resources available in the system (this number also corresponds to the number of mutants in the

system under test). This resource lies at the core of the conflict among the concerned agents competing for its use at a given moment.

• **t**: Represents the time instant at which a conflict occurs between the agents in the list $\{A_{C0}\ (EW_{A_{C0}}), A_{C1}\ (EW_{A_{C1}}), \ldots, A_{Ck-1}\ (EW_{A_{Ck-1}})\}$ for the use of resource R_i. This moment is crucial for determining the exact point in time when the interference between the agents in this list takes place.

• $\mathbf{V_{dd'}}$: Represents the identifier of the data vector that triggered the error. This vector belongs to the population POP_d, where d is an index used to number the generated populations, ranging from 0 to h, with h being the index of the last generated population. d' belongs to the interval $[0, P_d]$, where d' indicates the position of the vector within the population POP_d. For example, V_{01}, represents the identifier of the second data vector in population POP_0.

• $\mathbf{ED_i}$: The Error Degree associated with $Mutant_i$, where the index i represents the unique identifier of this mutant. This is a key parameter that measures the extent of conflicts related to the use of a specific resource in the system. It corresponds to the total number of agents involved in the conflict over access to resource R_i. It is directly linked to the size of the list $\{A_{C0}\ (EW_{A_{C0}}), A_{C1}\ (EW_{A_{C1}}), \ldots, A_{Ck-1}\ (EW_{A_{Ck-1}})\}$, which includes the conflicting agents. For example, an error degree of 3 means that three agents, $\{A_{C0}, A_{C1}, A_{C2}\}$, are simultaneously in conflict for access to resource R_i at a given time. As the error degree increases, the situation becomes more complex and critical, since it involves a greater number of agents and may impact the overall performance of the system. This criterion thus serves to measure the intensity of the conflict and helps guide the prioritization of adjustments to restore balance in the MAS. For each $Mutant_i$, killed by a data vector $V_{dd'}$, the associated ED_i is calculated using the following Eq. (2):

$$ED_i = Size\ (\{A_{C0}\ (EW_{A_{C0}}), A_{C1}\ (EW_{A_{C1}}), \ldots, A_{Ck-1}\ (EW_{A_{Ck-1}})\}) \quad (2)$$

• $\mathbf{EW_i}$**:** The Error Weight assigned to $Mutant_i$, where i denotes the unique identifier of this mutant. This parameter represents the accumulated weight of the agents involved in a conflict over access to resource R_i. It is calculated as the sum of the individual error weights (EW_{A_j}, Error Weight) of each agent included in the conflict list for that resource. In other words, EW_i reflects the cumulative impact of the agents participating in the conflict over R_i. Thus, for each $Mutant_i$, killed by a data vector $V_{dd'}$, the associated EW_i is computed using the following Eq. (3):

$$EW_i = \sum_{j=C0}^{j=Ck-1} EW_{A_j} = EW_{A_{C0}} + EW_{A_{C1}} + \ldots + EW_{A_{Ck-1}} \quad (3)$$

• **"// ERROR":** Indicates that an error has been detected in the system. This marker signals that the agents in the list $\{A_{C0}\ (EW_{A_{C0}}), A_{C1}\ (EW_{A_{C1}}), \ldots, A_{Ck-1}\ (EW_{A_{Ck-1}})\}$ are in conflict over the use of resource R_i at time t, thereby generating an abnormal situation in the system.

After completing the mutant generation process, the number of mutants produced will be equal to the number of resources in the system under test. The next step is to generate test case inputs that, when executed, are capable of killing as many mutants as possible. More precisely, among all the possible execution inputs, the goal is to identify those that expose the highest number of errors previously injected into these mutants. These inputs are then retained as test case inputs ($Inputs_h$), as they are particularly effective in detecting and eliminating the introduced errors.

It is important to note that a condition must be met for the error to be triggered: there must be at least one conflict ($ED_i \geq 2$). This implies the presence of at least two agents using the same resource at the same time t. To this end, the use of parallel genetic algorithms facilitates the acquisition of high-quality test case inputs, ensuring a thorough test of the effectiveness of the coordination mechanism adopted by the system under test. This process is illustrated in Figure 5.

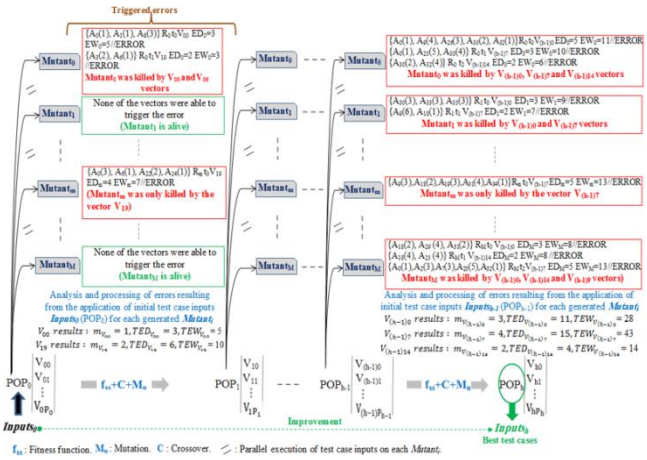


Figure 5. Illustration of test case generation using mutation analysis techniques and genetic algorithms

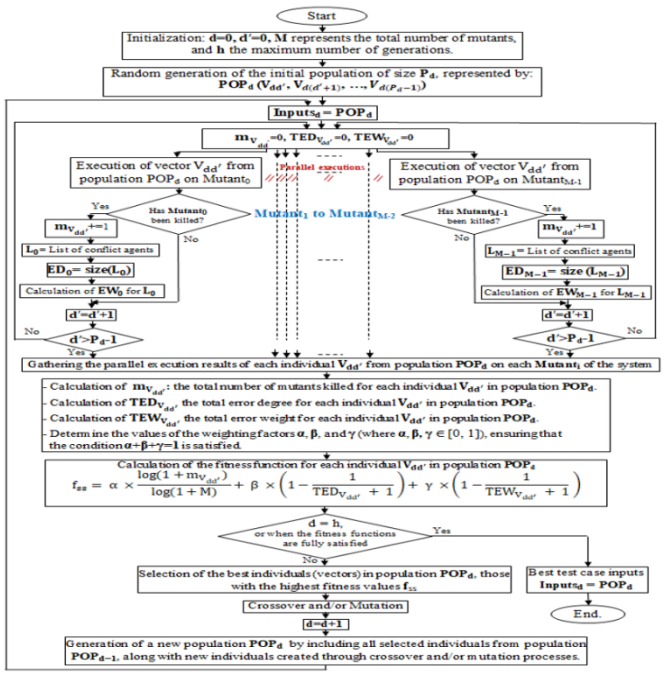


Figure 6. Flowchart of test case input generation using the parallel genetic algorithms technique

The application of parallel genetic algorithms begins with the generation of an initial population, POP_0, composed of several individuals. Each individual, representing a possible test case input, is described by a data vector $V_{dd'}$. Each element of this vector represents one of the data necessary for the execution of the system under test. The initial population POP_0

is created randomly and serves as the basis for the generation of subsequent populations. Each new population, such as POP_1, is improved compared to the previous one, and the process is repeated to produce successive populations. This cycle continues until we obtain a final population, POP_h, capable of achieving the desired objectives.

To clarify and eliminate any ambiguity regarding the process of generating and improving test inputs using genetic algorithms, the flowchart presented in Figure 6 details all the steps involved, from the beginning to the end of this process, and what follows is a complete explanation of all the steps presented therein:

• Creation of an initial population POP_0, consisting of P_0 individuals, each represented by a data vector $V_{0d'}$, where d′ ranges from 0 to P_0-1, such that $(V_{00}, V_{01}, V_{02}, \ldots, V_{0(p0-1)})$. This population is generated randomly.

• Parallel execution of all individuals (vectors) from this population on each previously generated mutant.

• Collection of execution results for each individual in this population: for each data vector $V_{dd'}$, we compute $m_{V_{dd'}}$, representing the total number of mutants killed by this individual $V_{dd'}$. This includes the calculation of $m_{V_{00}}$, $m_{V_{01}}$, $m_{V_{02}}$, …, $m_{V_{0(P0-1)}}$. Additionally, we calculate the Total Error Degree, denoted $TED_{V_{dd'}}$, which includes $TED_{V_{00}}$, $TED_{V_{01}}$, $TED_{V_{02}}$, …, $TED_{V_{0(p0-1)}}$. This parameter is obtained using the following Eq. (4):

$$TED_{V_{dd'}} = \sum_{i=0}^{i=M-1} ED_i \tag{4}$$

Finally, we also calculate the Total Error Weight, denoted $TEW_{V_{dd'}}$, which includes $TEW_{V_{00}}$, $TEW_{V_{01}}$, $TEW_{V_{02}}$, …, $TEV_{V_{0(p0-1)}}$. This parameter is determined by applying the following Eq. (5):

$$TEW_{V_{dd'}} = \sum_{i=0}^{i=M-1} EW_i \tag{5}$$

• Evaluation of each individual in this population is carried out by calculating its fitness function, denoted f_{ss}. In our approach, this fitness function is specifically designed to measure the ability of each solution to maximize conflicts among agents, by promoting an increase in the number of errors and optimizing multiple performance criteria. The fitness function is defined as Eq. (6):

$$f_{ss} = \alpha \times \frac{\log(1+m_{V_{dd'}})}{\log(1+M)} + \beta \times \left(1 - \frac{1}{TED_{V_{dd'}} + 1}\right) + \gamma \times \left(1 - \frac{1}{TEW_{V_{dd'}} + 1}\right) \tag{6}$$

With the following constraint: $\alpha + \beta + \gamma = 1$

In this context, the fitness function is used to identify the best solutions for maximizing conflicts between agents. It does so by considering the number of mutants killed (i.e., the number of generated errors), the degree of error representing the intensity of agents involved in the conflicts, and the cumulative impact of these agents in the conflicts. This ensures a thorough evaluation of each individual in the population.

The meaning of each term in this function is as follows:

• First term: $\alpha \times \frac{\log(1+m_{V_{dd'}})}{\log(1+M)}$

Description. This term expresses the proportion of mutants killed by the individual $V_{dd'}$, denoted by $m_{V_{dd'}}$, relative to the total number of mutants (M). It is weighted by α, the main coefficient, which gives this term a priority role in the optimization of the fitness function.

Impact. By maximizing this term, the approach favours solutions that generate the highest number of errors by killing a greater number of mutants, which indicates a strong ability to trigger conflicts between agents.

• Second term: $\beta \times \left(1 - \frac{1}{TED_{V_{dd'}} + 1}\right)$

Description. This term takes into account the total error degree generated by the vector $V_{dd'}$, denoted as $TED_{V_{dd'}}$, It corresponds to the sum of the error degrees associated with each $Mutant_i$ in the system under test, as mentioned in Eq. (4). The factor β weights this term as a secondary criterion.

Impact. Maximizing this term favors solutions that involve a larger number of agents in conflicts, contributing to a more complex conflict dynamic between agents.

• Third term: $\gamma \times \left(1 - \frac{1}{TEW_{V_{dd'}} + 1}\right)$

Description. This term considers the total error weight, $TEW_{V_{dd'}}$, where the vector $V_{dd'}$ is the source of these errors. It represents the cumulative impact of the agents involved in the errors that have occurred and corresponds to the sum of the individual error weights of agents involved in a conflict for each $Mutant_i$ in the system's set of mutants. This total weight is calculated according to Eq. (5) mentioned previously. This term is weighted by the factor γ, which is the least prioritized of the three.

Impact. Increasing this term aims to favour solutions where the conflicting agents have a high error weight, thereby amplifying the severity of their involvement in these conflicts.

It should be noted that the final result of this function belongs to the interval [0, 1], with 1 representing an ideal solution that simultaneously maximizes the three evaluation criteria.

• Selection of the most promising individuals, aiming to enhance the quality of the initial population, POP_0. Only the individuals with the highest scores according to the fitness function are selected to contribute to this improvement.

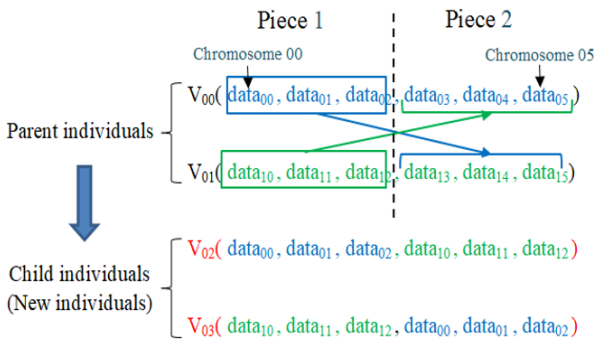


Figure 7. Example of crossover for creating a new individual

• Enrichment of our population by crossing a certain number of previously selected individuals to generate new

individuals. In our approach, we use a simple crossover technique. It involves selecting two or three individuals (parents) and exchanging one or more chromosomes between them to create a new individual (child). A correction step is then performed to eliminate any duplication errors and ensure that the new individuals are unique and valid (see Figure 7).

• A mutation step is integrated into the process. It involves randomly modifying certain individuals in the population by altering one or more genes. These modifications pave the way for new possibilities, which could very well be useful for creating good solutions.

• Generation of a new population, named POP_1. This new generation consists of promising individuals selected from the previous population, to which new individuals created by crossover and mutation are added. These two mechanisms aim to improve the diversity and quality of the population, while preserving the favourable traits identified in the selected individuals.

• Once the new population is generated, the process of improving individuals can be repeated by reapplying the previous steps, until a population of robust individuals fully meeting the fitness function is obtained. This process continues up to a maximum number of generations, denoted as h, determined based on the tester's evaluation. If, at the end of these generations, convergence is not achieved, the best individuals of the last generation (POP_h) will be selected.

3.2 Description of the second phase

After obtaining a series of high-quality test case inputs ($Inputs_h$) and reactivating all the coordination mechanisms in the system under test, previously deactivated during the earlier phase, we will proceed to execute the system with these test inputs. If the system manages to process all these cases without encountering any problems or generating conflicts, we will consider the adopted coordination mechanism to be effective.

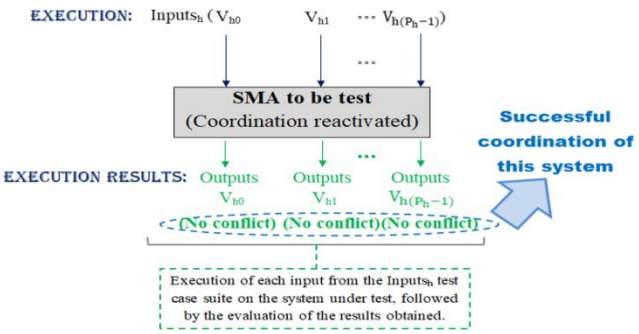


Figure 8. Evaluation of the coordination of the multi-agent system under test (success of the system coordination)

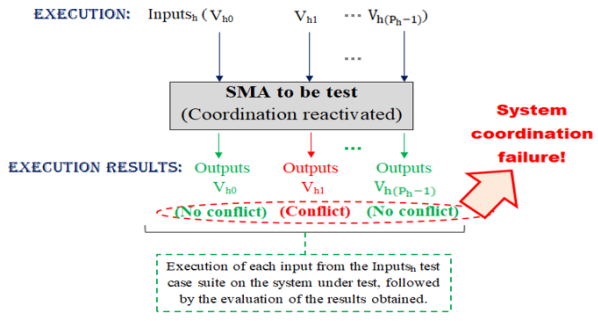


Figure 9. Evaluation of the coordination of the multi-agent system under test (system coordination failure)

On the other hand, if it fails in any of the test case inputs, it will undoubtedly indicate the inefficiency of the adopted coordination mechanism. Figures 8 and 9 briefly illustrate the processes that take place at this stage, while Algorithm 3 demonstrates how coordination is reactivated in the MAS under test.

Algorithm 3: Reactivate_Coordination_for_a_Multi-Agent System

Inputs: Uncoordinated system **S'**;
Outputs: Coordinated system **S**;
Begin
// Step 1: Identify agents responsible for coordination.
CoordinationAgents ← Identify_Coordination_Agents(S');
// Step 2: For each coordination agent, reactivate its coordination action.
for each Agent Aj ∈ CoordinationAgents **do**
for each coordination action ac ∈ Aj **do**
/*
Formal reactivation of a coordination action ac:
This operation is the inverse of Deactivate (ac).
It restores the coordination mechanism ac, such that:
Reactivate(ac) ≡ ac.disabled := false
Meaning: the action ac (e.g., a method, protocol, or rule managing shared resources)
is re-enabled, allowing agent Aj to resume coordinated interactions with other agents
or shared entities.
*/
Reactivate ac;
end;
end;
End.

4. CASE STUDY

4.1 General presentation

In order to validate our proposed approach, we selected the En-route air traffic control system (EATC) as a case study, given its critical and complex nature as a representative domain for testing coordination mechanisms in MAS. EATC, managed in area control centres (ACCs), ensures the surveillance and management of aircraft during the cruise phase to prevent collisions by maintaining minimum vertical (VSM), lateral (LASM), and longitudinal (LOSM) separations, as defined by international regulations (Table 2). To achieve this, controllers rely on advanced technologies such as primary and secondary radar, ADS-B, TCAS, ACARS, and satellite communications (SATCOM), which provide real-time data on aircraft position, velocity, trajectory, and environmental conditions. Figures 10-16 illustrate these separation distances and violation scenarios that lead to conflicts requiring corrective actions (e.g., altitude, heading, or speed adjustments). In this context, EATC is modeled as a MAS using a centralized approach, where an Air Traffic Controller Agent (ControllerAgent) supervises a set of AircraftAgents, each representing an aircraft in cruise. AircraftAgents share their complete trajectories as vectors of waypoints, each characterized by precise attributes such as position, altitude, speed, heading, estimated time of arrival, protection zones, and priority (see Figure 17). Every waypoint is associated with a resource zone (PointResource), which defines its protection area based on VSM, LASM, and LOSM criteria. The ControllerAgent continuously analyzes the

resources of all AircraftAgents and detects conflicts whenever protection zones overlap simultaneously in all three dimensions of separation. Upon conflict detection, it generates tailored corrective instruction vectors for the involved aircraft, adjusting parameters such as altitude, trajectory, or speed to restore safe separation without creating new conflicts. Each AircraftAgent then updates its trajectory accordingly. The cruise environment is represented as a dynamic operational airspace, encompassing external factors such as weather conditions, airspace configuration, and relative speeds, all of which influence both the ControllerAgent's decisions and AircraftAgents' behaviors. This MAS-based modeling of EATC provides a realistic and demanding framework for evaluating the robustness and effectiveness of coordination mechanisms in critical systems by subjecting them to representative and challenging conflict scenarios, thereby testing their resilience and capacity to ensure flight safety.

Table 2. Summary of minimum vertical and horizontal separation distances between aircraft in cruise phase

Type of Separation	Minimum Separation Distance	Separation Conditions
Vertical separation minimal (VSM) (See Figure 10)	1000 feet[1] (304.8 meters)	Below FL290 (Flight Level 290, approximately 29000 feet)
	1000 feet (304.8 meters)	Between FL290 and FL410 for aircraft equipped with RVSM[2] systems
	2000 feet (609.6 meters)	Above FL290 for No-RVSM aircraft
Horizontal separation minimal (HSM[3])		
Lateral separation minimal (LASM) (See Figure 12)	5 nm[4] (approx. 9.26 km)	general regulations concerning the minimum distance for lateral separation in cruise for radar-controlled aircraft.
	3 nm (approx. 5.56 km)	applicable in certain airspace areas with dense traffic or very accurate surveillance systems, such as near airports.
Longitudinal separation minimal (LOSM) (See Figures 14 and 15)	5 nm (approx. 3 minutes)	between two aircraft on the same route with different speeds, where the leading aircraft is assumed to fly at least 44 kt[5] (81.49 km/h) faster than the following aircraft.
	10 nm (approx. 5 minutes)	between two aircraft on the same route with different speeds, where the leading aircraft is assumed to fly at least 22 kt (40.74 km/h) faster than the following aircraft.

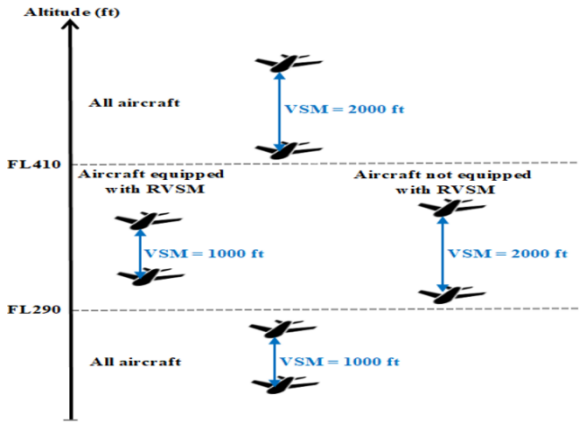


Figure 10. Representation of different vertical separation minima (VSM) between aircraft in cruise phase

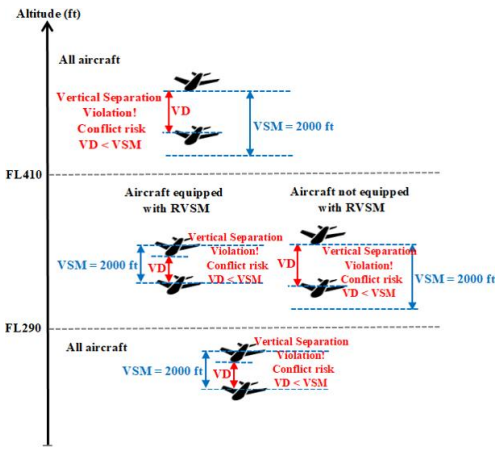


Figure 11. Representation of vertical separation violation situations when one of the two aircraft breaches their VSM

Notes: 1. Feet: 1 foot (ft) equals 0.3048 metres. 2. RVSM (Reduced Vertical Separation Minima): RVSM is a standard that reduces the vertical separation minima (VSM) between aircraft flying between flight levels FL290 and FL410 from 2,000 feet to 1,000 feet. The aim is to increase airspace capacity by allowing more aircraft to operate at optimal cruising altitudes, thereby improving fuel efficiency and reducing traffic congestion [45-47]. 3. HSM (Horizontal Separation Minimum): Can be measured in two ways: by lateral distance and by longitudinal distance. 4. Nautical Miles: 1 nautical mile (nm) equals 1.852 kilometres. 5. kt (knot): kt is the standard unit of measurement for aircraft speed, where 1 kt = 1 nautical mile per hour = 1.852 kilometres per hour.

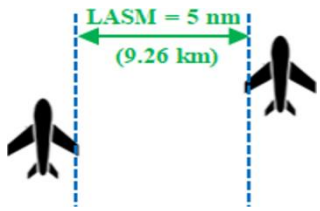


Figure 12. Representation of lateral separation minima (LASM) between two aircraft flying at the same or closely spaced altitudes

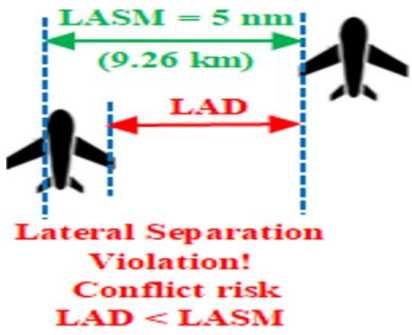


Figure 13. Representation of a lateral separation violation situation when one of the two aircraft breaches their LASM

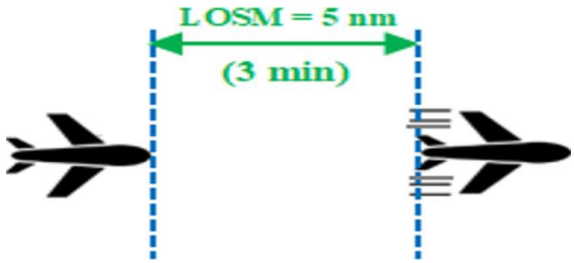


Figure 14. Representation of longitudinal separation minima (LOSM) between two aircraft flying at the same or closely spaced altitudes, where the leading aircraft must fly at least 44 kt (81.49 km/h) faster than the following aircraft

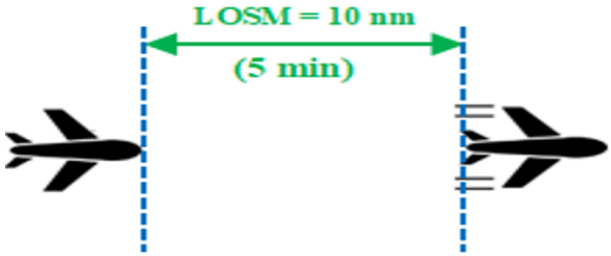


Figure 15. Representation of LOSM between two aircraft flying at the same or closely spaced altitudes, where the leading aircraft must fly at least 22 kt (40.74 km/h) faster than the following aircraft

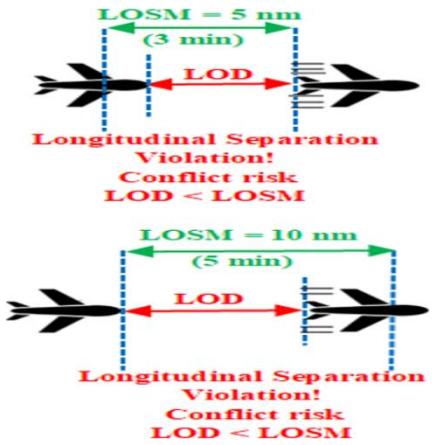


Figure 16. Representation of a longitudinal separation violation situation when one of the two aircraft breaches their LOSM

Figure 17. An example of a trajectory for an aircraft agent (AircraftAgent0)

4.2 Application of the proposed approach to the EATC system

In accordance with our testing methodology, which is based on seven essential steps, we shall now examine and apply them one by one:

4.2.1 MAS to be tested

As outlined in Section 4.1, the system under test is a centralised MAS designed for cruise-phase air traffic control (EATC). It comprises 20 aircraft agents (AircraftAgent), each following its own trajectory in cruise airspace (TrajectoryVector). Each trajectory consists of a series of points (Point), with the number of points ranging from 6 to 30, excluding the departure and arrival points. These points represent estimated spatial positions of each agent at specific time instances, under the assumption of a constant cruising speed. External influences such as wind, weather, or other environmental factors are not considered in this model. To manage traffic and prevent conflicts, a controller agent (ControllerAgent) is responsible for coordinating the actions of all aircraft, thereby facilitating safe and efficient navigation within the cruise airspace.

Figure 18 illustrates the overall architecture of the system under test, highlighting the interactions between the various agents involved in managing cruise-phase air traffic. Within this framework, each AircraftAgent periodically transmits its trajectory information (TrajectoryVector or TV) to the ControllerAgent, who oversees the coordination process to detect and resolve potential conflicts, ensuring smooth and secure navigation.

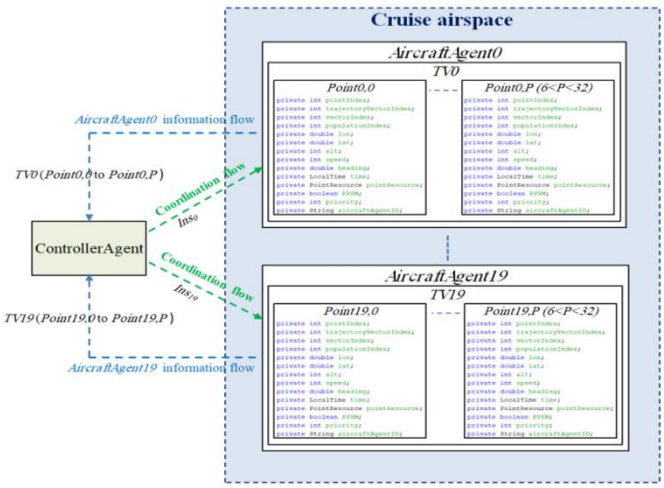


Figure 18. Multi-agent system to be tested (EATC)

4.2.2 Disabling coordination actions for this system

In this step, the coordination mechanism that enables trajectory adjustments in the event of conflicts is deliberately deactivated. This prevents the ControllerAgent from sending adjustment instructions to the AircraftAgents, effectively disabling any conflict resolution intervention. Figure 19 illustrates this deactivation by showing that the ControllerAgent continues to receive the trajectories of the 20 AircraftAgents (from TV0 to TV19), but that the correction instructions (Ins_0 to Ins_{19}) are no longer sent. The red crosses clearly indicate that the transmission of adjustments is blocked, thus preventing any modification of heading or altitude by the AircraftAgents.

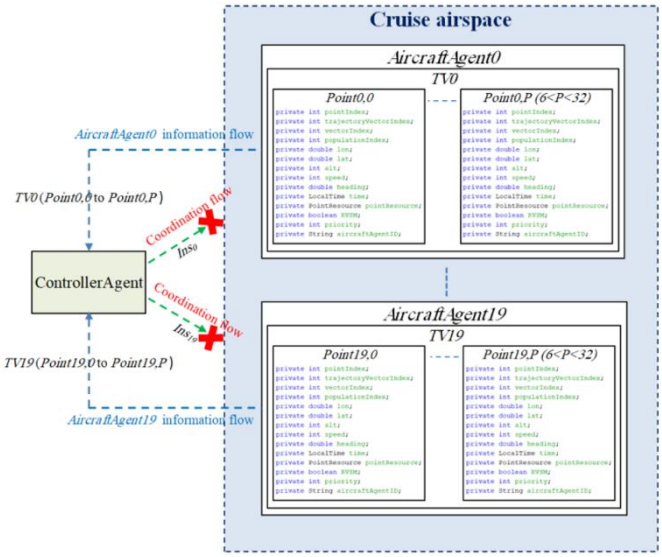


Figure 19. Illustration of the coordination flow deactivation for the EATC system

4.2.3 Generation of mutants for each system resource

Following the deactivation of the coordination mechanism in the previous step, the system under test becomes an "EATC-Without Coordination" configuration. In this state, communication between the ControllerAgent and the AircraftAgents is completely disabled. Each AircraftAgent then follows its trajectory autonomously, without receiving any corrective adjustments in the event of a conflict. The resources of the system are the spatial coordinates in cruise airspace, since each agent navigates independently during this phase and uses these coordinates to position itself in space. Each AircraftAgent passes through multiple waypoints along its trajectory. If a conflict is to arise, it will necessarily occur at these points-where the trajectories of different aircraft agents may intersect.

In this step, each waypoint in a trajectory is treated as a resource. According to the mutation analysis technique, an error is injected at each point (as defined in Eq. (1), Section 3.1). An error is triggered if another point, belonging to a different trajectory, enters the safety zone around that point (as defined by VSM, LOSM, and LASM thresholds). In other words, each time an error is injected, a mutant is created. This mutant is an exact copy of the original system, except for a single error introduced at one specific point.

At the end of this process, the total number of mutants corresponds to the total number of points across all trajectories. Since the system comprises 20 AircraftAgents (and thus 20 trajectories), each containing between 8 and 32 points, the total number of mutants ranges from 160 to 640. This approach allows the system's coordination mechanism to be tested under conflict scenarios in cruise phase, by observing how each mutant behaves when safety zones (VSM, LOSM and LASM) are violated.

4.2.4 Generation of test case inputs

As illustrated in Section 3.1, through Figures 7 and 8, the process of generating test case inputs is carried out using parallel genetic algorithms. Below is a detailed presentation of the various steps that make up this process, from initialisation to the generation of the desired test case inputs:

Process 1: Generation of an Initial Population. Test cases are modelled as individuals (Vector or V) within an initial population (with a fixed size of 100), which is generated randomly. This diversity is crucial for effectively exploring the solution space and avoiding bias from the outset.

Each individual consists of 20 genes (TrajectoryVector or TV), and each gene contains between 8 and 32 waypoints (Point). Figures 20 and 21 illustrate the structure of the first and last individuals in the initial population. Furthermore, Table 3 clearly presents the parameters used to generate a trajectory (TrajectoryVector).

Table 3. Key parameters for generating an aircraft trajectory (Trajectory Vector)

Parameter	Description	Value/Range	Unit
InitialPointIndex	index of the starting point	0	Integer
minIntermediatePoints	minimum number of intermediate points	6	Integer
maxIntermediatePoints	maximum number of intermediate points	30	Integer
stepPointsInVector	step between points in the vector	1	Integer
leastDistance	minimum distance between origin and destination	1000	Nautical miles (nm)
minLon / maxLon	longitude range	-180 to 180	Degrees
stepLon	longitude step	0.00001	Degrees
minLat / maxLat	latitude range	-90 to 90	Degrees
stepLat	latitude step	0.00001	Degrees
minAltitude / maxAltitude	altitude range	20,000 to 51,000	Feet (ft)
stepAltitude	altitude step	20	Feet (ft)
minSpeed / maxSpeed	speed range	350 to 600	Knots (kt)
stepSpeed	speed step	20	Knots (kt)
minStartingTime / maxStartingTime	time range	0 to 23	Hours
stepStartingTime	time step	1	Hours
VSMValue	vertical separation minimum	1000 or 2000	Feet (ft)
LASMValue	lateral separation minimum	500	Nautical miles (nm)
LOSMValue	longitudinal separation minimum	1000	Nautical miles (nm)
RVSM	reduced vertical separation minimum	True or False	Boolean
priority	agent priority	0 to 19	Integer
EARTH_RADIUS_NM	earth's radius	3440.065	Nautical miles (nm)
AgentName	aircraft agent name	AircraftAgent	String
maxIntermediatePoints	maximum number of intermediate points	30	Integer

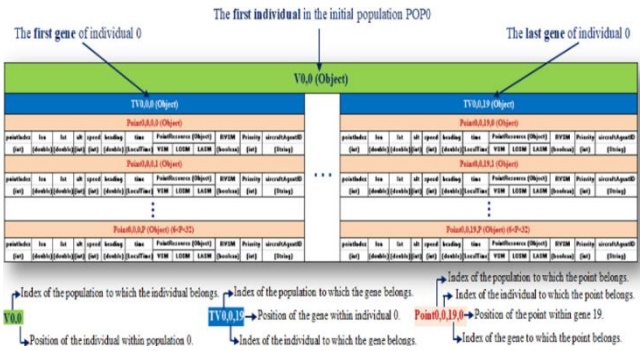


Figure 20. Structure of the first individual in the initial population (POP0)

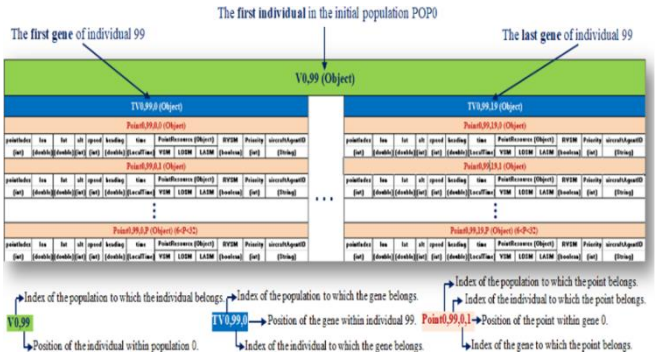
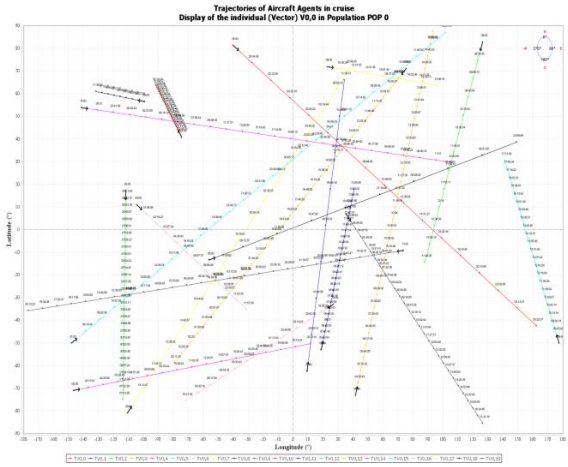
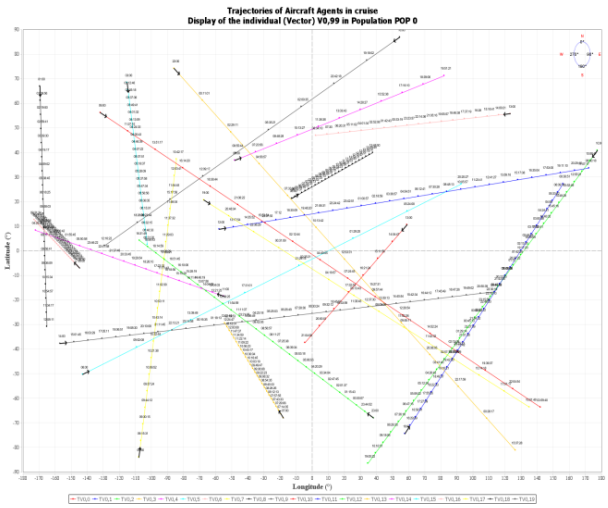


Figure 21. Structure of the last individual in the initial population (POP0)

Figure 22. Summary of the generation of the initial population (POP0)



(a) Graphical representation of individual V0,0



(b) Graphical representation of individual V0,99

Figure 23. Graphical Representation of the First (V0,0) and Last (V0,99) Individuals of the Generated Initial Population (POP0)

Figure 22 provides a summary view of the display after the generation of the initial population POP0, while Figure 23 graphically represents the first individual (V0,0) and the last (V0,99) of this population.

Process 2: Mutant Generation. In an initial population of 100 individuals, where each individual is composed of 20 genes, and each gene contains between 8 and 32 waypoints (resources), the total number of mutants generated per individual ranges from 160 to 640. Consequently, at the population level, the total number of generated mutants lies between 16000 and 64000. Figure 24 illustrates a sample output showing a subset of the generated mutants.

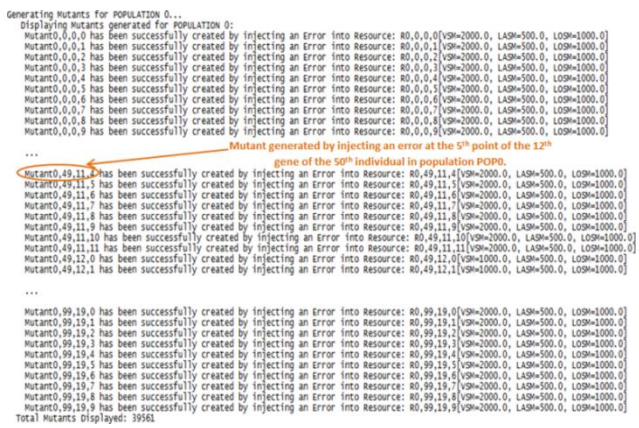


Figure 24. Excerpt of the display of mutants generated for the initial population POP0

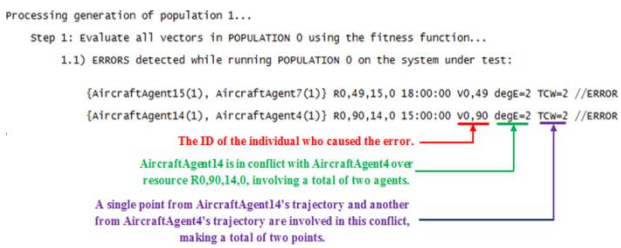


Figure 25. Excerpt of errors detected during the parallel execution of POP0 on each generated mutant

Process 3: Error Triggering and Evaluation of Individuals. During the parallel execution of each individual from the initial population on each previously generated mutant (see Figure 5, Section 3.1), one or more errors may occur (as defined in Eq. (1), Section 3.1). This indicates simultaneous use of the same resource, meaning that safety separation rules are violated at certain waypoints. In such cases, the corresponding mutants are considered "killed". Figure 25 illustrates a representative excerpt of the detected errors, and Table 4 details the extracted information.

Table 4. Information extracted from the parallel execution of the initial population on all generated mutants

Triggered Errors	Individual ID (Error Source)	Mutant ID (Killed)	Involved Agent	Error Degree (ED or degE)	Conflict Weight (CW)	Total Conflict Weight (TCW)	Number of Killed Mutants (m)	Total Mutants (M)	Total Error Degree (TED)	Total Error Weight (TEW)
Error 1	V0,49	M0,49,15,0	AircraftAgent15 AircraftAgent7	2	1 point included 1 point included	2	1	427	2	2
Error 2	V0,90	M0,90,14,0	AircraftAgent14 AircraftAgent4	2	1 point included 1 point included	2	1	436	2	2

Following the analysis of the triggered errors and the extraction of relevant information for the evaluation of each individual in the initial population, we apply the previously defined fitness function, using the parameters α=0.6, β=0.3 and γ=0.1 (see Eq. (6), Section 3.1). Figure 26 provides a detailed excerpt of the evaluations for each individual, while Figure 27 offers a graphical overview of the fitness-based evaluation for individuals ranging from V0,40 to V0,79.

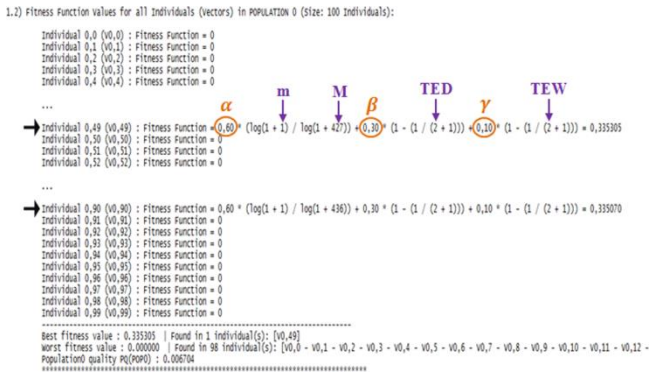


Figure 26. Detailed extract of individual evaluations from population POP0

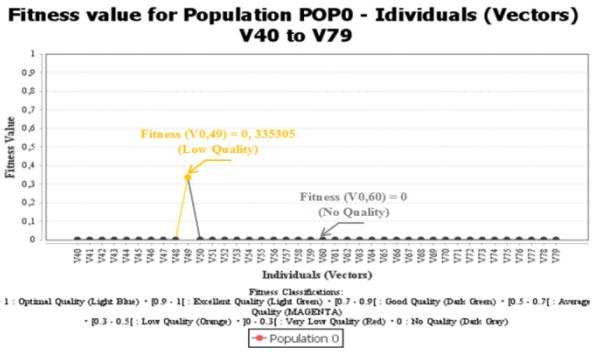


Figure 27. Graphical representation of fitness values of individuals (v0,40 to v0,79) in population POP0

Process 4: Selection of the Fittest Individuals. Following the evaluation of the initial population, the next step is to select the fittest individuals - those with the highest performance scores. A selection rate is randomly determined within a predefined range of 60% to 90%. In this case, the selected rate is 82.13%, meaning that 82 individuals from the initial population have been chosen for the next stage.

Process 5: Crossover and Mutation. To diversify and enrich our population, we performed simple crossover between every two parent individuals to produce two offspring. The crossover probability was randomly selected between 70% and 100%. Additionally, the crossover point was chosen from the following authorized values: 30%, 40%, 50%, 60%, and 70%. However, due to computational and memory constraints, the total number of individuals generated after each crossover operation (parents + offspring) was limited to a maximum of 40.

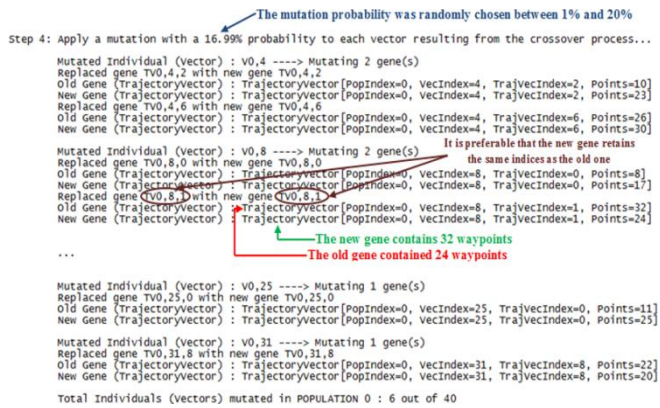


Figure 28. Display of mutated individuals from the initial population after crossover

Mutation was also applied, with a randomly selected probability between 1% and 20%, determining the chance for each individual to be mutated. When an individual was selected for mutation, only one of its genes was modified, with the gene being randomly chosen. Figure 28 illustrates an excerpt of the displayed results, showing a mutation probability of 16.99%, which led to the mutation of 6 individuals out of a total of 40.

Process 6: Generation of the New Population. This process involves creating a new population that incorporates all individuals after mutation. To achieve this, new indices must

be generated: the index of the previous population is incremented by 1, and all indices of individuals, genes, and waypoints are updated accordingly. Figure 29 presents a sample display of a newly generated population, POP1.

Figure 29. An excerpt of the display of a new population POP1

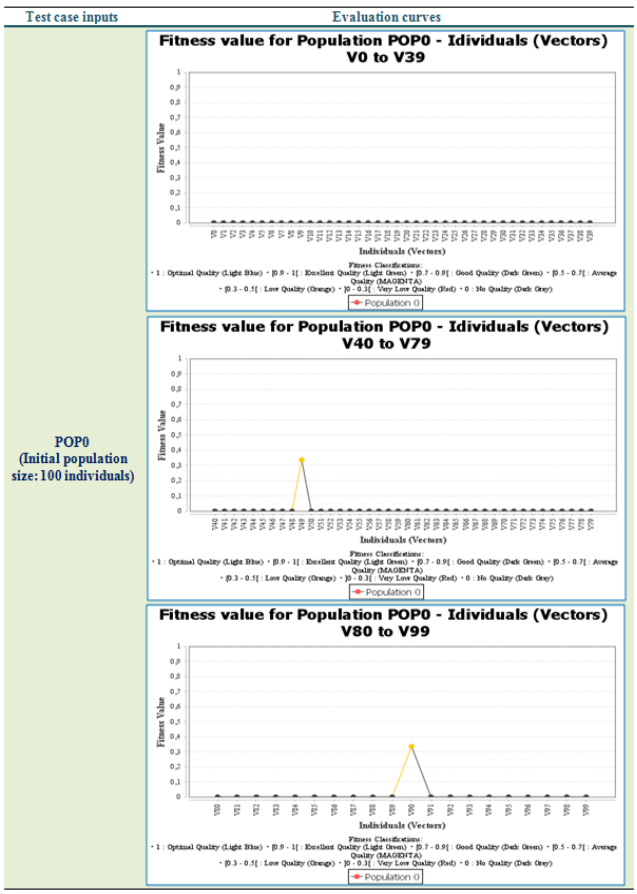


Figure 30. Evaluation curves of test case inputs (Inputs0)

Process 7: Development of Test Case Inputs. The development of test case inputs is based on the repeated execution of the previous processes (Processes 1 to 6) until the fitness function reaches a stable state. In other words, as long as the fitness scores of individuals continue to improve, the algorithm proceeds with its evolution. However, once no significant improvement is observed, it is considered that the algorithm has reached convergence.

Despite constraints - most notably the limitation to 40 individuals per new generation - a fitness score of 98.1281% was achieved after 150 generations. While this restriction may have slowed convergence, it allowed for more efficient management of computational and memory resources.

Once convergence is achieved, individuals from the final generation are selected as the best difficult test case inputs. These represent the most relevant scenarios for evaluating the coordination mechanism of the system under test.

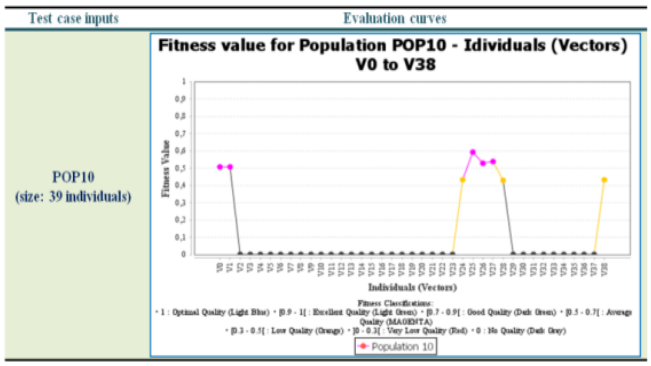


Figure 31. Evaluation curves of test case inputs (Inputs10)

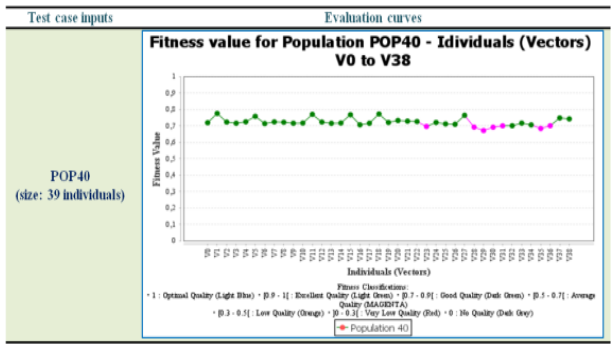


Figure 32. Evaluation curves of test case inputs (Inputs40)

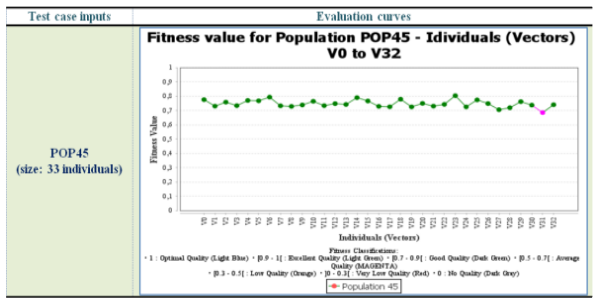


Figure 33. Evaluation curves of test case inputs (Inputs45)

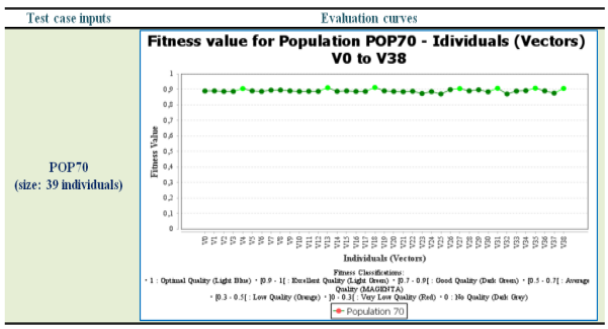


Figure 34. Evaluation curves of test case inputs (Inputs70)

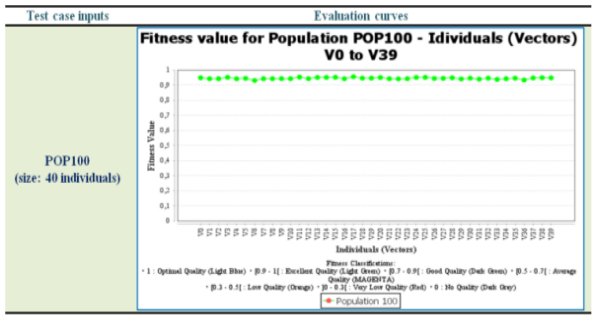


Figure 35. Evaluation curves of test case inputs (Inputs100)

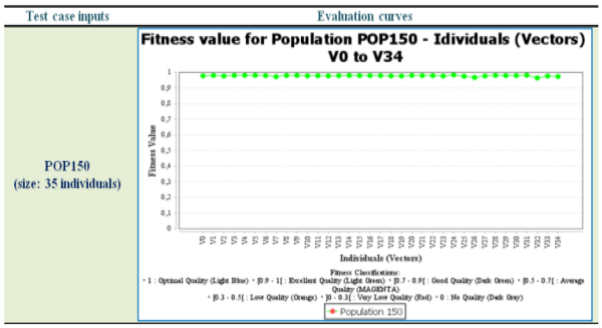


Figure 36. Evaluation curves of test case inputs (Inputs150)

Figures 30 to 36 illustrate the progressive evolution of test case input quality, ranging from poor to excellent. They first show low-quality inputs, followed by moderate and good quality, and finally, excellent-quality inputs. In this final category, the best score achieved is 98.1281%.

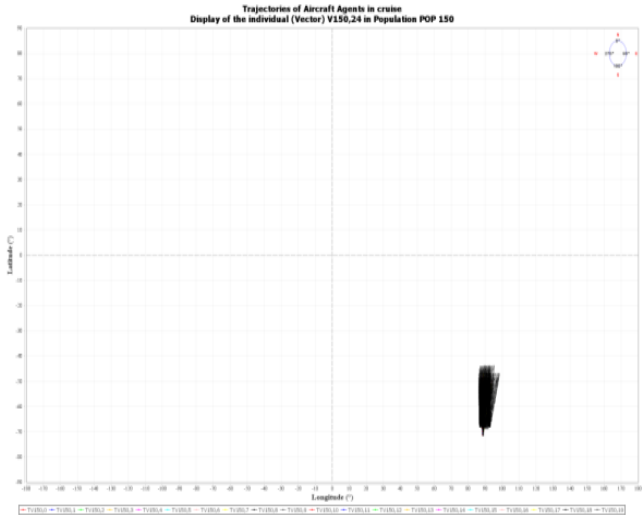


Figure 37. Graphical representation of one of the best test case inputs, represented by individual V150,24

At the peak of this progression, Figure 37 provides a graphical representation of individual V150,24, identified as the best test case input. Additionally, Table 5 presents a detailed analysis of each test case input illustrated in Figures 30-36, highlighting their evolution in terms of performance and quality.

Table 5. Detailed analysis of test case input evolution in terms of performance and quality

Test Case Inputs	Size	Population Analysis Results				Analysis Results of the Worst Individual in the Population				Analysis Results of the Best Individual in the Population			
		Number of Detected Errors (Conflicts)	worst fitness Value	Best fitness Value	Population Quality (PQ)	Number of Killed Mutants (m)	Number of Generated Mutants (M)	Total Error Degree (TED)	Total Error Weight (TEW)	Number of Killed Mutants (m)	Number of Generated Mutants (M)	Total Error Degree (TED)	Total Error Weight (TEW)
Inputs0 (POP0)	100 Individuals	2 Errors	0.000000	0.335305	0.006704	0	/	0	0	1	427	2	2
Inputs10 (POP10)	39 Individuals	27 Errors	0.000000	0.591954	0.101607	0	/	0	0	7	435	29	29
Inputs40 (POP40)	39 Individuals	1001 Errors	0.670151	0.775251	0.721229	15	444	150	150	44	423	174	174
Inputs45 (POP45)	33 Individuals	1098 Errors	0.684276	0.802932	0.746361	16	356	80	80	57	411	235	235
Inputs70 (POP70)	39 Individuals	5710 Errors	0.869194	0.910780	0.889581	119	450	485	485	166	404	580	580
Inputs100 (POP100)	40 Individuals	10579 Errors	0.929362	0.953865	0.943855	230	474	872	872	293	469	1039	1039
Inputs150 (POP150)	35 Individuals	13805 Errors	0.962033	**0.981281**	**0.976064**	323	476	1175	1175	428	519	1522	1522

In our coordination testing approach, the main objective is clearly defined: to generate the most challenging possible test case inputs. This means producing scenarios that maximise conflicts, trigger a higher number of errors, and eliminate as many mutants as possible.

This objective is directly reflected in Table 5, which illustrates the evolution of generations and highlights the progressive increase in the number of detected conflicts, as well as the growing complexity of the generated errors, as expressed by the TED (Total Error Degree) and TEW (Total Error Weight) indices. Furthermore, it underscores the rise in the number of eliminated mutants, represented by the term (m). The analysis of the results confirms that the genetic algorithm successfully increases the complexity of test cases, thereby enabling a more rigorous and in-depth evaluation of the coordination mechanism within the tested system.

4.2.5 Enabling coordination actions for this system

This step corresponds to the reactivation of the coordination mechanism, which had been disabled in Step 2 (see Figure 19).

Figure 38 provides an abstract illustration of this reactivation: the ControllerAgent continues to receive the trajectories of the 20 aircraft agents (from TV0 to TV19), but this time, it is once again able to intervene in the event of a conflict. Adjustment instructions are therefore sent to the relevant agents to correct their trajectories when necessary.

The ControllerAgent functions as a central coordinator, with the primary role of ensuring safe separation between the trajectories transmitted by the various AircraftAgents. It does so by enforcing the criteria for vertical, lateral, and longitudinal separation (VSM, LASM and LOSM). The adopted coordination strategy focuses exclusively on adjusting the altitude of conflicting trajectories, without altering either the speed or the heading.

• Coordination Strategy Adopted:

a. Trajectory reception: The process begins when each AircraftAgent sends its complete trajectory to the ControllerAgent, represented as a list of waypoints (Point). Once all trajectories have been received, the controller can initiate the analysis.

b. Conflict detection: The controller identifies potential conflict points between aircraft using the function "getAllConflictPoints(trajectoryVectorList)", which compiles all such points for further processing.

c. Conflict resolution (Multi-Cycle Approach): he controller attempts to resolve conflicts over multiple successive cycles, continuing until either all conflicts are resolved or the maximum number of cycles (MAX_CYCLES, set to 10) is reached.

In each cycle, the controller calls the "resolveConflicts(allConflictPointsMap)" function to process the detected conflicts and generate adjustment instructions to be sent to the relevant aircraft agents.

For each group of conflicting points:

• The point with the highest priority (i.e., the lowest numerical value) is protected and remains unchanged.

• The remaining points are adjusted one by one, in descending order of priority (from less to more important).

• For each adjusted point, a safe altitude is determined using the method "findSafeAltitude(pointToAdjust, referencePoints)", taking into account:

• The required minimum vertical separation (VSM) must be maintained between all points;

• The altitude of the highest-priority point remains unchanged;

• The altitudes of other points already adjusted during the same cycle are considered to avoid introducing new conflicts;

• The suggested altitude must remain within authorised bounds, between FL200 (minimum) and FL510 (maximum).

All adjustments are recorded locally using structures of type "AdjustmentInstructions".

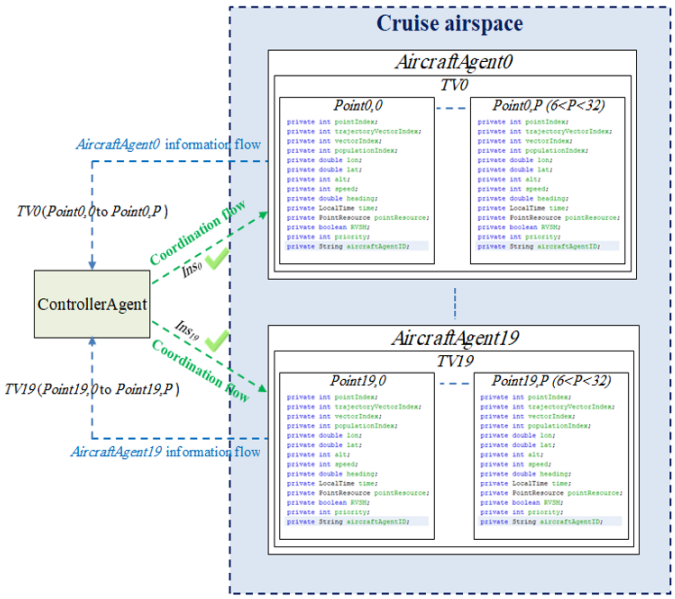


Figure 38. Reactivation of the coordination process

Table 6. System execution results for the best test cases from each predefined input set

Inputs	Inputs Quality (%)	Top 2 Test Cases per Input	Fitness Score (%)	Test Case Execution Results: Number of Planned Conflicts (Errors)	Resolved Conflicts / Resolution Success: Number of Adjustments / Total Number of Adjustment Points After Cycle 1	After Cycle 2	After Cycle 3	After Cycle 4	After Cycle 5	After Cycle 6	After Cycle 7	After Cycle 8	After Cycle 9	After Cycle 10	Latest Agents' IDs Affected by the Adjustments	Index of the Affected Points in the Latest Adjustments	Previous Altitude (ft)	Safe Altitude (ft)	Planned Conflicts / Resolved Conflicts
$Inputs_0$: 100 Test Cases	0.6704	V0,49	33.5305	1	1/Yes 1/1	/	/	/	/	/	/	/	/	/	AircraftAgent15	Point0,49,15,0	36600	38600	1 / 1 Success: 100%
		V0,90	33.5070	1	1/Yes 1/1	/	/	/	/	/	/	/	/	/	AircraftAgent14	Point0,90,14,0	32440	34440	1 / 1 Success: 100%
$Inputs_{10}$: 39 Test Cases	10.1607	V10,25	59.1954	7	7/Yes 4/5	/	/	/	/	/	/	/	/	/	AircraftAgent14 AircraftAgent15 AircraftAgent17 AircraftAgent18	Point10,25,14,0 Point10,25,15,0 Point10,25,17,0 Point10,25,18,0 Point10,25,18,23	32440 32440 32440 32440 32440	28440 36440 30440 34440 34440	7 / 7 Success: 100%
		V10,27	53.7376	4	4/Yes 3/3	/	/	/	/	/	/	/	/	/	AircraftAgent14 AircraftAgent17 AircraftAgent18	Point10,27,14,0 Point10,27,17,0 Point10,27,18,0	32440 32440 32440	36440 30440 34440	4 / 4 Success: 100%
$Inputs_{40}$: 40 Test Cases	72.1229	V40,1	77.5251	44	43/No 9/27	44/Yes 1/1	/	/	/	/	/	/	/	/	AircraftAgent12	Point40,1,12,0	32440	36420	44 / 44 Success: 100%
		V40,11	76.9086	38	37/No 9/23	38/Yes 1/1	/	/	/	/	/	/	/	/	AircraftAgent12	Point40,11,12,0	38420	36420	38 / 38 Success: 100%
$Inputs_{45}$: 33 Test Cases	74.6361	V45,23	80.2932	57	54/No 12/35	55/No 2/2	55/No 1/1	55/No 1/1	55/No 1/1	55/No 1/1	57/Yes 1/1	/	/	/	AircraftAgent19	Point45,23,19,0	47960	48960	57 / 57 Success: 100%
		V45,6	79.2381	51	50/No 9/30	51/Yes 1/1	/	/	/	/	/	/	/	/	AircraftAgent12	Point45,6,12,0	38420	36420	51 / 51 Success: 100%

Table 7. Additional results for the remaining test cases

Inputs	Inputs Quality (%)	Top 2 Test Cases per Input	Fitness Score (%)	Number of Planned Conflicts (Errors)	Test Case Execution Results: Resolved Conflicts / Resolution Success Number of Adjustments / Total Number of Adjustment Points After Cycle 1	After Cycle 2	After Cycle 3	After Cycle 4	After Cycle 5	After Cycle 6	After Cycle 7	After Cycle 8	After Cycle 9	After Cycle 10	Latest Agents' IDs Affected by the Adjustments	Index of the Affected Points in the Latest Adjustments	Previous Altitude (ft)	Safe Altitude (ft)	Planned Conflicts / Resolved Conflicts
Inputs$_{70}$: 39 Test Cases	88.9581	V70,18	91.0780	166	162/No 13/99	162/No 3/3	162/No 2/2	162/No 2/2	162/No 2/2	164/No 2/2	164/No 1/1	164/No 1/1	165/No 1/1	164/No 1/1	AircraftAgent19	Point70,18,19,0	48960	47960	166 / 164 Success: 98.7952 %
		V70,13	90.9035	184	176/No 15/104	172/No 7/7	172/No 6/6	174/No 6/6	174/No 5/5	175/No 5/5	175/No 5/5	176/No 5/5	177/No 5/5	176/No 5/5	AircraftAgent13	Point70,13,13,0	46960	45960	184 / 176 Success: 95.6522 %
															AircraftAgent16	Point70,13,16,0	47960	46960	
															AircraftAgent17	Point70,13,17,0	47960	48960	
															AircraftAgent18	Point70,13,18,0	46960	47960	
															AircraftAgent19	Point70,13,19,0	48960	47960	
Inputs$_{100}$: 39 Test Cases	94.3855	V100,17	95.3865	293	284/No 17/167	277/No 8/8	277/No 8/8	279/No 8/8	279/No 7/7	281/No 7/7	281/No 6/6	283/No 6/6	283/No 5/5	284/No 5/5	AircraftAgent11	Point100,17,11,0	45960	46960	293 / 284 Success: 96.9283 %
															AircraftAgent13	Point100,17,13,0	46960	47960	
															AircraftAgent16	Point100,17,16,0	47960	48960	
															AircraftAgent18	Point100,17,18,0	48960	49960	
															AircraftAgent19	Point100,17,19,0	50960	49960	
		V100,15	95.0592	286	278/No 15/174	272/No 7/7	272/No 7/7	274/No 7/7	274/No 6/6	276/No 6/6	276/No 5/5	278/No 5/5	278/No 4/4	279/No 4/4	AircraftAgent13	Point100,15,13,0	46960	47960	286 / 279 Success: 97.5525 %
															AircraftAgent16	Point100,15,16,0	47960	48960	
															AircraftAgent18	Point100,15,18,0	48960	49960	
															AircraftAgent19	Point100,15,19,0	50960	49960	
Inputs$_{150}$: 35 Test Cases	97.6064	V150,24	98.1281	428	418/No 18/252	410/No 9/9	412/No 9/9	412/No 8/8	414/No 8/8	414/No 7/7	416/No 7/7	416/No 6/6	416/No 6/6	417/No 6/6	AircraftAgent12	Point150,24,12,0	45960	46960	428 / 417 Success: 97.4299 %
															AircraftAgent14	Point150,24,14,0	46960	47960	
															AircraftAgent16	Point150,24,16,0	46960	47960	
															AircraftAgent17	Point150,24,17,0	47960	48960	
															AircraftAgent18	Point150,24,18,0	49960	50960	
															AircraftAgent19	Point150,24,19,0	50960	49960	
															AircraftAgent12	Point150,31,12,0	45960	46960	
															AircraftAgent14	Point150,31,14,0	46960	47960	
		V150,31	97.9693	410	400/No 18/244	392/No 9/9	394/No 9/9	394/No 8/8	396/No 8/8	396/No 7/7	398/No 7/7	398/No 6/6	398/No 6/6	399/No 6/6	AircraftAgent16	Point150,31,16,0	47960	48960	410 / 399 Success: 97.3171 %
															AircraftAgent17	Point150,31,17,0	49960	50960	
															AircraftAgent18	Point150,31,18,0	48960	49960	
															AircraftAgent19	Point150,31,19,0	50960	49960	

d. Sending adjustments:

• If, during a given cycle, no further conflicts are detected, the adjustments calculated in the previous cycle are sent to the relevant AircraftAgents.

• If the maximum number of cycles is reached and some conflicts remain unresolved, the controller sends the most recent available adjustments regardless.

e. Receiving adjustments: Each AircraftAgent receives only the adjustments that concern it. Each adjustment specifies: the waypoint to be modified and the new altitude to be applied to that point.

4.2.6 Execution of this system using pre-generated test case inputs

Table 6 and Table 7 present the execution results of the system under test. For each pre-generated test set (Inputs0 (POP0), Inputs10 (POP10), Inputs40 (POP40), Inputs45 (POP45), Inputs70 (POP70), Inputs100 (POP100), and Inputs150 (POP150)), the two best test cases were selected and applied.

4.2.7 Evaluation of the coordination adopted in this system under test

This step constitutes the final phase of the approach. Now that the execution results have been obtained in the previous step, it is possible to evaluate the coordination strategy

implemented in the EATC. The objective is to assess the effectiveness of the strategy used to resolve conflicts between aircraft.

This evaluation is based on the number of cycles required to resolve all detected conflicts, according to a simple and interpretable scale presented in Table 8.

Table 9 clearly and concisely presents the evaluation results obtained for each test case, based on the grading scale defined in Table 8. This allows for a concrete assessment of the effectiveness of the coordination strategy, particularly in terms of how quickly the system resolves conflicts between aircraft.

Table 8. Coordination evaluation scale based on the number of cycles

Number of Cycles	Coordination Rating	Meaning
1	Excellent coordination	Conflicts are resolved very quickly with minimal adjustments.
2 to 5	Good coordination	Conflicts are resolved efficiently over several cycles.
6 to 9	Acceptable coordination	Conflicts are more complex and require more effort to resolve.
10 or more	✕ Coordination failure	Not all conflicts could be resolved despite multiple attempts.

Table 9. Coordination evaluation of the EATC system under test

Top 2 Test Cases Per Input	Fitness Score (%)	Planned Conflicts	Resolved Conflicts	Number of Cycles	Coordination Rating	Meaning
V0,90	33.5070	1	1	1	Excellent coordination	Conflicts are resolved very quickly with minimal adjustments
V0,49	33.5305	1	1	1	″	″
V10,27	53.7376	4	4	1	″	″
V10,25	59.1954	7	7	1	″	″
V40,11	76.9086	38	38	2	Good coordination	Conflicts are resolved efficiently over several cycles.
V40,1	77.5251	44	44	2	″	″
V45,6	79.2381	51	51	2	″	″
V45,23	80.2932	57	57	7	Acceptable coordination	Conflicts are more complex and require more effort to resolve.
V70,13	90.9035	184	176	10 or more	✕ Coordination failure	Not all conflicts could be resolved despite multiple attempts.
V70,18	91.0780	166	164	10 or more	″	″
V100,15	95.0592	286	279	10 or more	″	″
V100,17	95.3865	293	284	10 or more	″	″
V150,31	97.9693	410	399	10 or more	″	″
V150,24	98.1281	428	417	10 or more	″	″

5. DISCUSSION

The application of our approach to the case study demonstrated its effectiveness and yielded promising results. It enabled a thorough evaluation of the coordination mechanism under test, validating its ability to detect coordination flaws and assess the system's resilience. A major strength of the method lies in its generality: it can be applied regardless of the coordination mechanism used - whether planning, negotiation, or rule-based - and across both centralized and distributed MAS architectures. The strategic combination of mutation analysis and genetic algorithms is another key advantage. Mutation analysis deliberately injects faults to evaluate robustness under degraded conditions, while the genetic algorithm generates and evolves test cases, automatically prioritizing those most likely to expose conflicts. This evolutionary process helps uncover subtle vulnerabilities that may escape manual testing.

Future improvements can further enhance this framework. One promising direction involves the integration of machine learning [48, 49] and deep learning techniques [50, 51] to enrich the test case generation process. For instance, supervised learning could exploit historical conflict data to predict high-risk situations and guide the mutation process toward more critical test cases. Deep learning, particularly sequence-based models such as recurrent or transformer architectures, could capture temporal interaction patterns among agents, allowing the generation of conflict scenarios that more closely resemble real-world dynamics. Reinforcement learning could also be applied to iteratively refine test strategies, rewarding test inputs that reveal previously undetected vulnerabilities. These integrations would transform the framework into a self-adaptive testing system, capable of continuously improving as more data becomes available.

Another important avenue is the adaptation of the approach

for open MAS environments, where the number of agents and resources may change dynamically. Unlike static systems, open environments introduce uncertainty that challenges traditional conflict modelling. To address this, the approach could be extended with dynamic resource tracking and adaptive conflict detection algorithms, capable of recalibrating the system model in real time as new agents or resources appear. Probabilistic models could further be incorporated to anticipate coordination risks under uncertainty, ensuring that the testing process remains effective despite the fluid nature of open MAS. While these future directions promise to extend the applicability of the method, it is important to acknowledge its limitations. The primary challenge remains the preparatory modelling effort: the accuracy of conflict detection depends heavily on correctly identifying and modelling all coordination-sensitive resources. Any omissions can lead to undetected conflicts. In highly dynamic open MAS, this limitation becomes even more pronounced, as the evolving system state complicates accurate resource representation.

In summary, our approach provides a flexible and effective framework for testing coordination in MAS, with demonstrated scalability and robustness in safety-critical applications such as air traffic control. Its integration with advanced learning techniques and its extension to open environments represent concrete and impactful avenues for future research, bringing us closer to comprehensive and adaptive testing strategies for real-world MAS.

6. CONCLUSION

Testing coordination in multi-agent systems (MAS) is a challenging and often overlooked task, despite its critical role in ensuring coherent system behavior. This work introduced an innovative methodology that combines mutation analysis with parallel genetic algorithms, enabling the systematic generation of conflict-intensifying scenarios. The approach functions as a targeted stress test for coordination mechanisms, capable of uncovering weaknesses that traditional testing techniques frequently miss. The experimental validation on the En-route air traffic control (EATC) system provided concrete evidence of the method's effectiveness. Indeed, the obtained results clearly demonstrate both the scalability of the method and its practical relevance for safety-critical domains where reliability is paramount. Beyond this case study, the methodology offers a generic testing strategy applicable to centralized, distributed, or hybrid MAS architectures. Its ability to create realistic and conflict-intensive scenarios opens the door to a more rigorous assessment of coordination robustness. Looking ahead, several concrete directions for extension emerge:

- Adaptation to open MAS environments, where the number of agents and resources changes dynamically, reflecting real-world operational challenges that are inherently unstable and harder to model;
- Integration of machine learning and deep learning techniques, which could leverage past data to generate more diverse, realistic, and critical scenarios, thereby uncovering subtle coordination vulnerabilities;
- Application to distributed architectures, where decentralized coordination introduces specific challenges related to communication, synchronization, and fault tolerance;
- Scaling to larger agent populations, to simulate dense and complex environments such as high-traffic airspaces or large-scale industrial systems.

In summary, this study lays the foundations of a robust and extensible framework for testing MAS coordination under realistic and adverse conditions. By combining mutation analysis and genetic algorithms, the proposed approach not only identifies coordination vulnerabilities but also contributes to the design of more resilient, adaptive, and trustworthy multi-agent systems capable of thriving in complex, real-world environments.

REFERENCES

[1] Wooldridge, M. (2009). An Introduction to Multiagent Systems. John Wiley & Sons.

[2] Ferber, J. (1997). Les systÈmes Multi-Agents: Vers une Intelligence Collective. InterEditions.

[3] Dorri, A., Kanhere, S.S., Jurdak, R. (2018). Multi-agent systems: A survey. IEEE Access, 6: 28573-28593. https://doi.org/10.1109/ACCESS.2018.2831228

[4] Cortés, J., Egerstedt, M. (2017). Coordinated control of multi-robot systems: A survey. SICE Journal of Control, Measurement, and System Integration, 10(6): 495-503. https://doi.org/10.9746/jcmsi.10.495

[5] Kernbach, S. (2008). Structural Self-Organization in Multi-Agents and Multi-Robotic Systems. Logos Verlag Berlin GmbH.

[6] Satunin, S., Babkin, E. (2014). A multi-agent approach to intelligent transportation systems modeling with combinatorial auctions. Expert Systems with Applications, 41(15): 6622-6633. https://doi.org/10.1016/j.eswa.2014.05.015

[7] Mahela, O.P., Khosravy, M., Gupta, N., Khan, B., et al. (2020). Comprehensive overview of multi-agent systems for controlling smart grids. CSEE Journal of Power and Energy Systems, 8(1): 115-131. https://doi.org/10.17775/CSEEJPES.2020.03390

[8] Anvari-Moghaddam, A., Rahimi-Kian, A., Mirian, M.S., Guerrero, J.M. (2017). A multi-agent based energy management solution for integrated buildings and microgrid system. Applied Energy, 203: 41-56. https://doi.org/10.1016/j.apenergy.2017.06.007

[9] Shavandi, A., Khedmati, M. (2022). A multi-agent deep reinforcement learning framework for algorithmic trading in financial markets. Expert Systems with Applications, 208: 118124. https://doi.org/10.1016/j.eswa.2022.118124

[10] Li, X., Mao, W., Zeng, D., Wang, F.Y. (2008). Agent-based social simulation and modeling in social computing. In International Conference on Intelligence and Security Informatics, Berlin, pp. 401-412. https://doi.org/10.1007/978-3-540-69304-8_41

[11] Tessier, C., Chaudron, L., Müller, H.J. (2005). Conflicting Agents: Conflict Management in Multi-Agent Systems (Vol. 1). Springer Science & Business Media.

[12] Cao, Y., Yu, W., Ren, W., Chen, G. (2012). An overview of recent progress in the study of distributed multi-agent coordination. IEEE Transactions on Industrial Informatics, 9(1): 427-438. https://doi.org/10.1109/TII.2012.2219061

[13] Berndt, J.O., Herzog, O. (2011). Efficient multiagent coordination in dynamic environments. In 2011

IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology, Lyon, France, 2: 188-195. https://doi.org/10.1109/WI-IAT.2011.63

[14] Tonino, H., Bos, A., de Weerdt, M., Witteveen, C. (2002). Plan coordination by revision in collective agent based systems. Artificial Intelligence, 142(2): 121-145. https://doi.org/10.1016/S0004-3702(02)00273-4

[15] Albrecht, M. (2009). Supply Chain Coordination Mechanisms: New Approaches for Collaborative Planning (Vol. 628). Springer Science & Business Media.

[16] Benameur, H., Chaib-Draa, B., Kropf, P. (2002). Multi-item auctions for automatic negotiation. Information and Software Technology, 44(5): 291-301. https://doi.org/10.1016/S0950-5849(01)00216-6

[17] Lopes, F., Coelho, H. (2014). Negotiation and Argumentation in Multi-Agent Systems: Fundamentals, Theories, Systems and Applications. Bentham Science Publishers.

[18] Chen, C., Xu, C. (2018). A negotiation optimization strategy of collaborative procurement with supply chain based on Multi-Agent System. Mathematical Problems in Engineering, 2018(1): 4653648. https://doi.org/10.1155/2018/4653648

[19] Doostmohammadian, M., Aghasi, A., Pirani, M., Nekouei, E., et al. (2025). Survey of distributed algorithms for resource allocation over multi-agent systems. Annual Reviews in Control, 59: 100983. https://doi.org/10.1016/j.arcontrol.2024.100983

[20] Tang, X., Zeng, T., Tan, Y., Ding, B.X. (2020). Conflict analysis based on three-way decision theoretic fuzzy rough set over two universes. Ingénierie des Systèmes d'Information, 25(1): 75-82. https://doi.org/10.18280/isi.250110

[21] Fatima, S.S., Wooldridge, M., Jennings, N.R. (2004). An agenda-based framework for multi-issue negotiation. Artificial Intelligence, 152(1): 1-45. https://doi.org/10.1016/S0004-3702(03)00115-2

[22] Kraus, S., Wilkenfeld, J. (2013). Strategic Negotiation in Multiagent Environments. In Coordination Theory and Collaboration Technology, pp. 93-124. Psychology Press.

[23] Hady, M.A., Hu, S., Pratama, M., Cao, Z., Kowalczyk, R. (2025). Multi-agent reinforcement learning for resources allocation optimization: A survey. Artificial Intelligence Review, 58(11): 354. https://doi.org/10.1007/s10462-025-11340-5

[24] Crouzet, Y., Thévenod-Fosse, P., Waeselynck, H. (1998). Validation du test du logiciel par injection de fautes: l'outil SESAME. In 11eme Colloque National de Fiabilité & Maintenabilité, pp. 551-559.

[25] Fabbri, S.C.P.F., Maldonado, J.C., Sugeta, T., Masiero, P.C. (1999). Mutation testing applied to validate specifications based on statecharts. In Proceedings 10th International Symposium on Software Reliability Engineering (Cat. No. PR00443), Boca Raton, FL, USA, pp. 210-219. https://doi.org/10.1109/ISSRE.1999.809326

[26] Mouelhi, T., Le Traon, Y., Baudry, B. (2009). Utilisations de la mutation pour les tests de contrôle d'accès dans les applications. In SARSSI 2009: 4ème conférence sur la sécurité des architectures réseaux et des systèmes d'information.

[27] Pizzoleto, A.V., Ferrari, F.C., Offutt, J., Fernandes, L., Ribeiro, M. (2019). A systematic literature review of techniques and metrics to reduce the cost of mutation testing. Journal of Systems and Software, 157: 110388. https://doi.org/10.1016/j.jss.2019.07.100

[28] Rashmi, V., Mohan, C.C., Bhavani, V., Anuradha, Y., Kumar, L.K., Sowjanya, B., Kumar, K.S., Rao, K.K., Pallikonda, A.K. (2023). Experimental investigations to fault reduction system for software applications. Ingénierie des Systèmes d'Information, 28(3): 567-573. https://doi.org/10.18280/isi.280304

[29] Menassel, Y., Marir, T., Mokhati, F., Gupta, V. (2025). Operational profile-based test case generation for normative MAS. Journal of Systems and Software, 112553. https://doi.org/10.1016/j.jss.2025.112553

[30] Harshbarger, S., Collins, M., Heckle, R.R. (2024). Transforming the testing and evaluation of autonomous Multi-Agent Systems: Introducing in-situ testing via distributed ledger technology. The ITEA Journal of Test and Evaluation, 45(1). https://doi.org/10.61278/itea.45.1.1003

[31] Guassmi, D., Dehimi, N.E.H., Derdour, M. (2023). A state of art review on testing open multi-agent systems. In Novel & Intelligent Digital Systems Conferences, Athens, Greece, pp. 262-266. https://doi.org/10.1007/978-3-031-44097-7_28

[32] Guassmi, D., Dehimi, N.E.H., Derdour, M., Kouzou, A. (2024). Using machine learning techniques for multi-agent systems testing. In Artificial Intelligence and Its Practical Applications in the Digital Economy. I2COMSAPP 2024. Lecture Notes in Networks and Systems, pp. 194-201. https://doi.org/10.1007/978-3-031-71426-9_16

[33] Gray, P. (2019). Automated Mutation Testing for Concurrent Software. MSc dissertation. Department of Computer and Information Sciences, University of Strathclyde, Glasgow, Scotland, United Kingdom. https://local.cis.strath.ac.uk/wp/extras/msctheses/papers/strath_cis_publication_2742.pdf.

[34] Omicini, A., Ricci, A., Viroli, M. (2008). Artifacts in the A&A Meta-Model for Multi-Agent Systems. Autonomous Agents and Multi-Agent Systems, 17(3): 432-456. https://doi.org/10.1007/s10458-008-9053-x

[35] Dehimi, N.E.H., Mokhati, F., Badri, M. (2015). Testing HMAS-based applications: An ASPECS-based approach. Engineering Applications of Artificial Intelligence, 46: 25-33. https://doi.org/10.1016/j.engappai.2015.09.013

[36] Dehimi, N.E.H., Mokhati, F. (2019). A novel test case generation approach based on auml sequence diagram. In 2019 International Conference on Networking and Advanced Systems (ICNAS), Annaba, Algeria, pp. 1-4. https://doi.org/10.1109/ICNAS.2019.8807874

[37] Winikoff, M. (2017). BDI agent testability revisited. Autonomous Agents and Multi-Agent Systems, 31(6): 1094-1132. https://doi.org/10.1007/s10458-016-9356-2.

[38] Gonçalves, E.M.N., Machado, R.A., Rodrigues, B.C., Adamatti, D. (2022). CPN4M: Testing multi-agent systems under organizational model Moise+ using colored Petri nets. Applied Sciences, 12(12): 5857. https://doi.org/10.3390/app12125857

[39] Rehman, M.S.U., Nadeem, A., Sindhu, M.A. (2019). Towards automated testing of multi-agent systems using prometheus design models. The International Arab

Journal of Information Technology, 16: 54-65. https://dblp.uni-trier.de/db/journals/iajit/iajit16.html#RehmanNS19.
[40] Huang, Z., Alexander, R., Clark, J. (2014). Mutation testing for Jason agents. In International Workshop on Engineering Multi-Agent Systems, pp. 309-327. https://doi.org/10.1007/978-3-319-14484-9_16
[41] Dehimi, N.E.H., Benkhalef, A.H., Tolba, Z. (2022). A novel mutation analysis-based approach for testing parallel behavioural scenarios in Multi-Agent Systems. Electronics, 11(22): 3642. https://doi.org/10.3390/electronics11223642
[42] Savarimuthu, S., Winikoff, M. (2013). Mutation operators for the GOAL agent language. In International Workshop on Engineering Multi-Agent Systems, Berlin, pp. 255-273. https://doi.org/10.1007/978-3-642-45343-4_14
[43] Dehimi, N.E.H., Tolba, Z., Medkour, M., Hadjadj, A., Galland, S. (2025). Improving testing of multi-agent systems: An innovative deep learning strategy for automatic, scalable, and dynamic error detection and optimisation. Bulletin of the Polish Academy of Sciences Technical Sciences, pp. e154062-e154062. https://doi.org/10.24425/bpasts.2025.154062
[44] Kadri, R.L., Boctor, F.F. (2018). An efficient genetic algorithm to solve the resource-constrained project scheduling problem with transfer times: The single mode case. European Journal of Operational Research, 265(2): 454-462. https://doi.org/10.1016/j.ejor.2017.07.027
[45] Zeng, L., Wang, B., Tian, J.W., Wang, Z. (2021). Threat impact analysis to air traffic control systems through flight delay modeling. Computers & Industrial Engineering, 162: 107731. https://doi.org/10.1016/j.cie.2021.107731
[46] Ming, Y. (2024). Research on RVSM flight test data processing and real-time monitoring for civil aircraft. In International Conference on Aerospace System Science and Engineering, pp. 375-381. https://doi.org/10.1007/978-981-96-2440-9_36
[47] Kim, J., Nam, G., Min, D., Kim, N.M., Lee, J. (2023). Safety risk assessment based minimum separation boundary for UAM operations. In 2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC), Barcelona, Spain, pp. 1-8. https://doi.org/10.1109/DASC58513.2023.10311141
[48] Khatibsyarbini, M., Isa, M.A., Jawawi, D.N., Shafie, M.L.M., Wan-Kadir, W.M.N., Hamed, H.N.A., Suffian, M.D.M. (2021). Trend application of machine learning in test case prioritization: A review on techniques. IEEE Access, 9: 166262-166282. https://doi.org/10.1109/ACCESS.2021.3135508
[49] Witten, I.H., Frank, E. (2002). Data mining: Practical machine learning tools and techniques with Java implementations. Acm Sigmod Record, 31(1): 76-77. https://doi.org/10.1145/507338.507355
[50] LeCun, Y., Bengio, Y., Hinton, G. (2015). Deep learning. Nature, 521(7553): 436-444. https://doi.org/10.1038/nature14539
[51] Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S. (2011). A systematic literature review on fault prediction performance in software engineering. IEEE Transactions on Software Engineering, 38(6): 1276-1304. https://doi.org/10.1109/TSE.2011.103