People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

Abbes Laghrour University of Khenchela

Faculty of Sciences and Technology

Department of Mathematics and Computer Science

Order Number:
Series:

Thesis for the degree of

Doctorate (LMD) in Computer Science

Specialty: Artificial Intelligence

Research laboratory: Ingénierie des Connaissances et Sécurité Informatique (ICOSI)

Toward a New Coordination Mechanism for Multi-Agent Systems Based on a Metaheuristic Approach

---

Presented by

Laassami Ferial

Jury Members:

President	Dr. Ledmi Makhlouf	University of Khenchela
Supervisor	Prof. Souidi Mohammed el Habib	University of Khenchela
Co-Supervisor	Dr. Ledmi Abdeldjalil	University of Khenchela
Examinators	Prof. Maarouk Toufik Messaoud	University of Khenchela
	Prof. Mokhati Farid	University of Oum El Bouaghi
	Dr. Marir Toufik	University of Oum El Bouaghi

ABSTRACT

Multi-agent systems (MAS) are now a critical tool for tackling complex problems in dynamic environments. Because of their high dimensionality and the computational complexity of these environments, it remains a significant challenge to coordinate these agents and plan their tasks and paths. The ability of metaheuristic approaches to find optimal solutions has often made them a promising option, but they struggle with issues like local optima, premature convergence, and a limited balance between exploration and exploitation. This thesis tackles these challenges by developing and evaluating the recent enhanced and hybridized metaheuristic algorithms tailored for solving MAS path planning and task coordination. The main contributions of this thesis include a new method for Particle Swarm Optimization (PSO), named IB-PSO, which presents a novel approach to dynamically increase the inertia weight (IW) parameter inspired by the Butterworth function. It enhances the balance between exploration and exploitation of the PSO algorithm. This method is designed to solve the MAS path-planning problem. Furthermore, the research provides a hybrid approach named CSO, which combines the Crow Search Algorithm (CSA) and PSO. It is particularly designed for MAS path planning with collision avoidance, leveraging the strengths of CSA in exploration and PSO in exploitation, using the velocity regulating method. The performance of these proposed methods is evaluated and compared with existing enhanced techniques. Ultimately, this research introduces new optimization methods that help advance the field of MAS coordination.

Keywords: Multi-agent systems, Coordination mechanism, Metaheuristics, Particle Swarm Optimization, Crow Search Algorithm.

ملخـــــــص

تُعد الأنظمة متعددة الوكلاء (MAS) في الوقت الراهن أداة حاسمة لمعالجة المشكلات المعقدة في البيئات الديناميكية. ونظرًا للأبعاد العالية والتعقيد الحسابي لهذه البيئات، لا يزال تنسيق هؤلاء الوكلاء وتخطيط مهامهم ومساراتهم يمثل تحديًا كبيرًا. لطالما كانت قدرة نهج الخوارزميات فوق الاستدلالية (Metaheuristics) على إيجاد حلول مثلى خيارًا واعدًا، إلا أنها تواجه صعوبات تتعلق بمشكلات مثل الحلول المثلى المحلي (Local Optima)، والتقارب المبكر، والتوازن المحدود بين الاستكشاف (Exploration) والاستغلال (Exploitation). تتناول هذه الأطروحة هذه التحديات من خلال تطوير وتقييم خوارزميات فوق الاستدلالية حديثة، محسنة وهجينة، مصممة خصيصًا لحل مشكلات تخطيط المسارات وتنسيق المهام في الأنظمة متعددة الوكلاء. تشمل المساهمات الرئيسية لهذه الأطروحة طريقة جديدة لتحسين سرب الجسيمات (PSO)، تسمى **IB-PSO**، والتي تقدم نهجًا مبتكرًا لزيادة معلمة وزن القصور الذاتي (Inertia Weight) ديناميكيًا باستلهام من دالة (Butterworth function)؛ مما يعزز التوازن بين الاستكشاف والاستغلال في خوارزمية PSO، وقد صُممت هذه الطريقة لحل مشكلة تخطيط المسارات في الأنظمة متعددة الوكلاء. علاوة على ذلك، يقدم البحث نهجًا هجينًا يسمى **CSO**، يجمع بين خوارزمية البحث عن الغراب (CSA) وخوارزمية تحسين سرب الجسيمات(PSO) . صُمم هذا النهج خصيصًا لتخطيط مسارات الأنظمة متعددة الوكلاء مع تجنب الاصطدام، مستفيدًا من نقاط قوة خوارزمية CSA في الاستكشاف وقوة PSO في الاستغلال، وذلك باستخدام طريقة تنظيم السرعة. تم تقييم أداء هذه الطرق المقترحة ومقارنتها بالتقنيات المحسنة الموجودة حاليًا. وفي الختام، يقدم هذا البحث طرق تحسين جديدة تساهم في تطوير مجال التنسيق في الأنظمة متعددة الوكلاء.

الكلمات المفتاحية: الأنظمة متعددة الوكلاء، آلية التنسيق، الخوارزميات فوق الاستدلالية، تحسين سرب الجسيمات، خوارزمية البحث عن الغراب.

RÉSUMÉ

Les systèmes multi-agents (SMA) constituent désormais un outil essentiel pour aborder des problèmes complexes dans des environnements dynamiques. En raison de la dimensionnalité élevée et de la complexité computationnelle de ces environnements, la coordination de ces agents ainsi que la planification de leurs tâches et de leurs trajectoires demeurent un défi majeur. Bien que la capacité des approches métaheuristiques à trouver des solutions optimales en fasse souvent une option prometteuse, elles se heurtent à des obstacles tels que les optima locaux, la convergence prématurée et un équilibre limité entre l'exploration et l'exploitation. Cette thèse relève ces défis en développant et en évaluant de nouveaux algorithmes métaheuristiques améliorés et hybrides, adaptés à la résolution de la planification de trajectoires et à la coordination des tâches dans les SMA. Les principales contributions de cette thèse incluent une nouvelle méthode d'optimisation par essaim de particules (PSO), nommée IB-PSO, qui présente une approche novatrice pour augmenter dynamiquement le paramètre du poids d'inertie (IW) en s'inspirant de la fonction de Butterworth. Elle améliore ainsi l'équilibre entre l'exploration et l'exploitation de l'algorithme PSO. Cette méthode est conçue pour résoudre le problème de planification de trajectoires multi-agents. De plus, la recherche propose une approche hybride nommée CSO, combinant l'algorithme de recherche de corbeaux (CSA) et le PSO. Elle est spécifiquement conçue pour la planification de trajectoires multi-agents avec évitement de collisions, en exploitant les forces du CSA pour l'exploration et du PSO pour l'exploitation, grâce à une méthode de régulation de la vitesse. Les performances de ces méthodes proposées sont évaluées et comparées aux techniques améliorées existantes. En fin de compte, cette recherche introduit de nouvelles méthodes d'optimisation qui contribuent à l'avancement du domaine de la coordination des SMA.

Mots-clés: Systèmes multi-agents, mécanisme de coordination, métaheuristiques, optimisation par essaim de particules, algorithme de recherche de corbeaux.

Acknowledgments

First and foremost, I offer my deepest gratitude to God the Almighty and Merciful. It is by His grace, strength, and patience that I have been able to navigate this journey and bring this work to fruition.

I would like to express my profound gratitude to my supervisor, **Dr. Mohammed El Habib Souidi** of the University of Khenchela. I am deeply indebted to him for his unwavering availability, his insightful guidance, and the confidence he placed in me. His vast experience and scientific rigor have not only shaped this research but have also been a profound source of inspiration throughout my academic path.

My sincere thanks also go to my co-supervisor, **Dr. Abdeldjalil Ledmi**. Our discussions were a vital catalyst for this work, and I am grateful for his keen perspectives and the high quality of his advice, which helped refine my vision.

I am also grateful to all the members of the **ICOSI laboratory**. Their collaborative spirit and the stimulating environment they provide were essential to the achievement of this research. I wish to extend my sincere appreciation to the honorable **members of the jury** for dedicating their time and expertise to evaluate and review this work; their participation is a true honor.

A special, heartfelt tribute goes to my **late grandparents**; though they are no longer with us, their prayers and memory remain the light that guides my path. I hope this work honors the values they instilled in me.

A very special mention goes to my beloved nieces, **Mary** and **Mei**, whose innocent joy and smiles provided the most beautiful distractions and lightened my heart during the long hours of work.

I dedicate this achievement to my family, the pillars of my life. To my **father**, my **mother**, and my **beloved siblings**—each by their name and their cherished place in my heart—thank you for your boundless love and for being my constant source of strength. A special word of appreciation goes to my **aunt Saida** and **Ala** for their kindness and support. I also extend my gratitude to my entire family for their collective encouragement throughout this long journey.

On a deeply personal note, I offer my heartfelt thanks to my partner, **Djamel-Eddine**, for being my greatest believer. I am profoundly grateful for his support and the sacrifices he made to help me reach this goal.

Finally, I cannot forget my **dear friends**. Thank you for your constant encouragement and for always being there when I needed a reminder of why I started.

Contents

List of Figures

List of Tables

GENERAL INTRODUCTION

The widespread use of multi-agent systems (MAS) has created opportunities for solving complex problems across various fields, including transportation systems, robotics, and sensor networks. A MAS is a group of independent and smart individual agents working together. Each agent can make its own decisions and interact with each other and their surroundings to reach a shared goal. Additionally, it adapts to changes in the environment to support the overall system's success. However, coordinating these agents and planning their paths or tasks and making them all work together presents a significant challenge. Significantly, it is difficult to assign specific tasks to agents and manage their collective decisions and actions to achieve a common goal. Moreover, it becomes more challenging to generate a safe, collision-free path for each agent from its start point to its targeted point. This must take into account static and dynamic obstacles as well as the dynamic position of all other agents in the system. The challenge of coordinating MAS scales with the number of agents and the dimensionality of its environment. As the number of agents in a system increases, the potential for interactions and conflicts grows. This makes it difficult for traditional algorithms to find an optimal solution across all agents. Also, a high-dimensional environment has the risk of having many variables like obstacles, changes, and uncertain conditions. Consequently, traditional optimization methods that depend on exhaustive search or deterministic methods become too hard to compute. They rely on a precise mathematical representation of the environment and the agents. Also, they do not use randomness. They become impractical, requiring too many resources to be useful to provide a real-time solution as the environmental complexity increases. The need for better performance has driven the need for more advanced, scalable approaches like decentralized control and swarm intelligence algorithms.

To address these challenges, metaheuristics have emerged as promising approaches. These methods are inspired by collective behaviors observed in nature. They are capable of finding near-optimal solutions to complex optimization problems even with an incomplete or uncertain mathematical model of the environment. The most important characteristic of these methods is their ability to balance between phases of the exploration of new solutions and exploitation of the current ones in the search space. Also, they demonstrate flexibility, scalability, computational efficiency, and hybridization. All this makes them ideal candidates for MAS path planning and task coordination. Despite how effective they are, metaheuristic performance is strongly influenced by their parameters (parameter sensitivity) and can suffer from limitations caused by issues such as premature convergence or trapping in local optima. To address this, the objective of this thesis is to improve and hybridize existing metaheuristics to handle the complexities of path planning in MAS. The contributions of this research focus on developing new optimization approaches and demonstrating their effectiveness through a comprehensive evaluation and comparison with the other current methods.

To address this, the thesis is structured as follows:

Chapter 1 represents a general introduction, which provides an overview, outlining the scientific foundation and the motivation for this work.

Chapter 2 gives a broad examination of the existing research on the application of metaheuristics to MAS task coordination and path planning problems. This chapter analyzes different approaches and their applications to understand the trade-offs between key performance indicators. This provides the necessary background for the rest of the thesis.

Chapter 3 provides a new improved version of the Particle Swarm Optimization (PSO) algorithm. It presents a novel method for calculating the *IW* parameter, named the inversed Butterworth function, inspired by the Butterworth function. The aim of this method is to enhance the balance between exploration and exploitation to solve the MAS path-planning problem. The performance of this algorithm is evaluated and compared with other existing and improved *IW* versions of PSO.

Chapter 4 solves the problem of MAS path planning with collision avoidance. This chapter provides a hybridization of the Crow Search Algorithm (CSA) and the PSO algorithm, named CSO for Crow Swarm Intelligence. This process aims to exploit the strength of CSA in exploration and the strength of PSO in exploitation. Also, it uses a velocity regulation mechanism to handle collisions between agents. The results of this hybridization are compared with recent methods based on both CSA and PSO.

Finally, the thesis will conclude with a general conclusion recapping the main contributions and proposing future research directions.

CHAPTER 1:

INTRODUCTION

1.1 Introduction

The growing complexity of systems in the real world raised interest in distributed artificial intelligence, especially in the paradigm of MAS. These systems are characterized by decentralized decision-making, dynamic interactions, and agents ought to effectively coordinate their actions in shared environments. Tackling these difficulties means dealing with problems of deep connection of task planning and path planning despite uncertainty and resource constraints. For these, advanced optimization techniques are necessary and must be robust and cost-effective. For this, this chapter outlines key concepts in MAS, coordination of path and tasks, and metaheuristics, forming the foundation for the proposed approaches. Its primary goal is to establish the theoretical and methodological work for designing intelligent coordination strategies within MAS frameworks. The chapter is organized as follows:

Section 1 offers an introduction. The Section 2 introduces the foundations of MAS, including agent definition, properties, and topologies. This is on one hand. On the other hand, it defines the environment where an agent operates, focusing on characteristics and a formal model. The last part of this section detailed the MAS concept, emphasizing its architectures, basic interactions, taxonomy and different applications. Moreover, Section 3 explores the coordination of task and path planning, emphasizing their different methods and collision avoidance concept. Lastly, Section 4 presents an overview of metaheuristic algorithms, highlighting their relevance to MAS coordination problems and their potential for hybridization in complex search spaces. Section 5 provides a summary of the final thoughts.

1.2 Multi-agent systems

1.2.1 Agent

1.2.1.1 Definition of an agent

MAS technologies are considered one of the most promising paradigms [1] for designing distributed frameworks. They are distributed computing systems [2] composed of a group of autonomous agents that work together or compete to solve complex problems. Typically, these entities are the key components in the design of an MAS. The performance of an MAS is heavily influenced by the capabilities and interactions of its individual agents. It defines how well its agents work together. Firstly, according to [3], the agents are collaboratively solving tasks. They offer flexibility due to their ability to learn and make decisions. They use interactions with neighboring agents or the environment to learn new contexts and actions, and then they use their knowledge to solve their assigned tasks. The MAS requires addressing complex challenges like coordination [4] among agents. Secondly, the study [5], presents two main uses of the term "agent": The broad, weak notion of an agent describes autonomous,

interactive, reactive, and proactive systems without emphasizing human similarity. In addition, the stronger, more specific view sees agents as systems modelled with human-like mental states and attributes, sometimes represented visually for better understanding. Moreover, an agent can be a software program, not just a human or physical entity. Another definition in [6] provides that the agent is a software component capable of performing tasks on behalf of its user. It is a meta-term that covers various agent types, reducing philosophical arguments and allowing for the recasting of old software as agent-based software.

According to [2], it is typically associated with a sensor that provides only a partial view of its environment. Knowing that the environment refers to the physical or virtual world in which an agent operates, this last equipped with its localized perception, allowing it to operate and make decisions. Subsequently, it acts based on its perceptions and internal state. It uses effectors to execute decisions made by their individual, limited viewpoints. Furthermore, the agent's control is not absolute; it possesses partial and limited control over its environment. Its repertoire includes a predefined set of actions. It operates by executing these actions based on its programming and immediate perceptions of its environment. That means that it will not perform any actions that have not been explicitly defined in its initial configuration. Briefly, an agent is an entity that can sense its environment by physical sensors in the case of a physical agent or software sensors in the case of a software agent.

While a basic agent might react to its environment based on its design specifications, an intelligent agent [7] (as shown in Figure 1.1) can think and learn. It is a more specific type of agent that exhibits characteristics commonly associated with intelligence. An intelligent agent is one that can perform flexible autonomous actions to meet its design objectives, where flexibility [7] includes three key aspects: reactivity, pro-activeness and social ability.

- *Reactivity:* means that intelligent agents sense and react quickly to changes to achieve their goals.
- *Pro-activeness:* signifies that they take initiative to meet their objectives.
- *Social ability:* refers to their interactions with other agents and humans to fulfill their purpose.

1.2.1.2 Properties of an agent

Agents and intelligent agents typically possess a range of additional properties [8], the utility of which is for solving particular problems. In addition to the previously mentioned characteristics (perception, action, reactivity…), it can be mentioned:

- *Learning/Adaptability:* the agents can learn from experience and learn then adjust their actions, improving performance and adapting to new environments.
- *Autonomy:* autonomous agents operate independently, making decisions based on internal state and environment, without human intervention, allowing them to initiate actions without explicit command.
- *Rationality:* means finding an optimal solution that works well enough.
- *Mobility:* the agent possesses the unique ability to move autonomously within a network.
- *Benevolence:* the agent is willing to help the other agents; it prioritizes benefiting other agents or the overall system.
- *Veracity:* the agent aims to provide honesty and reliability in its interactions and data sharing.
- *Goal-Oriented:* agents are made to accomplish particular and specific goals or objectives.

1.2.1.3 Typologies of an agent

To enhance comprehension of the diverse functionalities and behaviors of agents, it is crucial to examine their core topology. The agent topology outlined in the research [6] outlines a multi-dimensional typology of software agents, categorizing them according to their key characteristics and roles, including:

- *Mobility*: it divides agents into Static, which are fixed-location systems that perform tasks in their local environment, reducing network load and enabling asynchronous operation. They are easier to implement and secure. Vs. Mobile agents, which can move autonomously across networks, enabling distributed environments.
- *Cognitive approach*: it refers to how the agents make decisions. It divides agents into two categories: deliberative (symbolic reasoning), which use symbolic reasoning and plan before acting, vs. reactive (stimulus-response), which respond immediately to environmental changes without internal reasoning.
- *Core attributes*: truly intelligent agents should possess three core attributes: autonomy, learning, and cooperation. Autonomy allows agents to make decisions based on targets and adjust an internal state independently. Learning improves performance over time through observation and feedback. Cooperation allows agents to interact and coordinate with others to achieve shared goals.

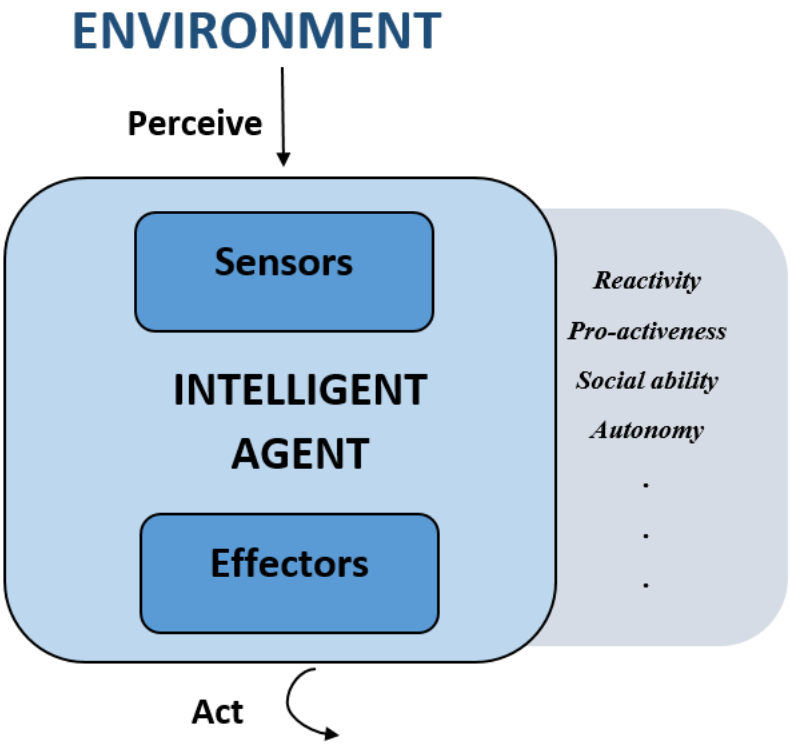

Figure 1.1: Architecture of an intelligent agent in a MAS

To summarize, an agent, particularly an intelligent agent, is an autonomous unit capable of sensory interaction with its environment and acting upon it, often reflecting the learning of new information, reasoning, and goal-directed behavior to accomplish its design objectives effectively and autonomously. Subsequently, the environment in which an agent operates is discussed.

1.2.2 Notion of an environment

1.2.2.1 Definition and characteristics of an environment

The environment of an intelligent agent is the external context that influences its perceptions and actions, and its nature influences its design and complexity, shaping how it perceives its surroundings and plans its actions. Consequently, according to [7], the environment's characteristics can be described along several key dimensions (as shown in Figure 1.2):

- *Accessible vs. inaccessible*: accessible environments offer complete, accurate observations, while inaccessible environments limit observations. The agents perceive part of the environment.

- *Deterministic vs. non-deterministic*: deterministic environments lead to a single predictable outcome, while non-deterministic or stochastic environments include uncertainty due to inherent randomness or lack of complete knowledge.
- *Episodic vs. Sequential*: in episodic environments, agents' experiences are divided into independent episodes, allowing them to act based on the current episode. In sequential environments, agents must consider past actions and their consequences, requiring memory and planning.
- *Static vs. Dynamic*: a static environment simplifies planning, while dynamic environments require real-time adaptation, as they can change independently of an agent's actions, even during decision-making.
- *Discrete vs. Continuous*: discrete environments have a finite set of percepts and actions, simplifying modelling, while continuous environments have infinite states or actions, often requiring approximation for transitions and interaction
- *Single Agent vs. Multi-Agent*: in a single-agent environment, one agent interacts, while in multi-agent environments, multiple agents interact simultaneously, presenting challenges like coordination and competition.

Ultimately, these environmental dimensions play a crucial role in designing the architecture of an intelligent agent, influencing how it processes information, makes decisions, and adjusts its behaviour to achieve its goals with efficiency and precision.

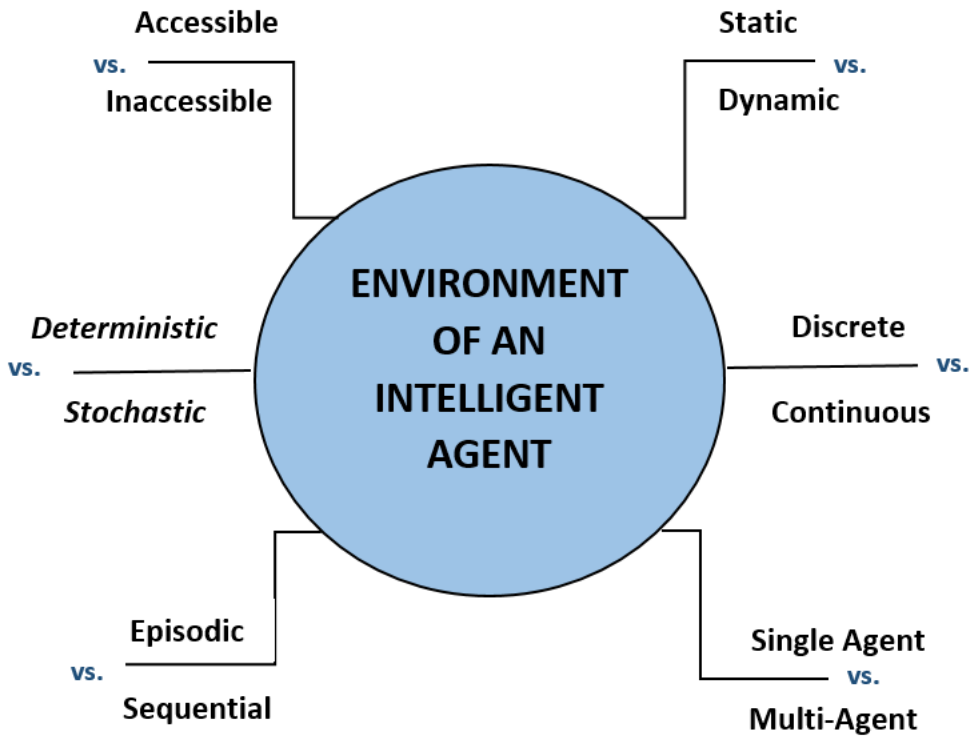

Figure 1.2: Environment of an intelligent agent

1.2.2.2 Formal model of an environment

However, the formal model of the environment in which an intelligent agent operates is defined in [7] by $S=\{s_1,s_2,s_3,\ldots\}$ of all possible environment states. Where the transitive function that describes how the environment changes in response to actions is the following equation 1.1 [7]:

$$env : S \times A \rightarrow P(S) \qquad (1.1)$$

This function takes a current state $s \in S$ and an action $a \in A$ where $A=\{a_1,a_2,a_3,\ldots\}$, and provides a set of possible successor states. If each result is a singleton, the environment is *deterministic*; else, it is *non-deterministic*.

Furthermore, an agent is defined by two primary functions named Observation Function and Agent Decision Function, presented in equations 1.2 and 1.3, respectively. The first one translates the current state of the environment into the perceptual inputs P available to the agent. Consequently, the second one reflects how the agent makes decisions based on its experience, from perceptual input sequences to actions.

$$see : S \rightarrow P \qquad (1.2)$$

$$action : P^* \rightarrow A \qquad (1.3)$$

Concisely, the agent uses *see* to observe the environment, creates a sequence of percepts, and chooses an *action*, while the environment reacts using the *env* function to transition to a new state.

1.2.3 Multi-agent system

1.2.3.1 Definition of multi-agent systems

MASs are a focus of research in artificial intelligence, where multiple intelligent agents interact within a shared environment to achieve common or conflicting goals. However, due to the evolving and diverse nature of this field of research, there are many definitions for MAS. This variety in definitions shows MAS's depth and flexibility, making it an exciting area for further study and development. Starting from the definition in [9], the MASs can be computer programs or people, they are the entities that perform knowledge representation field, which focuses on representing and reasoning about environments with different properties to make decisions about how to act in them. The simplest scenario for knowledge representation is when there is only one agent in the environment. However, when there are multiple agents, the situation becomes more interesting and challenging. Another work [10] holds that MASs are used to study complex systems, such as human societies, computer networks, neural tissue, and cell biology. In addition, MAS can be subject to formal analyses. It can be decomposed into two aspects: agents and interactions among agents. Another research [11] describes MAS as a system of multiple decision-making entities interacting and cooperating to achieve optimal global performance, with autonomous, adaptable, learning, and proactive agents communicating with each other and the environment. Another definition in [12]; is that MAS, a technology consisting of homogeneous or heterogeneous agents, is essential for effectively exploiting diverse online information sources. It can also provide a framework for the development of large, complex, and robust distributed information processing systems that utilize organized behavior efficiencies.

1.2.3.2 Key interactions of multi-agent systems

By general consensus, the MASs are defined as systems with multiple autonomous agents, with each acting as a decision-making entity in a shared environment, capable of *communication* [13], *coordination* [14], and *cooperation* [12]. These lasts can be summarized as key interaction types in Table 1.1 as follows:

Communication	It is a crucial mechanism for agents to exchange information, coordinate actions, and negotiate decisions, facilitating interaction protocols and supporting collaborative behaviors in dynamic environments, ensuring consistency and efficient problem-solving.
Coordination	It is the process of aligning the actions of multiple agents to avoid conflicts and achieve coherent global behavior. At the same time, its mechanisms are the concrete methods or protocols that facilitate the realization of this alignment. Coordination is crucial in distributed environments, where agents manage dependencies, resolve resource contentions, and synchronize activities without centralized control.
Cooperation	It requires agents to cooperate towards a shared objective or goal, where each agent provides its distinct skills and resources to achieve outcomes that would be unachievable by a single entity. According to [12] , cooperation is not merely about communication and coordination but also about goal alignment, shared plans, and prioritizing the collective good above individual gain in agent behavior.

Table 1.1: Core interaction concepts in a MAS

Beyond that, while communication, coordination, and cooperation provide collaborative interaction mechanisms, the *competition* provides a self-interested in which agents pursue conflicting goals. In diverse MAS applications, both cooperative and competitive behaviors occur simultaneously, requiring hybrid interaction models.

Additionally, the Figure 1.3 illustrates the different interactions of an agent in its environment.

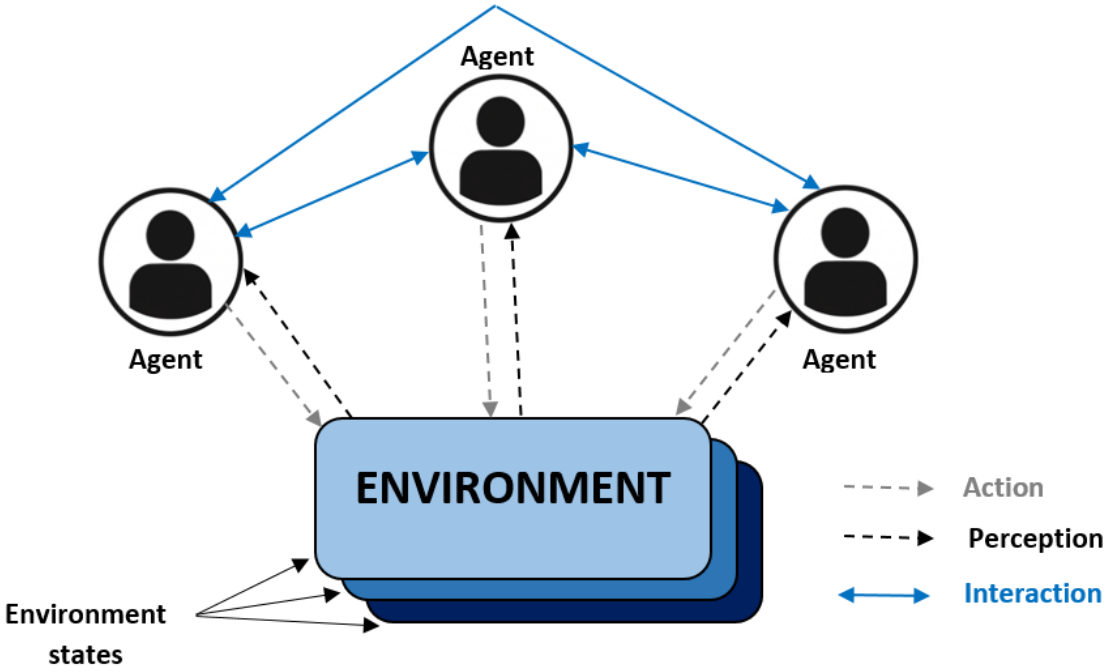

Figure 1.3: Interaction between an intelligent agent and its environment in an MAS

1.2.3.3 Taxonomy of multi-agent systems

To better understand the structure and behavior of MAS, it is useful to examine their categorizing taxonomy as outlined by Caridi and Cavalieri [11] (as shown in Figure 1.4). The model classifies MAS along five key dimensions: *Application Domain*, *Agent*, *Control*, *Organization*, and *Communication*. Each dimension represents a fundamental aspect of MAS design and implementation.

- *Application Domain* refers to the areas where agents operate, such as scheduling, monitoring, and distribution, with each agent has its own assigned task.
- *Agent dimension* highlights the personal properties of agents, emphasizing their roles, learning capabilities, autonomy, and adaptability.
- *Control* addresses the coordination mechanisms needed due to distributed decision-making, which can be either implicit (e.g. game theory [15]) or explicit (e.g., voting systems [16]).
- *Organization dimension* concerns the distribution of tasks and relationships between agents, taking various forms such as hierarchical structure.

- *Communication* underpins coordination and decision-making within the MAS, mainly through two paradigms: message passing (sharing information directly) and blackboard systems (sharing information indirectly via a standard data structure).

The structure of the figure emphasizes the multi-faceted nature of MAS, showing how agents interact, are organized, make decisions, and communicate within various industrial and operational contexts.

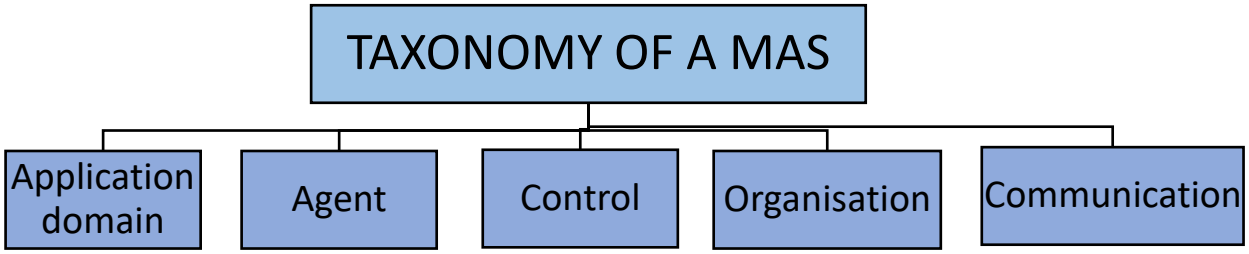

Figure 1.4: MAS Taxonomy

1.2.3.4 Applications of multi-agent systems

With a clear understanding of MAS architecture and agent types, building upon the foundational taxonomy of MASs, attention must shift toward how these systems are deployed in real-world scenarios. Accordingly different application examples of MAS are illustrated, according to the research [17] as follows:

- *Market Simulation*: MASs simulate energy market behavior and evaluate various market designs, with agents equipped with learning abilities like Q-learning to adapt and exploit market structures.
- *Monitoring and Security*: An illustrative example is provided using the Substation Physical Security Monitoring (SPSM) framework utilizes specialized techniques for real-time, remote monitoring of power substations, enhancing cybersecurity measures and managing trust.

- *System Diagnosis*: The application of logic-based or belief-desire-intention (BDI) agents in fault diagnosis systems, such as online fault detection in the Italian power system, is a notable example.
- *Distributed Energy Resource Management*: The integration and control of DERs in microgrids are facilitated by MAS platforms like VOLTTRON™ for decentralized control and automation of energy systems.
- *Smart Building Energy Management*: MASs VOLTTRON-based systems like BEMOSS, are used to control energy usage and enhance building efficiency.
- *Remedial Actions*: MASs is utilized in under-frequency load shedding (UFLS) to prevent blackouts by coordinating load shedding decisions across multiple agents in real-time.

1.3 Coordination in MAS

1.3.1 Basic principles of coordination in MAS

1.3.1.1 Definition of coordination in MAS

The characteristics of an effective MAS address the motivating questions of how can autonomous agents, each operating with limited information, partial knowledge, limited control over the global environment and local objectives, effectively collaborate to achieve coherent and globally optimal outcomes in a distributed environment. The answer lies in the coordination [14] in an MAS.

To begin with, according to [18] , coordination, in its simple definition, is the process that manages the dependencies between activities performed by agents, ensuring coherent interaction among distributed and autonomous entities. It is essential for effective project management. It synchronizes actions and ensures coherent system behavior. Another definition in [14] illustrates that coordination is a decentralized process in which autonomous agents align their actions to manage dependencies, prevent conflicts, and achieve coherent system behavior, particularly suitable for dynamic and distributed industrial environments. Numerous studies provide various definitions, but they all converge on a few essential characteristics that define the core of the concept. Therefore, effective coordination mechanisms are crucial for preventing conflicts and facilitating collective problem-solving in dynamic and distributed environments. Figure 1.5 reflects the coordination in MAS.

Furthermore, the coordination of MAS presents a significant challenge, as it requires the development of mechanisms that facilitate effective collaboration among agents while maintaining their autonomy

and managing uncertainty, incomplete information. This remains an active area of research with wide-ranging applications, including robotics [19].

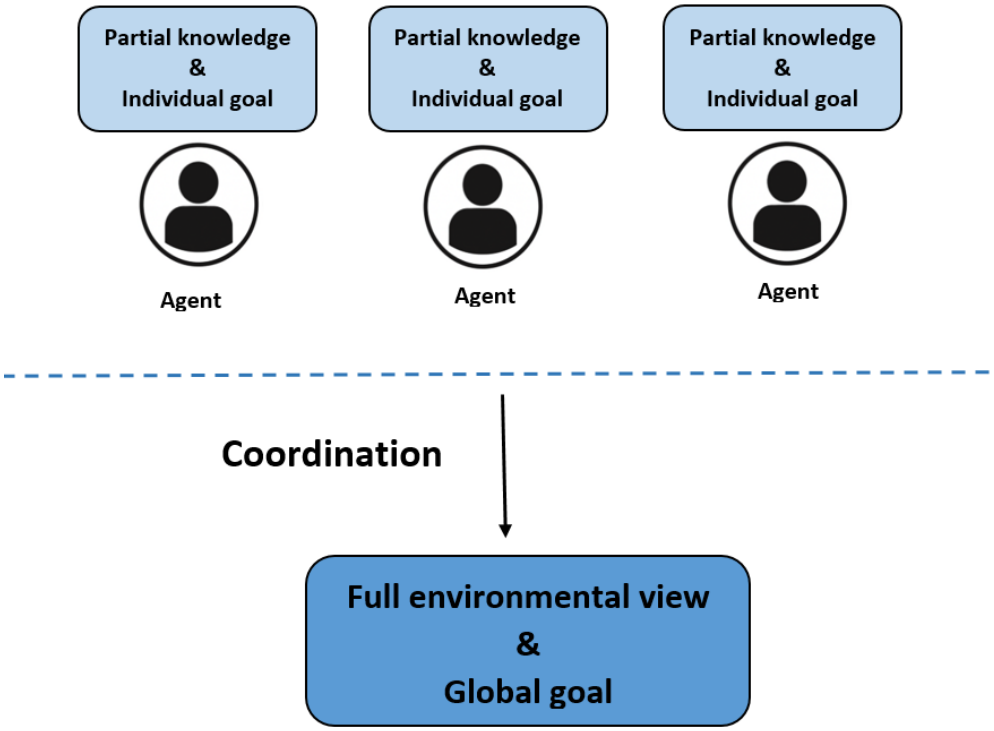

Figure 1.5: MAS coordination

1.3.1.2 Topologies of coordination in MAS

The MASs coordination topology is important for the efficiency of the system scalability, and resilience. It defines how agents are connected for decision-making, task allocation, and information sharing. A well-structured topology enhances fault tolerance, reduces redundancy, and improves communication. The challenge lies in designing mechanisms that preserve autonomy while dealing with uncertainty and potential conflicts. While the research does not explicitly list a formal typology, the paper [20] presents two key strategic dimensions for coordination called *information-processing structure* and *locus of control* [20]. In the first one, the process involves the generation and collection of information (locally or globally), its sharing methods (directly…), and the processing of this information (by a central planner or distributed). Moreover, the second aspect relates to the control structure of the system. whether centralized, distributed, or market-based, which determines where decisions are made and enacted, encompassing whether each agent acts independently or with minimal supervision.

Based upon them, coordination mechanisms can be categorized as follows:

- *Centralized Coordination*: a central agent manages global decisions, like optimal power dispatch, by collecting information from local agents, while coordination comes from a single agent, posing a single point of failure.
- *Distributed topology*: in contrast to centralized coordination, in this topology, agents interact directly or in hierarchical structures without a central controller. They can coordinate locally, enhancing robustness and scalability, but often require complex consensus protocols.
- *Hierarchical coordination topology:* coordination is organized in layers, with local agents making initial decisions and higher-level agents refining them, allowing for abstraction and modular control.
- *Negotiation-Based Coordination topology*: it is an additional topology discussed in [21], where agents negotiate resource allocation and conflict resolution based on utility or availability.

1.3.2 Multi-agent systems path planning

1.3.2.1 Basic principles of path planning MAS

In MAS, coordination and path planning [22] address the intelligent organization of agent movement. At the same time, each agent must establish its own path through space and time accounting, for the presence and actions of others. Instead, agents dynamically adjust their paths, factoring in the planned routes of others and communicating key information. Accordingly, they facilitate the achievement of common objectives and ensuring conflict avoidance. This coordination may be handled in various ways: through fully shared planning, where all agents are considered together or through more flexible methods where each agent plans independently, and adjustments are made to resolve conflicts. Generally, teams of agents are considered cohesive units, these units optimizes time and tasks through internal and external coordination. Ultimately, coordination is not only about preventing conflicts, it is about working together smartly, adapting in real-time, and making sure that all agents contribute to the mission in harmony. Another related research to autonomous navigation for intelligent vehicles, and robot navigation [23] involves four components: *perception*, *localization*, *cognition*, *path planning*, and *motion control*. The agent extracts meaningful information, determines its location, decides how to steer, and regulates its motion to achieve its desired trajectory. Figure 1.6 shows the relation between the four components.

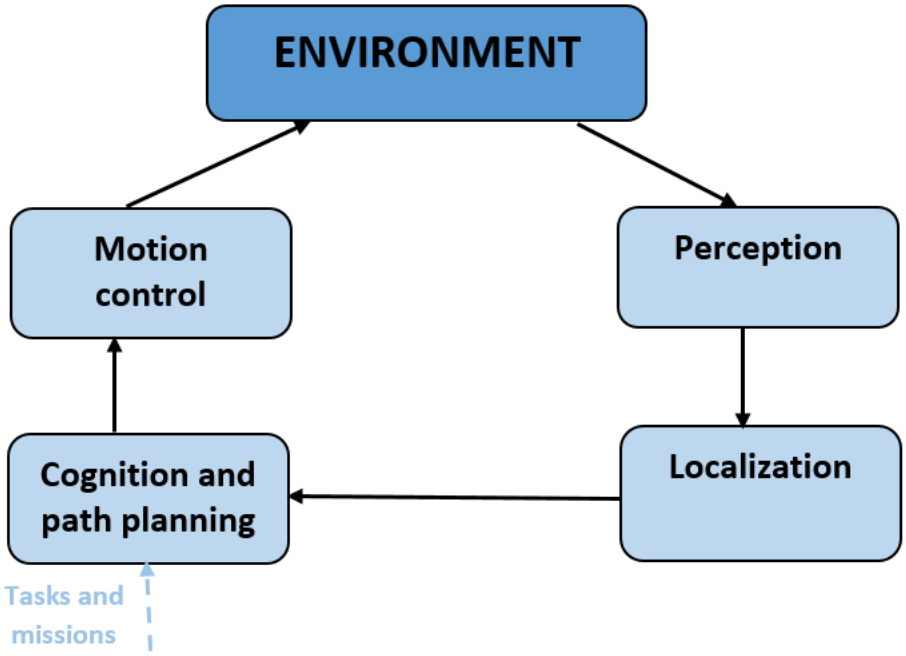

Figure 1.6: Agent navigation in an MAS

Building upon the foundational knowledge established from the literature on MAS path planning in various domains, the present investigation [24] discusses coordination and path planning in MASs in the context of multi-UAV (Unmanned Aerial Vehicles) missions, particularly for aerial search and rescue applications. Furthermore, another research [25] involves that MASs path planning and coordination involve finding free trajectories for multiple robots. These trajectories must satisfy mission goals while adhering to spatial and temporal constraints. Coordination mechanisms resolve conflicts between independently planned paths, ensuring safe and synchronized motion among robot teams. In addition, a recent study [26] offers a detailed and structured survey of the fundamental technologies enabling autonomous vehicle systems. It describes the foundation for how autonomous vehicles process and react to their surroundings, including path planning.

Consequently, although numerous applications of MAS path planning have been explored the cited examples represent some of the most practically applicable and widely studied cases in current literature.

Having established the relevance of MAS path planning in various domains, it is equally important to examine the criteria by which these systems are evaluated. Ultimately the paper [22] emphasizes the growing significance of path planning algorithms in enabling navigation in complex real-world environments efficiently and safely. It underscores that ideal path-planning algorithms should aim for these criteria:

- *Shorter path distance.*
- *Fast response times.*
- *Efficiency and optimality.*
- *Constraint satisfaction.*
- *Low computational time.*
- *Completeness (robustness across various scenarios).*

1.3.2.2 Path planning methods

Knowing that the objective of the MAS path planning coordination necessitates the development of algorithms capable of handling distributed decision-making, dynamic environments, and inter-agent communication. It is entirely appropriate to address the most important path planning algorithms in the context of discussing path planning itself. Accordingly, the same previous paper [22] has classified path planning algorithms into two main categories: *Classical approaches* and *Heuristic approaches***.** They both can be discussed as follows and can be summarized in the Figure 1.7:

- *Classical approaches*: in their definitions, they are deterministic, mathematically based, and accurate in static environments but computationally expensive for dynamic systems and limited in handling uncertainty.

  Over the decades, Lots of studies present various classical techniques that offer different trade-offs in terms of optimality, computational complexity, and environmental adaptability. Among this rich corpus of research, a set of well-established classical methods has consistently attracted scholarly attention due to their effectiveness and algorithmic simplicity.

  Starting with the study [27], it examines classical motion planning algorithms like *Potential Field (PF), Dijkstra's Algorithm (DA), Rapidly-explore Random Tree (RRT),* and *Probabilistic Road Map (PRM),* chosen for their widespread application in path planning problems. In addition to these algorithms, another work [22] introduced additional classical algorithms, among which are *cell decomposition* and *mathematical programming*. In addition, the *subgoal*

method (SG) and *sampling-based methods* are presented in [23]. In the same context, the paper [28] discusses different methods types, emphasizing classical ones such as *random walk algorithms* (levy flight, Brownian methods).

- *Heuristic approaches*: they prioritize computational efficiency and scalability, utilizing intelligent guessing, learning, or biological inspiration, suitable for dynamic, complex environments and faster execution but lacking guaranteed optimality.

  In the aim of discussing the primary heuristic methods, a previous work [28] classifies *Dijkstra's Algorithm* together with *A*, depth-first,* and others as greedy and graph search algorithms in the subclass of heuristic approaches. It also presented other subclass named bio-inspired heuristics clusters other subclasses, which concentrate on reinforcement learning (*Markov decision process framework*), deep reinforcement learning methods (*Deep Deterministic Policy Gradient*), and neural networks. The common point between them is that all of them are human-based algorithms. Besides them, there exist evolutionary algorithms; this subclass contains *metaheuristic-based methods*. The latter will be examined in detail later, as it forms one of the core elements of this work.

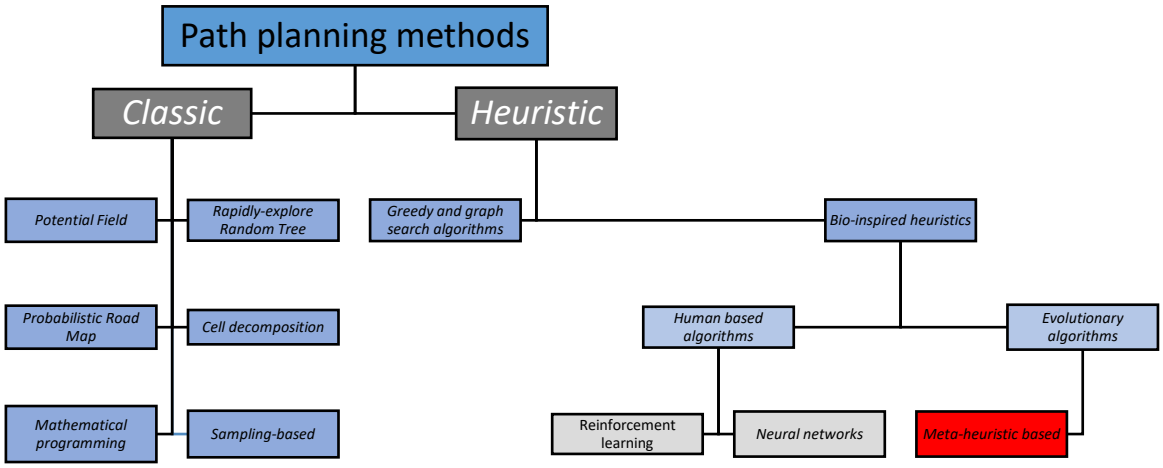

Figure 1.7: Path planning methods

1.3.2.3 Collision avoidance

Within the framework of MAS, path planning and collision avoidance [29] are fundamentally intertwined; the collision avoidance is the critical mechanism that ensures agents can navigate toward their goals without physically colliding with obstacles or with each other. It transforms them into a synchronized team, while maintaining progress toward individual goals. It overrides standard navigation when immediate threats are detected. When time is too short for a detour, the agent triggers a reflexive action, like braking or swerving, prioritizing immediate safety over reaching its destination to avoid a collision with dynamic, unpredictable obstacles. Consequently, in MAS, path planning is often global, whereas collision avoidance is local. In MAS, instead of one agent doing all the work, both agents move a little bit to stay out of each other's way. This team effort optimizes efficiency and conserves energy for the entire group. According to the article [30], collision avoidance is the indispensable element that allows an agent to reach its destination safely while minimizing risks in its environment. It reviews several methods for more complex or high-dimensional environments (where classic methods might fail or get stuck in local minima). Consequently, it categorizes collision and obstacle avoidance into two main schools of thought: Classic and Heuristic/Modern approaches. Additionally, the core contribution of the thesis [31] is the move toward Mathematical Optimization (Operations Research), where the constraints (maintaining distance, speed limits, and altitude bounds) are more important than the goal. Moreover, in relation to obstacle avoidance concept, the collision avoidance provides the real-time social awareness needed to safely navigate around other moving agents, whereas obstacle avoidance ensures that an agent prevents impact with immobile structural elements.

In the real world, this mechanism is the difference between an efficient, scalable operation and a costly disaster; it protects expensive physical assets, ensures operational continuity in busy environments like warehouses, and builds the public trust necessary for humans and machines to work safely side-by-side. In relation to MAS, the agent might be drones, robots, or self-driving cars

1.3.3 Multi-agent systems task planning

In distributed and complex environments, a good coordination of autonomous agents is crucial for achieving collective goals. Task Coordination (TC) in MAS tackles the challenge of allocating tasks among agents and ensures that assigned responsibilities are achieved in a clear and responsible way. It can be described as the structured assignment and execution of tasks among multiple autonomous agents to achieve a common objective. It includes both *task allocation* and *task responsibility*. In addition, task

allocation refers to the process of assigning specific tasks to individual agents or groups of agents. Additionally, task responsibility makes agents accountable for their assigned tasks based on capability, history, and actions. Assignments are considered justifiable when they are consistent with the previous task allocations and the results that have been observed.

According to [32], task coordination in MAS involves two main tasks: task allocation, which assigns tasks to agents to achieve a goal, and task responsibility, which monitors completion and accountability. Challenges include agents' autonomy and imperfect information, and traditional approaches assume perfect information and assign tasks to individual agents. The cited work categorizes the principles of task coordination into:

- *Suitability of the State of Affairs*: only states that are achievable by the collective actions of the entire agent group (the *grand coalition*) should be considered.
- *Validity of Task Allocation*: a task allocation is valid if it assigns exactly what is necessary; assuming all agents achieve their assigned tasks, the desired state of affairs is guaranteed.
- *Justifiability of Task Responsibility*: responsibility must be aligned with prior task allocation. It depends not only on the capabilities of an agent, but also on the tasks assigned and the history of actions taken.

This process becomes challenging when agents operate within perfect information or in a dynamic environment that requires strategic reasoning and coordination mechanisms. Effective MAS task coordination is fundamental in domains such as distributed robotics, sensor networks, and collaborative decision-making systems. In this case, developing formal frameworks for task coordination plays a central role in enabling reliable and efficient MAS performance across domains such as robotics, disaster response, and distributed sensing. The work [32] confirmed that. It emphasizes the necessity of advanced TC frameworks in MAS that extend beyond traditional assumptions. It suggests a conceptual and formal foundation for solving TC under strategic and informational constraints, aiming to bridge theoretical gaps in MAS coordination.

The study [33] provides TasCore. It offers MASs a framework for dynamic task coordination that takes into account both task responsibility and task allocation. In contrast to prior methods, it considers agents' strategic abilities and epistemic limitations, enabling coordination under imperfect information and supporting group based task assignments. TasCore integrates these aspects into a unified model and

formally verifies key properties such as validity, stability, fairness, and non-monotonicity of task responsibilities. This is in one hand.

On the other hand, the work [34] presents a decentralized framework for task coordination in multi-agent systems, where each agent is assigned a local task defined by scLTL formulas. Coordination relies on real-time message exchange and supports collaborative actions and task swapping. The approach is fully distributed, scalable, and resilient; it avoids centralized control by dynamically forming dependencies based on agent capabilities and task progress. If all of this is combined, the closed loop of the MAS will be as follows:

- The MAS initiates coordination.
- Task allocation is guided by strategic ability.
- A refined allocation accounts for epistemic limitations.
- Task responsibility ensures outcomes are monitored and traced.
- Feedback is sent to the MAS, closing the loop and preparing for future cycles.

Consequently, MAS task coordination requires the distribution of local tasks, and attributes responsibility, taking into account strategic capabilities and the limits of their information of agents. Recent frameworks incorporate decentralized, message-driven approaches that support collaboration, task swapping, and resilience without relying on centralized control.

Lastly, MAS path planning ensures agents move without conflict through space, and optimizing motion. In contrast, task coordination distributes interdependent tasks efficiently across agents. It requires synchronization and cooperation based on task complexity and agent capabilities.

In MAS, the effectiveness of task coordination and path planning is important for achieving collective goals in dynamic and often constrained environments. While numerous approaches exist, increasing attention has been directed toward metaheuristic-based methods due to their flexibility and scalability. Metaheuristic approaches offer innovative alternatives to traditional strategies. They enable decentralized decision-making and global optimization. The following section will explore these metaheuristic techniques in detail, highlighting their principles, advantages, and applications in MAS coordination and path planning.

1.4 Metaheuristics

1.4.1 Metaheuristics and optimization problems

With advances in information technology, several optimization problems are encountered in fields like engineering, operation research, bioinformatics, and geophysics. Most optimization problems are NP-hard, non-linear, and multi-dimensional search spaces, complicating their resolution. Research has developed alternative approaches, categorized as heuristics and metaheuristics. Heuristics are more problem-dependent and can be applied to specific problems while becoming insufficient for other ones. From the chapter [35], metaheuristics are generic algorithms that can be used for almost all optimization problems. These methods are adaptive strategies that navigate uncertainty, diversity, and dynamic constraints in complex systems. Drawing inspiration from evolution, swarms, and collective intelligence, they aim for good enough, fast enough, and often better than expected solutions. However, they cannot solve all optimization problems and have the same average performance. Additionally, the paper [36] illustrates that metaheuristics occupy a central role within the broader category of approximate optimization methods. In contrast to exact methods, which ensure optimality (e.g., branch-and-bound, dynamic programming), metaheuristics are intended to facilitate efficient searching in large, complex, and high-dimensional search spaces to find good-quality solutions within practical time limits. They serve as general-purpose guiding strategies that enhance and control the behavior of problem-specific heuristics. Another definition in [37] demonstrates that a metaheuristic is a high-level, problem-independent algorithmic structure providing general strategies for designing heuristic optimization algorithms. It aims to find "good enough" solutions in a reasonable computational time, making it suitable for large, complex, or NP-hard problems where exact methods are impractical. Stating the origin of the word and introducing it through its linguistic components, combining ‘meta’ (beyond) and ‘heuristic’ (to find/discover).

To facilitate comprehension, Figure 1.8 shows a metaheuristic algorithm processing candidate solutions from the problem space and identifying the best solution.

It is noteworthy that across a wide range of literature and definitions of the metaheuristic word, the most recurrent term is “*optimization problem*”. Consequently, and based upon [38], optimization problems are mathematical or computational challenges aimed at finding the best solution from feasible options, often involving maximizing or minimizing objective functions like profit, cost, efficiency, or distance while satisfying constraints. They are common in fields like engineering, economics, logistics, and machine learning. Specifically, distributed optimization problems involve multiple agents or nodes working together to minimize a shared global objective, with the global cost function typically expressing the sum of local objective functions. It can be presented as follows in equation 1.4 [38]:

$$\min_{x \in R^n} \sum_{i=1}^{N} f_i(x) \qquad (1.4)$$

Where: N is the number of iterations; each agent i has access only to its own local function $f_i(x)$.

Briefly, the main common ideas arise from the fact that heuristic are tailored to individual problem domains, while metaheuristics are domain-independent and act as high-level frameworks capable of adapting to a wide range of optimization problems. Their flexibility and scalability make them especially suited for solving complex problems making, them more adaptable for solving real-world optimization problems, including MAS coordination and path planning.

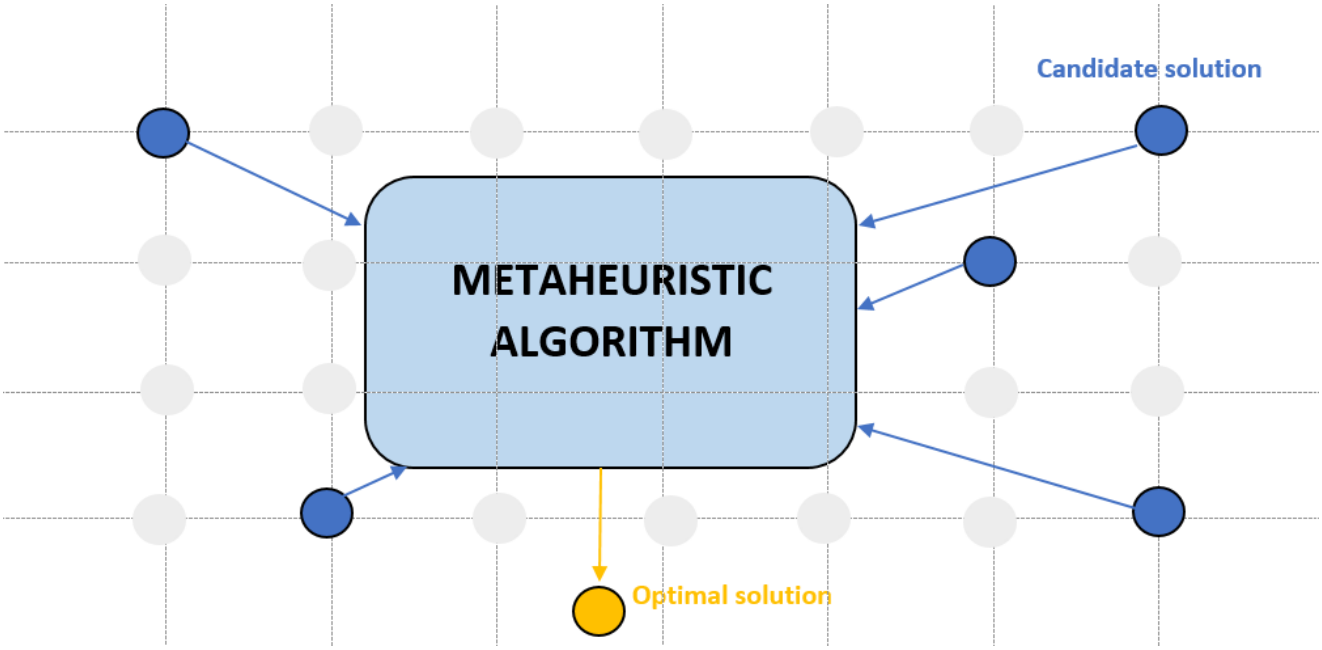

Figure 1.8: Metaheuristic algorithm for optimization problem solving

1.4.2 Exploration and exploitation

In essence, an effective metaheuristic balances exploration and exploitation well to optimize the exploration of the solution space and find effective solutions. Metaheuristics are robust searching mechanisms that harmonize exploration and exploitation search schemas. They leverage exploration to avoid local optima and exploitation to converge efficiently towards optimal solutions. They are inspired by natural processes and incorporate mechanisms to balance exploration and exploitation. Moreover, the study [39] examined these two concepts as central to search behaviors across many domains, from animal foraging to human decision-making and AI. It involves balancing using known opportunities (exploitation) and seeking new ones (exploration). Numerous solutions are available, including mixing

exploitation with exploration phases in ecology and reinforcement learning in computer science. It is key to understanding cognitive behavior and its wide-ranging effects.

To clarify further, the following Figure (Figure 1.9) illustrates how the process occurs. Starting by searching for information, optimize promising areas, and then return to exploration to avoid local optima and find new regions if the solutions found are not suitable.

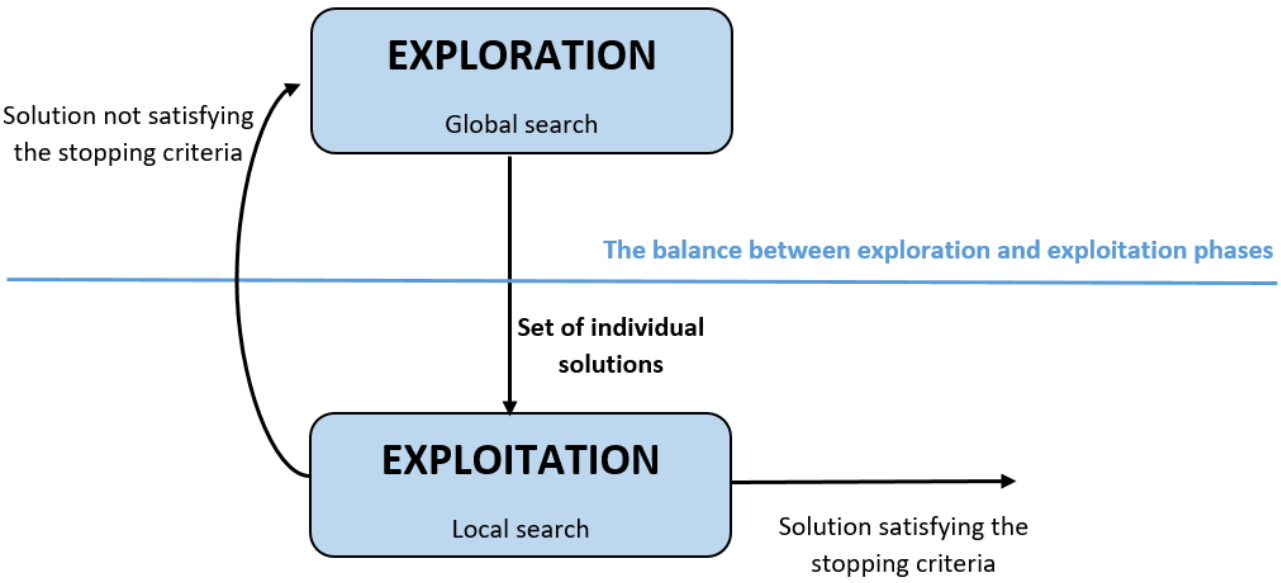

Figure 1.9: Metaheuristic optimization process

The following examples highlight how different algorithms balance between exploration and exploitation; although there are other examples, these are the most recent and relevant.

- *Genetic Algorithms:* use crossover and mutation to explore new solutions (exploration) while selecting the best for the next generations (exploitation).
- *Simulated Annealing:* it starts by accepting less optimal solutions, and then progressively focuses on refining the best options found.
- *Particle Swarm Optimization:* particles exchange information to diversify exploration, while simultaneously converging on optimal solutions.
- *The Crow Search Algorithm:* balances exploration and exploitation by allowing crows to explore new areas randomly while simultaneously refining the best solutions discovered. This adaptive behavior provides search for optimal solutions while avoiding local optima.

- *Ant Colony Optimization***:** Exploration involves ants moving randomly, while exploitation occurs when pheromone updates highly successful routes, guiding future searches toward promising solutions.

Based on the research [35], metaheuristics, which involve exploration and exploitation, are crucial for efficient search processes. Different classifications exist based on the method of employing these strategies and the metaphor of search procedures.

1.4.3 Metaheuristic classifications

Metaheuristics are classified in many ways in the literature; each one is distinguished by its own set of defining dimensions. Several factors account for this variety of classifications. The first factor refers to the diversity of optimization problems across different domains, which necessitate tailored approaches, leading to the development of various metaheuristic techniques. The second factor is related to the flexibility and diverse applications of metaheuristics. These approaches can be categorized based on multiple criteria, like their strategies, mechanisms, and sources of inspiration. Thirdly, the regular appearance of new algorithms and hybrid methods conduct to the development of numerous classification schemes. Lastly, different researchers may emphasize distinct aspects, such as exploration versus exploitation, population-based versus single-solution methods, or nature-inspired versus problem-specific heuristics, resulting in varied frameworks for categorization.

We try to explore and analyze these classifications and their respective works in a comprehensive manner. Starting with the work [35], it summarizes the most common classifications as follows:

- *Trajectory-Based vs. Population-Based Algorithms:*
    - *Trajectory-Based Algorithms:* they utilize a single solution that is improved iteratively. They include Simulated Annealing (SA) and Tabu Search metaheuristics.
    - *Population-Based Algorithms:* these algorithms maintain a population of solutions, evolving them over time. They include Genetic Algorithms (GA) and Particle Swarm Optimization (PSO).
- *Nature-Inspired vs. Non-Nature-Inspired Algorithms:*
    - *Nature-Inspired Algorithms:* they are modeled after natural behaviors or phenomena. For instance, GA are based on the principles of natural evolution, while PSO is inspired by the social behavior of birds.
    - *Non-Nature-Inspired Algorithms:* these algorithms are not inspired from natural phenomena.

- *Metaphor-Based vs. Non-Metaphor-Based Algorithms:* (the proposed classification in [35]):
    - *Metaphor-Based Algorithms:* they are inspired by metaphors or real-world processes. For example, the Ant Colony Optimization (ACO) algorithm is based on the metaphor of ants searching for food. According to the same research [35], it contains subclasses as follows:
        - *Biology.*
        - *Chemistry.*
        - *Music.*
        - *Math.*
        - *Physics.*
    - *Non-Metaphor-Based Algorithms:* these algorithms are not relied to metaphors. They are based on mathematical or statistical principles.
- *Other Classification Criteria:*
    - *Use of Memory:* some algorithms use memory structures to avoid reviewing explored solutions like Tabu Search.
    - *Neighborhood Structures:* the algorithms may differ in the way of defining and exploring the neighborhood of a solution.
    - *Objective Function Usage:* the objective function can be employed in different ways, influencing the algorithm's performance and convergence behavior.

According to the research [36], the different ways to classify metaheuristic algorithms based on characteristics used to differentiate amongst them include:

- *Nature inspired vs non-nature inspired*:
    - *Nature-Inspired**:*** these algorithms are inspired by natural phenomena.
    - *Non-Nature-Inspired**:*** these algorithms are not based on natural systems and behaviors.
- *Population-based vs single-point search*:
    - *Single-Point Search**:*** these methods manipulate a single solution at a time and are typically intensification-oriented. Such as Tabu Search (TS).
    - *Population-Based Search*: these methods uses a population of solutions. It allows for broader exploration of the search space, including GA.

- *Iterative vs greedy*:
    - *Iterative Algorithms**:*** these algorithms begin with an initial solution (or set of solutions) and improve them as the search progresses, such as PSO.
    - *Greedy Algorithms**:*** these algorithms build the solution step by step, adding components to the solution at each stage based on local decisions. Such as ACO.
- *Dynamic vs static objective function*:
    - *Static Objective Function**:*** these algorithms keep the objective function unchanged throughout the search process, including PSO
    - *Dynamic Objective Function**:*** these algorithms are characterized by the modification of the objective function during the search, for more effectiveness.
- *Memory usage vs memory less*:
    - *Memory-less*: algorithms that only use the current state of the search and do not retain any past information.
    - *Memory-Based*: algorithms that use information acquired during the iterative search process.
- *Single vs multiple neighbourhood structures*:
    - *Single Neighbourhood Structure:* algorithms that operate using only one neighborhood structure, such as Iterated Local Search (ILS).
    - *Multiple Neighbourhood Structures:* algorithms that use several neighborhood structures in their search process, such as Variable Neighbourhood Search (VNS).

For clarification, a simplified Figure (Figure 1.10) illustrates the classification of metaheuristics based on a previous research [28].

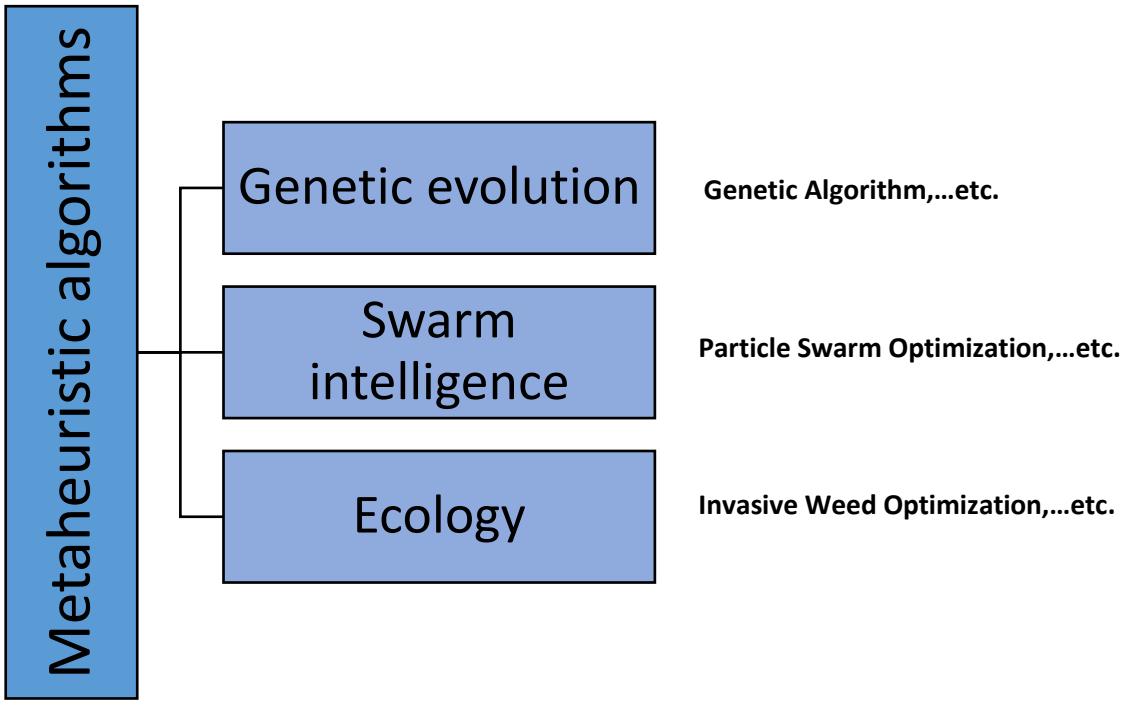

Figure 1.10: Metaheuristic classifications according to the research [28]

Additionally, a simplified Figure (Figure 1.11) provides another detailed illustration of the classification of metaheuristics, derived from the research [40]:

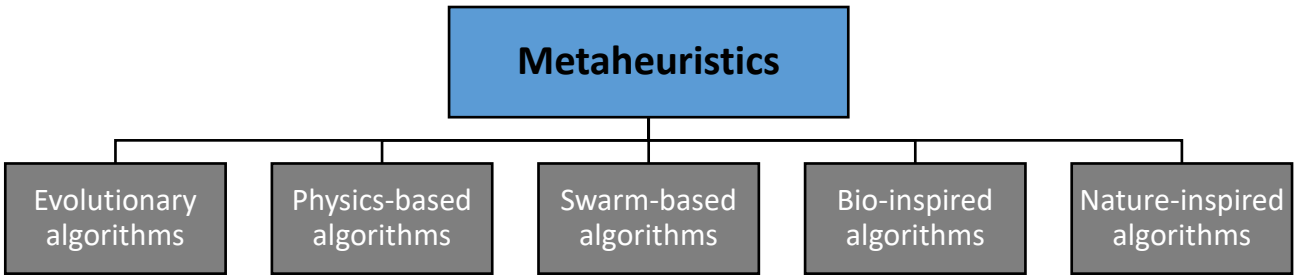

Figure 1.11: Metaheuristic classifications according to the research [40].

In summary, metaheuristic algorithms can be classified into various categories based on their underlying principles, such as evolutionary algorithms, swarm intelligence, physics-based approaches, and human-

inspired techniques. Each classification offers unique strengths tailored to specific optimization problems.

1.4.4 Metaheuristic applications

As mentioned previously, metaheuristics are versatile optimization methods that can be used in many areas, like bioinformatics, machine learning, transportation, and engineering design. They efficiently explore large search spaces, providing near-optimal solutions where traditional methods may struggle. The examination of their applications shows how useful they are and how they could be improved to solve real-world problems. Furthermore, research has examined the applications of metaheuristics across diverse fields. It has helped improve the theoretical understanding and practical use of metaheuristic techniques in real-world problems.

Taking an example, the book [41] provides an extensive overview of the various application domains of metaheuristics, including:

- *Engineering Design*: metaheuristic algorithms are necessary for solving challenging optimization problems in engineering design, which often include conflicting goals, design variables, and constraints. Telecommunication network design is one example, where they improve routing. Robotics is another, where they are used for planning paths, controlling motion, and placing sensors. They also include ways to make energy systems work better.
- *Machine Learning and data mining:* metaheuristics are used for tasks like feature selection, hyperparameter tuning, and model optimization, improving the performance and efficiency of machine learning models. Some specific applications include bioinformatics and finance.
- *System Modeling, Simulation, and Identification:* this field creates mathematical models of systems in chemistry, physics, and biology. These models can help make predictions and understand complicated systems. Some examples of applications are chemical reactions, physical systems, biological processes, control systems, signal processing, and image processing. Signal and image processing techniques involve changing and analyzing signals and images for different uses.
- *Planning in Routing, Scheduling, and Logistics:* this field focuses on solving optimization problems in routing, robot planning, scheduling, production, logistics, and transportation. The goal is to minimize travel time, optimize resource utilization, and reduce costs.

Passing to another research [40], it discusses metaheuristic algorithms and their applications in various domains. GA are used in optimization problems with combinatorial or discontinuous objectives and

constraints, while ACO is used in dynamic programming and multi-objective real-world problems. Simulated Annealing (SA) is a probabilistic method used to solve challenging optimization problems that does not take long computing time and can get around local optima. The PSO is ideal for real-time applications because it is fast. Differential Evolution (DE) works well with complicated cost functions, and Tabu Search (TS) works well for both global and local searches. Hybrid algorithms like HABCDE combine the strengths of different metaheuristic approaches, often outperforming individual algorithms in terms of solution quality and computational efficiency.

While metaheuristics have proven effective across various complex problems, combining them with other strategies can further improve solution quality and convergence speed. This combination is called hybridization of metaheuristics.

1.4.5 Hybrid metaheuristics

Even though each metaheuristic works well on its own, it can not find a good balance between exploration and exploitation. This leds to the creation of hybrid approaches that combine different algorithms. To comprehend the development and efficacy of hybrid metaheuristics, it is crucial to examine the principal research initiatives that have concentrated on the integration of diverse algorithms. Firstly, in the chapter [42], hybrid metaheuristics are optimization algorithms that combine various techniques, including metaheuristics and other methods like exact algorithms, to exploit strengths and compensate for weaknesses. It illustrates that the core motivation is to achieve synergy by combining strategies with complementary strengths (e.g., exploration vs. exploitation). In addition, it is useful for complex combinatorial optimization problems, where pure methods may fail to deliver optimal or near-optimal performance. Another study in [43] provides that hybrid algorithms combine multiple algorithms to enhance search performance, focusing on convergence speed, solution accuracy, and diversity maintenance during optimization.

The study [42] classifies hybrid metaheuristics based on four dimensions as follows:

- *Hybridized Algorithms*:
    - Between different metaheuristics
    - Between metaheuristics and exact methods (e.g., branch-and-bound, LP, CP)
    - Between metaheuristics and heuristics/soft computing (e.g., neural networks, fuzzy logic)
    - Human-guided hybrid search involving user input and visualization.
- *Level of Hybridization*:

  - High-level hybrids: loosely coupled, each algorithm retains identity.
  - Low-level hybrids: they are tightly coupled and depend on each other.
- *Order of Execution*:
  - Sequential: one method follows another.
  - Parallel execution with data exchange.
- *Control Strategy*:
  - Integrative (coercive): one method embedded into another.
  - Collaborative (cooperative): separate methods exchange information.

The study [43] illustrates that the past of hybrid metaheuristics is characterized by the transition from standalone evolutionary methods to structured hybridizations aimed to reducing their weaknesses. These initial experiments set the stage for more advanced hybrid frameworks seen in subsequent decades. Additionally, the current state of hybrid metaheuristics is characterized by the structured integration of established algorithms, seeking for stricking a balance between exploration and exploitation. The future promises new biologically inspired methods and more principled, adaptive hybrid architectures—but warns against random and unjustified algorithmic mashups. Only through theoretical grounding and simplicity can future hybrids achieve real-world impact and scientific relevance.

1.5 Summary

In summary, this first chapter has established the theoretical and methodological foundations necessary for addressing the coordination challenges inherent in MASs. It has examined three fundamental and interrelated domains: the principles of MASs, the coordination of path planning and task planning, and the use of metaheuristic algorithms as powerful tools for solving complex, nonlinear, and combinatorial optimization problems. By consolidating these core concepts, the chapter has laid the groundwork for the design and implementation of the proposed approach to intelligent coordination in distributed environments. The emphasis was placed not only on defining key terms and models but also on highlighting the motivations behind selecting these frameworks and techniques for the development of a scalable and adaptable MAS coordination. The next chapters will expand on this basis.

CHAPTER 2:

METAHEURISTICS FOR MULTI-AGENT CONTROL: AN OVERVIEW

2.1 Introduction

Solving critical problems like obtaining the optimal path planning or distributed task coordination in MASs involves navigating high-dimensional search spaces, where traditional algorithms often lack flexibility and scalability. Luckily, metaheuristic algorithms offer a strong solution to these limitations. They provide approximate solutions to complex optimization problems. Consequently, the main goal of this chapter is exploring the use of different metaheuristic algorithms in MASs and addressing optimal path planning and efficient task coordination. It highlights their strengths, limitations, and their future directions. To address this, this chapter is structured into:

Section 1 offers the introduction. Section 2 introduces detailed descriptions of many different metaheuristic algorithms and describes the relation of metaheuristic to MAS. Section 3 concentrates on the use of metaheuristics in the MAS path planning problem. It reviews recent studies and details the performance of different algorithms, which are employed for the efficient navigation of agents in constrained environments. Section 4 examines task coordination in MAS. It discusses how metaheuristics contribute to the dynamic allocation and scheduling of tasks among agents under complex operational conditions. A concise conclusion is presented in Section 5.

2.2 Concepts description

This section provides detailed descriptions of various metaheuristic algorithms and elucidates their relationship (in general) with MAS.

2.2.1 Metaheuristics methods

The theoretical background of metaheuristic algorithms are elaborated upon in Chapter 1, Section 4. A discussion of metaheuristics typically includes the most popular and commonly applied methods. In this context, specific mention would be:

2.2.2.1 Particle Swarm Optimization

The PSO algorithm for Particle Swarm Optimization. It was introduced by Russell Eberhart and James Kennedy in 1995 [44]. It is inspired by bird flocking behavior. The method is a population-based metaheuristic using particles as agents in MAS that collaborate and interact with each other and with their environment, to converge on optimal solutions. Aiming to balance between exploration and exploitation in the search space. In PSO, each particle indicates a potential solution. Moreover, it has a position. This position is evaluated based on an objective function in relation to its personal best solution and the global best solution found. Additionally, the particle has a velocity. This velocity is a physical

quantity representing the change in position over time. It is a vector quantity with both magnitude and direction.

The particle's position is updated during each iteration using equation 2.1 as follows [45]:

$$x\,(t + 1) = x(t) + v(t + 1) \qquad (2.1)$$

Where: x – position, v - velocity of the particle at time t.

The velocity is calculated using equation 2.2 as follows [45]:

$$v(t + 1) = wv(t) + c1r1(pBest(t) — x(t)) + c2r2\,(gBest(t) - x(t)) \qquad (2.2)$$

Where: *pBest* – best solution found by the particle, *gBest* – best solution found by the group of particles, *c1*, *c2* – acceleration coefficients, *r1*,*r2* – stochastic random constants between 0 and 1, *w* – inertia weight.

pBest is updated through a comparison with the new particle's position at each iteration by taking into account the fitness values. The *gBest* position is used to describe the position that had the highest fitness value until the current iteration. *w* controls the PSO algorithm's exploration and exploitation.

Figure 2.1 visually illustrates how PSO orchestrates particle movements to converge toward optimal solutions.

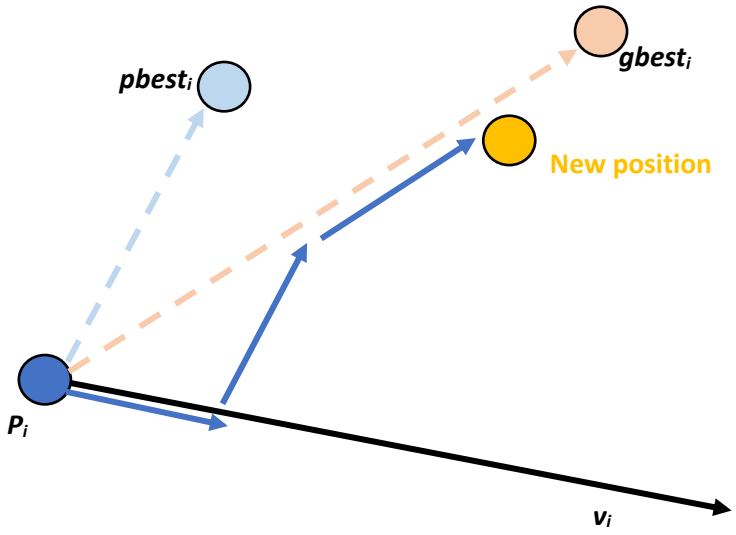

Figure 2.1: Visual representation of the PSO algorithm

2.2.2.2 Ant Colony Optimization

The ACO algorithm for Ant Colony Optimization was first introduced in Marco Dorigo's doctoral thesis in 1992 [46]. The ACO metaheuristic was inspired by the behavior of real ants searching for food. It works effectively for addressing various optimization problems, such as a dynamic TSP and dynamic QAP problem [47]. The algorithm uses artificial ants developing solutions iteratively through pheromone-based communication, allowing the colony to explore the solution space. The concentration of pheromones depends on how much the path is used. The more a path is traversed, the more pheromone levels increase. These lasts guide other ants toward optimal solutions. They are updated based on solution quality. Furthermore, the pheromones evaporate over time, escaping local minima. Ultimately, and in relation to MAS, the ants in ACO communicate through pheromones, sharing information about solution quality, mirroring the collaborative strategies observed in MASs.

ACO builds solutions iteratively, choosing components from a predefined set at each step. The equation 2.3 shows how to calculate the probability of selecting component $p(ci \mid s)$ [48]:

$$p(c_i \mid s) = \frac{[\tau_i]^{\alpha}.[\eta(c_i)]^{\beta}}{\sum_{c_j \in N(s)} [\tau_i]^{\alpha}.[\eta(c_i)]^{\beta}} \qquad (2.3)$$

Where: s – the current, partially constructed solution, c_i –solution component, τ_i – pheromone value associated with the solution component c_i, $\eta(c_j)$ – heuristic information for a component that is assigned to each feasible solution component $c_j \in N(s)$, α,β – positive parameters that control the relative influence of pheromone and heuristic information.

The pheromone update prevents premature convergence and encourages exploration of better solutions. It is updated using equation 2.4 as follows [48]:

$$\tau_i \leftarrow (1 - \rho).\ \tau_i + \rho.\ \sum_{\{s \in S_{upd} \mid c_i \in s\}} w_s F(s) \qquad (2.4)$$

Where: S_{upd} – set of solutions selected for the update. $\rho \in [0, 1]$ – evaporation rate, τi – remaining pheromone after evaporation, $F : S \rightarrow R+$ – the quality function such that $f(s) < f(s') \rightarrow F(s) \geq F(s')$, $\forall s \neq s' \in S$, w – The weight associated with a solution s can be used to influence the quality function, while $f(s)$ is the objective.

The evaporation rate controls how quickly pheromone levels decay. High ρ encourages exploration by decaying pheromone trails, while low ρ encourages exploitation by persisting strong trails, guiding ants towards known solutions. Consequently, $(1 - \rho)$.

Figure 2.2 illustrates a visual representation of the ACO algorithm, which is inspired by the natural foraging behavior of ants. It shows how ants find the shortest path from their nest to a food source. They use pheromone trail reinforcement. Moreover, a high pheromone level characterizes the shortest path.

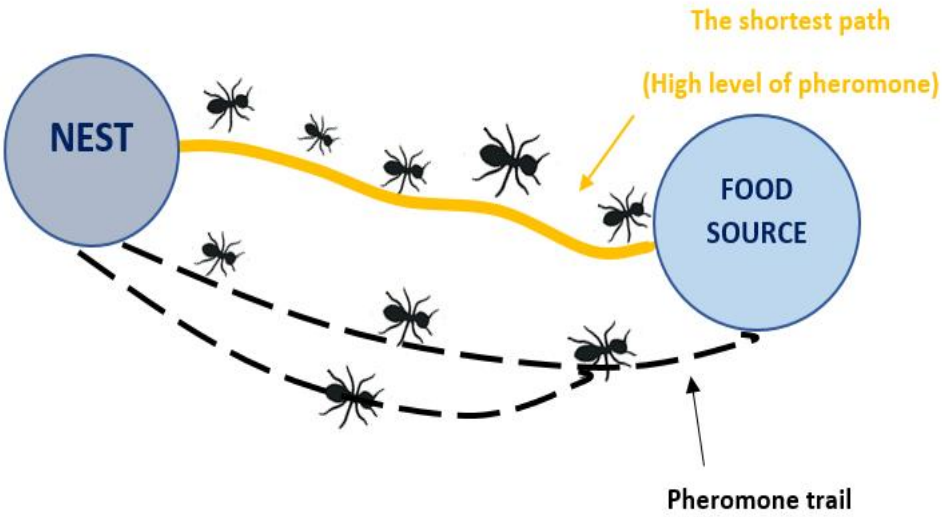

Figure 2.2: Visual representation of the ACO algorithm

2.2.2.3 Artificial Bee Colony

The ABC algorithm for Artificial Bee Colony was initially proposed by Dervis Karaboga in 2005 in his publication [49]. Consistent with its appellation, this algorithm is derived from the behavior of honeybees in searching for food and sharing it with other bees. It is considered a swarm algorithm, which divides the population's roles into three crucial roles. Starting with employed bees, each one represents a potential solution; it has to search for better food sources while exploring its neighborhood. Passing to onlooker bees, this kind of bee is occupied with selecting food sources based on employed bees' food quality (fitness) and proximity. In addition, scout bees play a crucial role. These bees search for new food sources in a random way, exploring new areas if the existing sources fail to produce better results. This ensures the balance between exploration and exploitation. In relation to MASs, in ABC, the bees are considered agents, which collaborate to acquire knowledge of food sources in search space.

An onlooker bee chooses a food source depending on the food source's probability value *pi* calculated using equation 2.5 as follows [50]:

$$p_i = \frac{f\,it_i}{\sum_{j=1}^{SN} f\,it_j} \qquad (2.5)$$

Where: $i = 1, 2,...,SN$, and SN is the size of employed bees or onlooker bees (solutions or food sources), $j = 1, 2,...,D$, and D is the number of optimization parameters; fit_i is the fitness value of solution i. Knowing that each solution is initialized using equation 2.6 [50].

$$x_{i,j} = x_{min,j} + rand(0, 1)(x_{max,j} - x_{min,j}) \qquad (2.6)$$

Where: $x_{min,j}$,$x_{max,j}$ – the lower and upper bounds for the dimension j, respectively.

To generate a new solution V_i, the algorithm takes an existing solution X_i, selects another different random solution X_k, and adds a random value φij in the range [−1, 1] times the difference between X_i and X_k to X_i. Where $k \in \{1, 2,...,SN\}$ and $j \in \{1, 2,...,D\}$, the equation 2.7 shows that. If a solution does not improve after a certain number of cycles, it is replaced with a random one using equation 2.6.

$$v_{i,j} = x_{i,j} + \varphi_{i,j}(x_{i,j} - x_{k,j}) \qquad (2.7)$$

The following Figure (Figure 2.3) illustrates a visual representation of ABC Algorithm**.** It represents how the algorithm explores and exploits solutions in an optimization problem using the honey bee behavior in search of food specifically focusing on how different roles are distributed across the process phases.

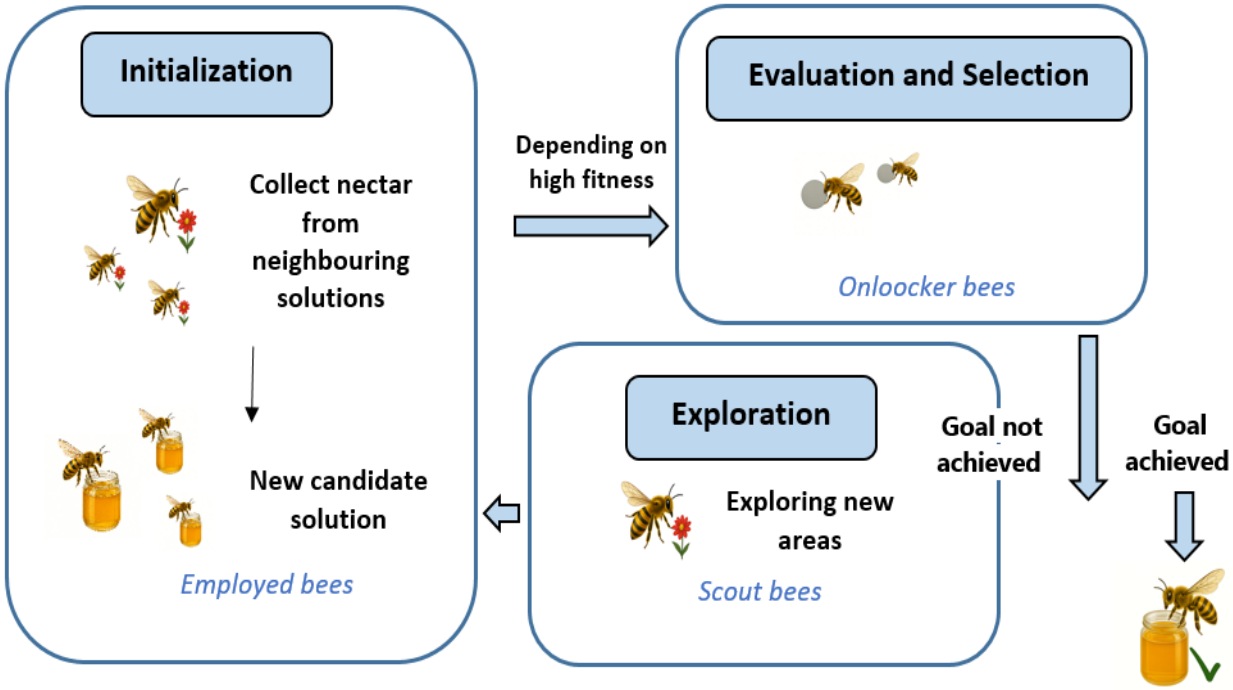

Figure 2.3: Visual representation of the ABC algorithm

2.2.2.4 Genetic Algorithm

GA for Genetic Algorithm is a metaheuristic that originated with John Holland in his book in 1975 [51]. This algorithm is inspired by the process of natural selection. The author explained how individuals could be encoded as strings (binary strings), similar to biological chromosomes, to generate a starting set of solutions. These last are evaluated based on fitness functions. The better individuals will be selected to create the next generation. After that, the most crucial stage begins, starting with the crossover process where selected individuals (parents) exchange genetic material, moving to mutation, which introduces random changes (may flip a bit in a binary representation) in order to explore search space and to improve solution quality. Ultimately, the new generation's individuals replace the individuals with the lowest fitness. In relation to MAS, each agent in MAS corresponds to an individual in the GA, which facilitates genetic operations through social interaction.

The following equation 2.8 represents the common approach used in order to effect the process of selection, called roulette wheel selection [52].

$$P\ (xi) = \frac{f(xi)}{\sum_{j=1}^{N} f(xj)} \qquad (2.8)$$

Where: N – total number of individuals, $f(xi)$ – fitness of individual *xi*.

Figure 2.4 provides a general and visual description of GA process**.** It illustrates its main steps as follows: initialization and evaluation of a population, selection of the fittest individuals, crossover, and the insertion of random variations via mutation. These phases collectively and iteratively simulate natural evolution, allowing the algorithm to explore and optimize potential solutions to a given problem.

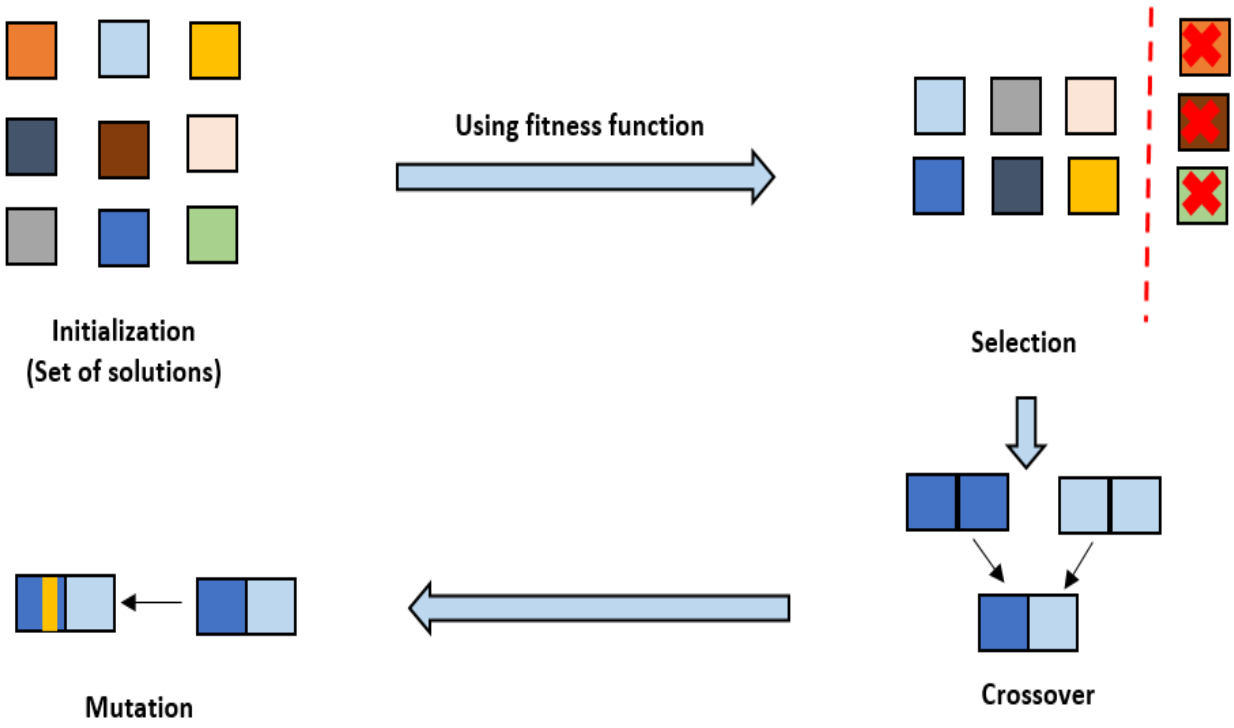

Figure 2.4: Visual representation of the GA

2.2.2.5 Firefly Algorithm

The Firefly Algorithm, or FA, is a new metaheuristic algorithm that appeared firstly in 2008 [53]. This algorithm is inspired by the behavior of fireflies based on the brightness or light emitting of them. Firstly, a random initialization of a population of fireflies in the search space. Each one is associated with a fitness value according to the objective function. After that, each firefly must update its position toward the brightest one, according to its fitness; otherwise, it should move randomly to ensure the exploration. The applications of FA vary in various domains of optimization problems, such as discrete, multi-objective, etc. In relation to MAS, fireflies represent agents or solutions that interact based on their brightness.

Attractiveness is directly proportional to the intensity of light perceived by other fireflies. The attractiveness variation β based on the distance r (here r is equal to 0) and β_0 (denotes the attractiveness) can be expressed in equation 2.9 as follows [54]:

$$\beta = \beta_0 e^{-\gamma r^2} \qquad (2.9)$$

Consider two fireflies, denoted as x_i and x_j. The movement of the first one in the direction of the second one using the shown equation 2.10 [54]:

$$x_i^{t+1} = x_i^t + \beta_0 e^{-\gamma r_{ij}^2} (x_j^t - x_i^t) + \alpha_t \epsilon_t \qquad (2.10)$$

Where: the part $\beta_0 e^{-\gamma r_{ij}^2}$ – the attractiveness of the jth firefly to the ith firefly, the part $(x_j^t - x_i^t)$ – the distance vector between the two fireflies, the part $\alpha_t \epsilon_t$ – a stochastic component to the firefly's movement, ϵ_t – a random vector with the same dimension as the search space. It generates a random direction in the search space. Where the parameter α_t scales this random direction, determining the step size.

It is important to note that, a large value of α promotes the exploration. In contrast, to a smaller value that promotes exploitation.

For more comprehensive details, the following Figure (Figure 2.5) provides a visual representation of FA. It shows the iterative nature of the algorithm and the balance between exploration and exploitation. This process involves randomly initializing the positions of the fireflies, the evaluation of the solution's quality, the movement toward the brighter fireflies, showing that the central firefly represents the optimal solution.

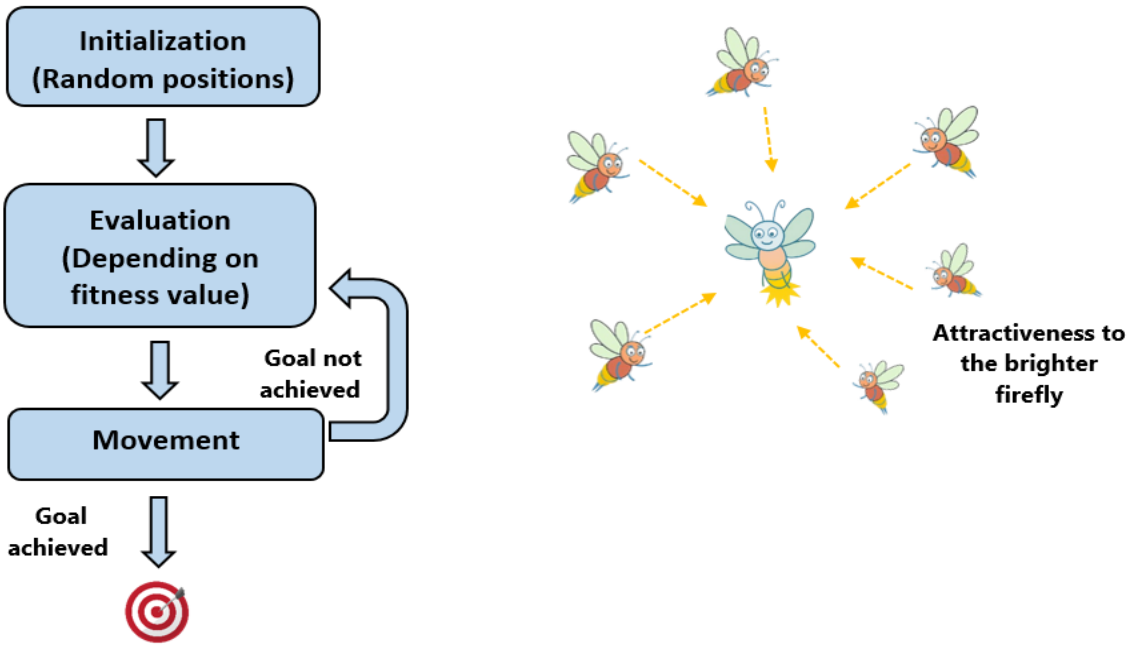

Figure 2.5: Visual representation of the FA

2.2.2.6 Cuckoo Search Algorithm

The Cuckoo Search Algorithm, or CS, is an optimization algorithm based on the natural behavior of cuckoo birds. It was first introduced in a paper [55] in 2009. It is combined with the concept of Lévy

flights, which enable efficient exploration of the search space. The main idea of this new method is the brood parasitism strategy that the cuckoo bird follows. Initially, cuckoos start by laying eggs in other birds' nests randomly. Each post nest represents a potential solution. The eggs represent new solutions. If the new solution (egg) outperforms the existing one (according to its fitness), it is replaced; otherwise, the nest may be abandoned if it lays poor solutions (eggs). Moreover, poor nests use Lévy flights, a random walk with step lengths from a Lévy distribution, used to explore the solution space through large jumps. Ultimately, CS is beneficial for complex optimization tasks where traditional methods may struggle. Specifically, multi-objective problems, such as a multi-objective scheduling algorithm [56].

The equation 2.11 is used to update a solution $X_i^{(t+1)}$ using Lévy flight [57]:

$$X_i^{(t+1)} = Xi^{(t)} + \alpha \oplus Lévy\ (\lambda) \qquad (2.11)$$

Where: $Xi^{(t)}$ – the current position of the cuckoo i ; α – a scaling factor; $\oplus$ – element-wise multiplication, Lévy(λ) – a Lévy flight random variable with parameter λ.

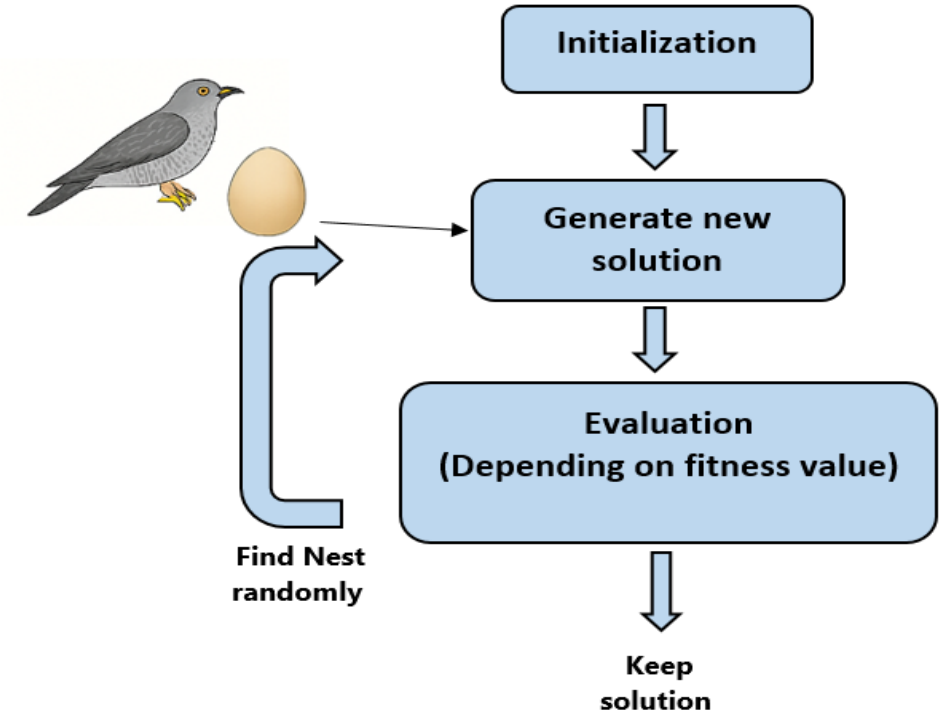

Figure 2.6: Visual representation of the CS

Figure 2.6 illustrates a visual representation of the CS Algorithm. It represents a cuckoo laying an egg in a random nest (random host) as a new candidate solution. The host bird discovers the egg and evaluates the solutions depending on fitness values and decides if it will keep the egg or not.

2.2.2.7 Whale Optimization Algorithm

The WOA for Whale Optimization Algorithm is a nature-inspired optimization algorithm that imitates the hunting strategies of humpback whales. It was originated in the paper [58] of Seyedali Mirjalili. This method is highly effective in solving multidimensional optimization problems. The idea of this method comes from the humpback whales' unique bubble-net feeding technique; they capture fish using bubble trapping, increasing the probability of catching prey. The WOA simulates the idea that was mentioned before by initializing a population of searching whales (which is considered an agent in MAS), assessing each one with a fitness based on an objective function. Subsequently, it calculates the distance between whales and the current best prey in order to update the whales' positions towards them. The algorithm selects a random number *p* between 0 and 1. If it is inferior to 0.5, the process encircles the prey; otherwise, it performs a spiral updating position to ensure exploration of new solutions. Continue the position update and fitness evaluation until a stopping criterion is reached.

In the first case (*p* is inferior to 0.5), the process of encircling the prey satisfies the following equation 2.12 [59]. This equation provides the distance *D* of a search agent *X(t)* from the best solution discovered so far, *X*(t).* Vector *C* is a coefficient vector that governs the wrapping behavior.

$$\vec{D} = |\vec{C} \,.\, \vec{X}^{*}(t) - \vec{X}(t)| \qquad (2.12)$$

The vector *C* is calculated as shown in equation 2.13 [59]. Where $\vec{r}$ is a random vector in [0, 1].

$$\vec{C} = 2 \,.\, \vec{r} \qquad (2.13)$$

Then, the updating position process is acquired with equation 2.14 as follows [59]:

$$\vec{X}(t+1) = \vec{X}^{*}(t) - \vec{A} \,.\, \vec{D} \qquad (2.14)$$

Where: A vector *A* is the coefficient with which the algorithm operates among the exploration vs exploitation process. The agents move away from the best solution when $|A| > 1$, exploring new areas. While in the exploitation phase when $|A| < 1$, they refine the current solution. It is calculated using equation 2.15, where $\vec{a}$ decreases linearly from 2 to 0 [59].

$$\vec{A} = 2\,\vec{a} \,.\, \vec{r} - \vec{a} \qquad (2.15)$$

In the second case, (*p* is superior to 0.5), the spiral updating position is affected using equation 2.16 as follows [59]:

$$\vec{X}(t+1) = \vec{D}' \,.\, e^{bl} \,.\, \cos(2\pi l) + \vec{X}^{*}(t) \qquad (2.16)$$

Where: $\vec{D}' = |\vec{X}^*(t) - \vec{X}(t)|$. l – a random number in [-1,1], b – a constant for a definition of the logarithmic spiral shape.

Figure 2.7 illustrates the core behavioral phases of the WOA, which is inspired by the hunting strategies of humpback whales. It shows the first phase of searching for prey, then encircling it and hunting it.

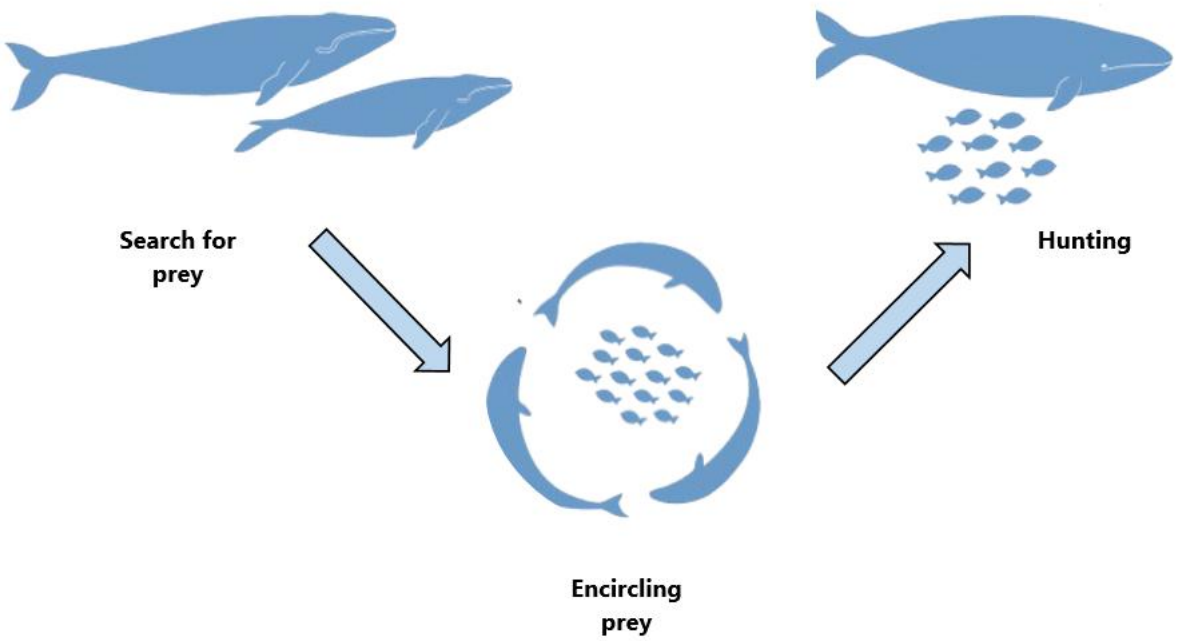

Figure 2.7: Visual representation of the WOA

2.2.2.8 Crow Search Algorithm

The CSA, is a metaheuristic optimization algorithm inspired by crows' intelligent behavior, storing and retrieving food as needed. It is simple and effective in solving optimization problems in different fields like engineering, computer science, etc [60]. The algorithm uses mathematical models to represent crows' social behavior, focusing on memory and balancing between exploration and exploitation. It starts by initializing a population of crows; each one represents a solution in search space and memorizes its best position. After that, the crows update their positions, randomly selecting another crow to steal food using on a random flight length. If the followed crow detects the following crow, it adjusts its position based on the awareness probability parameter. Eventually, the process terminates after a predetermined number of iterations. Crows show cooperative and competitive behaviors. Sometimes they work together to find food. On the other side, they hide food. Similarly, to MAS, the agents may

collaborate to enhance performance. Similarly, to MAS, the agents may collaborate to enhance performance, and their collaboration mimics the interaction between intelligent crows.

Each crow updates its position depending on equation 2.17 [60]:

$$x_{i,iter+1} = \begin{cases} x_{i,iter} + r_i \times fl_{i,iter} \times (m_{j,iter} - x_{i,iter}) & r_j \geq AP_{j,iter} \\ A\ random\ position & otherwise \end{cases} \quad (2.17)$$

Where: $x_{i,iter}$ – the current position of crow i in iteration, r_i,r_j– random numbers, $AP_{j,iter}$ – the awareness probability of crow *j* in the current iteration, $fl_{i,iter}$ – the flight length of crow *i* to remember crow *j*.

If the new position $x_{i,iter+1}$ has a better fitness value than the current memory $m_{i,iter}$. The memory is updated with the new position using equation 2.18 [60]:

$$m_{i,iter+1} = \begin{cases} x_{i,iter+1}\ f\ (x_{i,iter+1}) \leq f\ (m_{i,iter}) \\ mi, iter \end{cases} \quad otherwise \quad (2.18)$$

Figure 2.8 visualizes the updation of the new position of a crow *i* in CSA depending on the critical parameter value of the *fl*, it controls the balance between exploration and exploitation. It shows that when *fl* < *1*, the crow moves toward the target (crow *j*), favoring exploitation of known solutions. In contrast, when *fl* > *1*, the crow explores beyond the known position, promoting exploration of new regions in the search space.incre

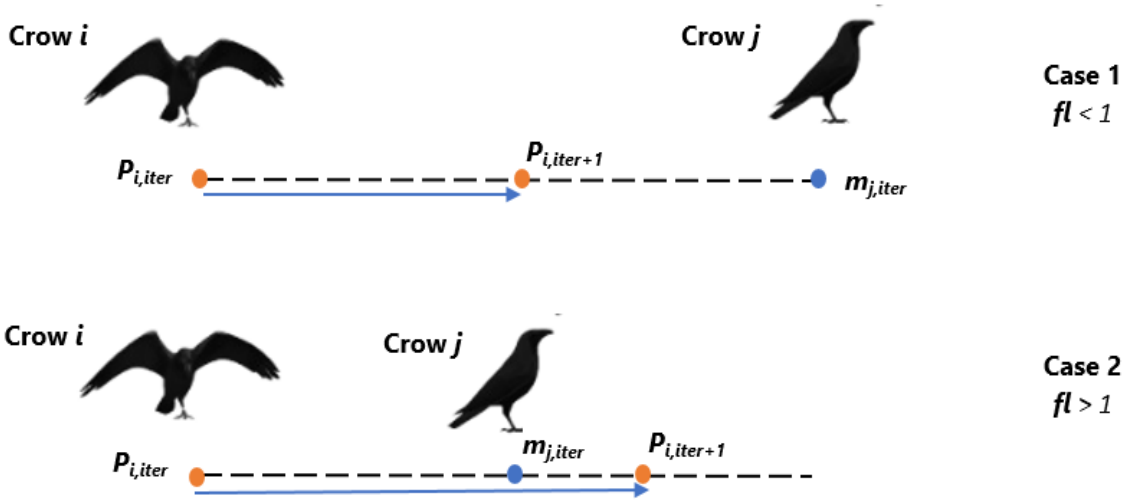

Figure 2.8: Exploration and exploitation in the CSA

2.2.2 Relation between multi-agent systems and metaheuristics

As demonstrated previously, MAS and metaheuristics are powerful tools to tackle challenging tasks. Consequently, combining them is a good way to take advantage of both approaches and to enhance each other's strengths. Therefore, cooperative search involves agents sharing information during the search process. Specifically, metaheuristics can be direct communication or indirect memory-based, sending information to a pool for other metaheuristics to utilize as needed. Metaheuristics can be distributed among multiple agents in MAS, enabling parallel exploration of the solution space. Researchers in [61], highlighted that the PSO metaheuristic is an adaptation of bee, fish, and bird behavior, enabling decentralized control, coordination, and efficient communication between large numbers of particles. Also, authors in [62] represent a robust framework that seeks to organize and improve the field of metaheuristics through the use of a MAS approach, allowing for a better understanding and efficient implementation of these algorithms. Moreover, the advantages of population-based Metaheuristic Algorithms (PMAs) over Single-agent Metaheuristic Algorithms (SMAs) in optimization tasks are discussed in [63]. It shows that PMAs utilize multiple agents that collaboratively share information, allowing for a more comprehensive exploration of the search space, which increases the likelihood of finding the optimal solution. The optimization process begins with a diverse population of agents positioned in various locations within the search space, and through iterations, they work together to identify the best global solution.

Concisely, on one side, metaheuristics can be distributed among agents in MAS for parallel exploration. On the other side, complex systems are modeled by MAS as interacting agents that understand and modify their behavior in response to interactions and their environment to solve problems effectively. These agents' performance can be improved by using metaheuristics. Ultimately, it must be admitted that this combination has the potential to revolutionize various fields.

2.3 Application of metaheuristics in multi-agent systems path planning

This section discusses exactly how metaheuristic algorithms can be used in the context of MAS path planning. It provides a comprehensive review of existing methodologies, critically examining the efficacy and limitations of them. A detailed analysis of these studies, presented in tabular form, offers a comparative perspective on their methodologies.

Metaheuristics in MAS improve path planning in dynamic environments, navigation for robots, vehicle transportation, and other applications. Based on this approach, researchers can develop and enhance solutions for difficult MAS path planning problems. There are many works aiming to evaluate the effectiveness of different metaheuristic algorithms for MAS path planning, assessing their performance,

strengths, and limitations, such as [64]. It provides a comprehensive overview of the application of metaheuristic algorithms in UAV path planning, highlighting advantages, limitations, and perspectives. Another one in [65], this work focuses on various path planning algorithms designed for UAVs. It categorizes classical and modern approaches and performs an empirical evaluation of their performance. Moreover, the case study in [66] focuses on employing swarm intelligence techniques to elevate the path planning performance of drones.

Accordingly, works were selected as samples for analysis, more detailed in Table 2.1, including [67–70] all of which revolve around applying the PSO method to solve MAS path planning. The first one proposes a PSO-based framework for cooperative task assignment and path planning in MASs The second one represents a modified PSO algorithm that improves path planning for UAVs by adjusting *IW* over time, balancing exploration and exploitation, and demonstrating superior performance compared to standard techniques. The third one presents an advanced approach to cooperative path planning for multiple Autonomous Underwater Vehicles (AUVs). This enhancement addresses the limitations of traditional PSO. It ensures a variety of places to start the search. In addition, it adjusts its behavior based on the problem and explores the local neighborhood, improving their quality. Moreover, the proposed method uses the technique of cubic spline interpolation to smooth the planned paths, making them more realistic and efficient. Finally, its objective function is based on various factors. Lastly, [70] introduces a hybrid approach that integrates the WOA and PSO to enhance path planning efficiency for differential wheeled mobile robots. In addition to these works, another article [71] examines and compares the ABC and ACO metaheuristic algorithms. It evaluates their effectiveness in path planning for mobile robots. The study reveals that ABC outperforms other methods in specific environments, indicating its effectiveness in MAS path planning. Moreover, there are other works like [72,73], They both present a novel approach to address the challenges of MAPP using an enhanced ACO algorithm. The authors adjust the transition probability formula to consider inter-agent interactions and dynamic obstacles. In addition, the algorithm updates the pheromone strength dynamically to improve exploration and exploitation. In order to promote shorter paths for better exploration and exploitation, it incorporates a reduction factor to avoid local optima. Besides this, there exists additional work [74]. Its strategy utilizes a group of MAS using hybridization of the algorithms ACO and GA to cover a specific area effectively and efficiently, with determining the most suitable routes for them to achieve maximum coverage. The common point among the last three works is the use of the ACO algorithm. Regarding the ABC algorithm, many works show how flexible this approach is for dealing with challenges of MAS path planning, especially in dynamic environments such as [75]. Turning to the GA, which is another powerful, popular, and widely used algorithm, numerous studies

have explored solutions to MAS path planning, such as [76], which proposes an enhanced crossover operator for GA path planning in static environments. Moreover, the authors in [77] design a hybrid controller for humanoid robot path planning that combines neural networks and GA. Lastly, a novel hybridization is presented in [78]. The study provides a novel method for MAS path planning by combining the FA and the CS algorithms.

Eventually, it is worth mentioning that there are hundreds, if not thousands, of solutions. In this regard, only the latest ones have been addressed. However, a detailed breakdown of them is provided in Table 2.1.

Metaheuristics algorithm	Reference	Strength	Limitation	Future perspectives
PSO	[67]	- Enhancing the safety of paths (treating other agents as dynamic obstacles). - Encourages cooperation. - Adaptability for various scenarios (different simulations with various environmental complexities).	- Limited scalability (number of agents).	- Applying the framework to a larger number of agents.
	[68]	- Similar set of advantages with [67]. - Balancing exploration/exploitation. - Addressing the limitation of trapping in local optima (it is not mentioned in [67]).	- Parameter sensitivity (w). - Limited scalability (offline path planning is less scalable for dynamic environments).	- Multivariable w.
	[69]	- Preventing local optima in complex 3D environments. - Effective path planning for multi-AUVs with high security, while [68]	- The algorithm would handle highly unpredicted obstacles and changes in the environment in real-time.	- Enhancing real-time coordination. - Incorporating real-time sensing to address unpredictable environments.

		focuses on the PSO algorithm itself.		
	[70]	- Integrating PSO's rapid convergence with WOA's deep exploration capabilities. - Balancing exploration/exploitation.	- Complexity of implementation. - Limited scalability (lacks evidence for large-scale robot performance). - Parameter sensitivity	- Real and dynamic environments testing. - Exploration of Other Hybridizations.
ACO	[71]	- Ensuring collision-free paths.	- Parameter Sensitivity (performance is highly dependent on chosen values).	- Creating an algorithm that is less sensitive to parameter tuning.
	[72,73]	- Handling static and dynamic obstacles. - Using an improved ACO avoids local optima.	- Limited scalability (computational cost increases with the number of agents and environment size).	- Real-time applications. - Applying it to more complex scenarios
	[74]	- The agents cooperate even though they have different characteristics.	- The algorithms might perform badly in contexts that change fast (offline or static planning approach).	- Hybridizing strategies for environmental changes in real-time.
ABC	[71]	- Ensuring collision-free paths. - The ABC algorithm outperformed ACO.	- Parameter Sensitivity (performance is highly dependent on chosen values).	- Creating an algorithm that is less sensitive to parameter tuning.
	[75]	- Flexibility and convergence speeds due to its stochastic nature.	- Limited scalability (high computational cost for managing pheromones and agent interactions as system size and complexity increase).	- Hybridizing strategies for real-time scenarios.

			-Trapping in local optima.	
GA	[76]	- Creating feasible paths.	- The algorithms might perform badly in a dynamic environment.	- Applying the algorithm for dynamic environments.
	[77]	- The close alignment of simulation and experiment	- Difficulty of combining GA/ NN, with sensitive parameter selection.	- Collaborative control among multiple humanoid robots.
FA and CS	[78]	- Capitalizing on the exploration of the FA and the exploitation of CS.	- Parameter Sensitivity. - Computational Complexity.	- Hybridization with parameter tuning.

Table 2.1: Analysis of metaheuristic algorithms for MAS Path Planning

To summarize, the most significant result of this analysis pertains to the choice of metaheuristic; it should be tailored to the particular path-planning problem. The process involves considering factors such as the number of agents, the complexity of the environment, the optimal balance between exploration and exploitation, parameter sensitivity, and the computational efficiency. Moreover, future research could emphasize developing hybrid MAS algorithms and optimizing solution quality in complex scenarios.

2.4 Application of metaheuristics in multi-agent systems task coordination

This section delves into the application of metaheuristic algorithms in the context of MAS task coordination. A comprehensive review of these studies provides contributions to the field.

The application of metaheuristics in MAS task coordination delivers significant improvements. That is why researchers are conducting a thorough investigation into the potential of these techniques to optimize complex task allocation and scheduling problems. Consequently, the following works were selected for detailed analysis, which is provided in Table 2.2.

Initially, the work [67] presented previously presents a framework that uses PSO for efficiently managing cooperative task assignment in MASs. Within the same context, there is another study that gives a hybrid approach called Discrete Differential Evolution PSO (DEPSO). The proposed integrates elements of both DE and PSO in order to tackle the issue of discrete task allocation problems in MASs, especially in epidemic scenarios [79]. Another approach presented in [80], was designed in order to

optimize scheduling and task distribution over several satellites. It combines both strengths of GA and SA for simulated annealing. Further, another study in [81] suggests an enhanced ACO algorithm called Max-Min Ant System. It handles the scheduling and task optimization difficulties in MASs. Utilizing graph theory's optimization techniques, the approach explores solution spaces for optimal task scheduling, minimizing path weight and overall completion time. Moreover, the paper [82] presents a new multi-objective ant colony system, MOACS. It provides also a version of ACO. The goal here is to distribute tasks to robots in a manner that reduces the overall cost and makes sure a fair workload distribution while taking into account different task durations and travel times. A related article explores a hybrid algorithm for task scheduling in multi-processor systems. It combines ACO and a variant of the Cuckoo Search Algorithm CVOA [83]. Lastly, an application of a multi-objective CSA aiming to optimize resource management in cloud data centers presented by authors in [84].

Metaheuristics algorithm	Reference	Strength	Limitation	Future perspectives
PSO	[67]	- Coordinating tasks among several of the agents. - Encourages cooperation (agents negotiate task assignments).	- Scalability issues (the total computation time increases with the increase in task assignment). - Agent heterogeneity (in terms of differing capabilities or roles)	- Developing a more efficient task assignment strategy.
	[79]	- Relevance to Real-World Scenarios. - Balancing exploration/exploitation.	- Limited scalability (in large-scale scenarios with many tasks).	- Dealing with complex scenarios.
GA	[80]	- Managing a large number of satellites and tasks.	- Complex implementation (combining GA and SA). - Parameter Sensitivity (Specific Parameters for Task Scenarios).	- Optimizing energy usage for satellites and hybridizing.
ACO	[81]	- Performance in terms of completion time.	- The algorithms might perform badly in dynamic environments	- Hybridization of the enhanced ACO

			(frequent changes, unexpected events).	with other techniques.
	[82]	- Reducing both travel distance and total task completion time.	- Complexity of computing time (challenges scalability to massive multi-robot systems).	- Handling Dynamic and Online Scenarios.
	[83]	- Encouraging exploration and exploitation.	- Complexity of computing time and computational resources.	- Real-time scheduling scenarios.
CSA	[84]	- Balancing resource utilization and energy efficiency.	- Limited scalability (the centralized approach limits its scalability for massive, real-world cloud data centers). - Computational complexity	- Hybridizing and investigating scalability.

Table 2.2: Analysis of metaheuristic algorithms for MAS task coordination

To summarize, a principal result of this analysis indicates that selecting a suitable metaheuristic is essential to solve particular task coordination issues. The strategic choice necessitates the evaluation of factors such as the number of agents, agents' heterogeneity, the complexity of the environment and parameter sensitivity. Understanding the significance of this selection, can result in better results and highly efficient solutions applicable to various applications. Moreover, Future research could emphasize on developing hybrid MAS algorithms, dealing with complex and dynamic environment.

2.5 Summary

In this chapter, a detailed overview of notable research findings and recent progress in the application of metaheuristics in each of path planning and task coordination MAS was provided. To accomplish this, the key concepts introduced were metaheuristic algorithms and their relationship to MAS. Then, cases of study were selected, and the most important points of advantages and limitations of each one were analyzed. Some cases, including those based on PSO, GA, ACO, etc., are expressed in terms of path planning, while others are expressed in terms of task coordination. The most important results reached lie in the great agreement between the studied cases on various aspects, whether positive or negative. Moreover, the comparative analysis shows that does not exist metaheuristic that dominates all scenarios; rather, the selection of the algorithm frequently relies on specific problem constraints,

requirement for scalability, convergence speed, and the balance between exploration and exploitation. Hybridization and adaptive control mechanisms are emerging as promising trends to enhance the robustness and efficiency of these algorithms, especially in dynamic and decentralized MAS settings. Ultimately, it is worth noting that a suitable metaheuristic must be selected to solve particular task coordination or path planning MAS issues. In future work, development efforts will focus on other enhanced versions of MAS algorithms or hybrid ones, dealing with complex and dynamic environments, adjusting parameters, and real-world testing.

The next chapters will present the simulation framework designed to evaluate the performance of selected metaheuristic strategies in MAS path planning.

CHAPTER 3: IB-PSO: AN INVERSED BUTTER WORTH PARTICLE SWARM OPTIMIZATION ALGORITHM FOR MULTI-AGENT PATH PLANNING

3.1 Introduction

MAS is considered as one of the major research that can be consistent with human behavior and make decisions by reasoning through a set of intelligent entities known as agents that interact with each other in distributed way to solve complex problems. Unlike single systems, MASs have high efficiency due to the use of several mechanisms, such as negotiation, cooperation, and coordination. Moreover, Path planning is considered as the coordination of these resulting paths to ensure a conflict-free and optimal movement for the entire multi-agent collective in the shortest period possible. The challenge of MAS path planning becomes more difficult when the number of agents is significant. Meta-heuristic techniques are hugely used to solve complex optimization problems such as, path planning. The main objective of metaheuristics is to approximate the optimal solution by maximizing or minimizing the objective function according to the processed problem. Additionally, PSO is one of these metaheuristics; it is based on a swarm of particles that update their positions and velocities in relation to its personal best solution and the global best solution found in the search space. The strength of the PSO algorithm lies in its ability to combine and balance the exploration of the search space and the exploitation of the discovered solutions. This balance is primarily controlled by the inertia weight (*IW*) parameter, which governs the extent to which a particle retains its previous velocity. Inspired by all of these, the main objective of this chapter is to propose a new MAS path planning based on an inversed Butterworth *IW* in the PSO algorithm, known as IB-PSO. The aim of the proposed algorithm is to provide a novel inertia calculation that provides an adequate balance between the exploration and the exploitation of the search space. To achieve this, the main goals of this chapter can be summarized as follows:

- The proposition of a MAS path planning algorithm based on an extension of PSO algorithm.
- The introduction of a new increasing *IW* calculation method aiming to balance the exploration and the exploitation of the search space.
- The evaluation of the increasing *IW* method.
- The comparison of the proposed PSO version with recent approaches aiming to control the *IW* parameter through their application to path planning problem.

The chapter is organized as follows: Section 1 is dedicated to the introduction. Section 2 discusses the recent research activities regarding the applications of metaheuristics in MAS by focusing on the PSO application, justifying the improved *IW* formulation. Section 3 first discusses different versions of the PSO algorithm and their respective equations. Subsequently, it outlines the procedure for updating agent locations within the path-planning framework. Section 4 reflects a detailed description of the proposed

path-planning algorithm. Section 5 analyzes the main obtained simulation results in comparison with related approaches. Finally, a conclusion summarizes the main contributions of this chapter in Section 6.

3.2 Critical Review of Metaheuristic Performance in Multi-Agent Systems

The aim of this section is to provide an analysis of the previous research activities related to the proposed work in this chapter. Specifically, consideration will be given to the most important and recent MAS research activities based on metaheuristic approaches. It addresses deficiencies in parameter adaptation and justifies the improved *IW* formulation.

There exist diverse applications of metaheuristic algorithms for solving optimization problems in real-life systems with complex configurations. In [85], the authors proposed a MAS model based on metaheuristics via reusable codes (MOF for Metaheuristic Optimization Framework), facilitating the development of applications for solving optimization problems. The study focused on the traits of hybridization, collaboration and the integration of MASs. Building upon the work proposed in [86], the authors introduced a different MAS optimization approach for routing and scheduling, comparing various methods. In the same context, a recent research using metaheuristics to address the same problem was proposed by Kamali, et al in [87]. Precisely, the authors presented a novel approach to enhance scheduling efficiency by integrating immune-based concepts with MAS in order to Contribute in the growing field of optimization solutions.

Concerning MAS task coordination, metaheuristics are usually proposed. In this kind of problem, each agent has a local search space in order to collaborate and find a global solution. For example, in [88], the educational timetable problem was solved with metaheuristic methods based on distributed MAS. Moreover, the authors discussed the challenges and advancements in this field, while summarizing the used metaheuristic techniques.

Swarm intelligence techniques are often used in combination with MAS in order to solve different problems. In [89], the authors introduced a novel metaheuristic known as the Total Interaction Algorithm (TIA). This method is a combination of five swarm intelligence algorithms developed through the coordination their competencies. Specifically, TIA depends on the agents' interactions to find the best answers to the processed complex problems.

Recently, the PSO algorithm has been applied to several MASs-based problems. In [90], the authors proposed a PSO-solving method based on distributed systems, by proving its effectiveness in solving the Flexible Job-Shop Scheduling Problems (FJSP). The main objective of the discussed approach is to manage a large number of entities by using a Multi-Objective Particle Swarm Optimization method (MOPSO). In addition, Dai et al [91] solved a smart Electric Vehicles charging station problem by using

solar energy. In other words, the authors have based on physical agents and the application of MAS-PSO with the aim of optimizing electricity and energy consumption. In another context [92], the authors solved the Multi-Pursuers Multi-Evader Game (MPMEG) via the use of a discrete approach of the PSO algorithm. The objective of the discussed study is to enhance the pursuers' reward acquisition, which is a measure of their learning rate during the task execution. Moreover, the authors tried to find a dynamic pursuers' grouping by comparing the results obtained with recent approaches. In another work [93], the authors introduced a combination of the PSO algorithm with other methods, which are the Ant Colony Optimization (ACO) and K-Opt Operation. In the sense that the role of the PSO comes after ACO, which is providing the swarm a sufficient number of times to find an optimal path in comparison with the existing algorithms to address the Traveling Salesman Problem.

MAS Path planning problems have been hugely processed by employing several PSO approaches. In this context, Biswas et al [94] used a new vectorized PSO extension (VPSO), for autonomous particles path planning and obstacle avoidance. Furthermore, the authors proved that their proposed optimization algorithm provides a safe navigation ability by guarantying a collision free path. In addition, the Unmanned Aerial Vehicles (UAVs) path-planning problem is becoming a very important research area. In this kind of problems, the goal is to optimize the flight trajectory of drones. In [95], the authors introduced a coordination mechanism for multiple UAVs to meet at a location in space and time using Distributed Cooperative Particle Swarm Optimization (DCPSO). Specifically, the discussed proposal focused on the study of the algorithm initialization, the update of velocities and positions, and the fitness function. In addition, they also provided an evaluation of the time complexity as well as simulation results for both 2-D and 3-D formation.

In relation to the proposed approach in this chapter, several works have based on the modification of the *IW* calculation in order to improve the PSO efficiency. In [66], Nayeem et al focused on enhancing the PSO algorithm for UAV path planning by providing a comparative analysis of different versions of *IW* that have been proposed, such as constant value, time-varying, single variable, and multivariable *IW*. The authors also illustrated which inertia calculation approach produces the best UAV path planning among all algorithms and strikes a better balance between exploration and exploitation parameters. Moreover in [96], an Improved PSO (IPSO) was proposed with the aim of finding the best path with collision-free using the minimal number of iterations possible and providing a balance between local and global optimization skills. Also, the authors have based on uniform initialization of the particles' positions in the search space, exponential attenuation *IW* by using an exponential function, cubic spline interpolation function, and learning factor of enhanced control aiming to decrease the cosine function of *IW*. The results were compared with classical PSO. Additionally, two other relevant works deal with

the different kinds of *IW* in [97,98]. The first approach [97] proved that various *IW* strategies, such as simulated annealing *IW* and linearly decreasing *IW* can enhance PSO's convergence speed and search performance through their application to the path-planning problem. The second approach [98] focused on the resolution of the economic load dispatch problem by applying the different *IW* functions.

Other research activities have focused on the increased *IW* calculation, such as [99,100]. In [99], Conducts a comparative analysis of 15 *IW* strategies applied to 5 optimization problem, such as linearly decreasing *IW* PSO, fuzzy adaptive *IW* PSO, and random *IW* PSO. Also in, [100] the proposed algorithm, known as Adaptive *IW* PSO (SAIW-PSO), achieves relevant performance and outperforms other PSO extensions. Moreover, the authors described how increasing *IW* can be advantageous. Also, Zheng, et al [101] proved that an increasing *IW* outperforms a decreasing approach in some case and according to the processed problem.

The main problem in current MAS metaheuristics is poor parameter tuning, which stops them from properly balancing exploration and exploitation. This failure, specifically the inability to increase speed (velocity) and search widely enough, makes existing *IW* methods get stuck in local optima. Therefore, a novel and robust improved *IW* method is needed, which is specifically designed to increase the *IW* to guarantee successful global exploration.

3.3 Problem description

This section describes different versions of the PSO algorithm, its different equations. Moreover, it describes how agents change their locations in the path-planning problem.

3.3.1 The compared PSO versions

Knowing that the main objective of this chapter is to propose a new *IW* calculation, the usefulness of discussing the main *IW* calculation methods of the literature review has been observed. The *IW* can be represented by a constant value or adjusted dynamically during the execution process. *IW* aims to measure how the previous velocity of the particle impacts its current velocity. This parameter also controls the balance between exploration and exploitation of the search space. On the one hand, a high *IW* value favorizes the exploration of the search space. On the other hand, a low *IW* value favorizes the exploitation process. Usually, the *IW* values used are in the range of 0.4 to 0.9.

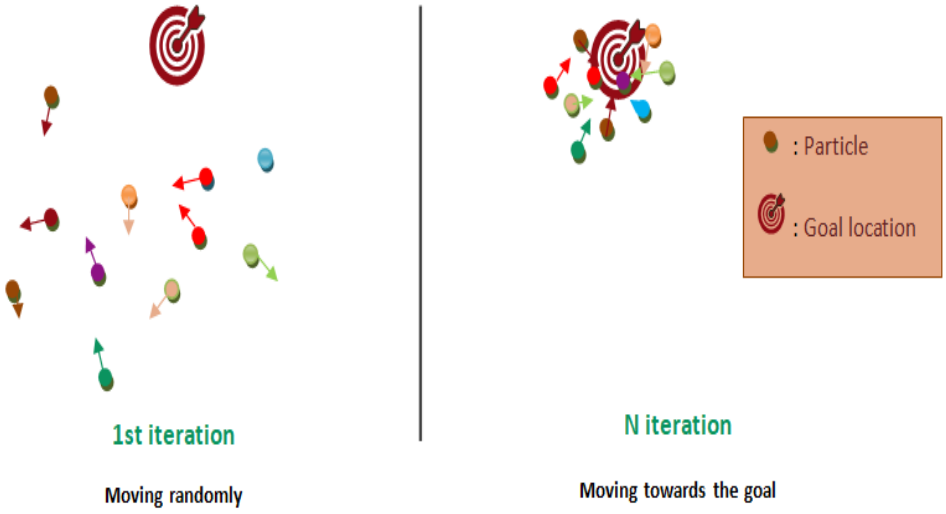

Figure 3.1: Particle motions in PSO Algorithm

In order to enhance the PSO algorithm, different extensions based on the *IW* control were proposed:

- **Constant IW** [97,102]: in this PSO extension, the *IW* is considered as static during the algorithm's iterations. The *IW* is calculated as follows:

$$w = \frac{1}{2*ln(2)} = 0.729 \qquad (3.1)$$

- **Linearly decreasing IW** [98,99,102]: in this version, the *IW* value linearly decreases during the execution from w_{max} to w_{min} by taking into consideration the maximum number of iterations (*MaxIter*):

$$w_t = w_{max} - \frac{Wmax - Wmin}{MaxIter} * t \qquad (3.2)$$

Where: $w_{min}= 0.4$, $w_{max}=0.9$, *t:* the current iteration.

- **Natural exponent Strategy 1 (e1-PSO) IW** [98]: in this case, the *IW* decreases in a non-linear way according to the following equation:

$$w_t = w_{min} + (w_{max} - w_{min})*e^{-[t/(MaxIter/10)]} \qquad (3.3)$$

Where: $w_{min}= 0.4$ and $w_{max}=0.9$.

- **Natural exponent Strategy 2 (e2-PSO) IW** [98]: this version has the same principles of e1-pso, however, the *IW* is differently calculated as follows:

$$w_t = w_{min} + (w_{max} - w_{min})*e^{-[t/MaxIter/4)]\wedge 2} \qquad (3.4)$$

Where: $w_{min}= 0.4$, $w_{max}=0.9$.

- Simulated Annealing IW [98]: this decreasing version takes into consideration a constant value (λ) instead of the maximum number of iterations. The *IW* is calculated during each iteration as follows:

$$w_t = w_{min} + [(w_{max} - w_{min}) * \lambda^{t-1}] \qquad (3.5)$$

Where: $\lambda = 0.95$.

- Standard IW [103,104]: this version refers to the original PSO algorithm, in which the *IW* is assumed to be neutral (*IW = 1*).

- Butterworth IW [102]: this non-linear decreasing *IW* version is inspired by the curve of Butterworth filter in electrical engineering. This type of filter is often used in signal processing [102]. The Butterworth IW is calculated as follows:

$$w_t = w_{max} * \frac{1}{1+(\frac{t}{p1})^{p2}} + w_{min} \qquad (3.6)$$

Where $p1 = \frac{MaxIter}{3}$, $p2 = 10$, $w_{max} = 0.5$; $w_{min} = 0.4$.

- Linearly Increasing IW [99–101]: in this case, the *IW* value linearly increases in relation to the algorithm's iterations. *IW* is calculated as follows:

$$w_t = w_{min} - \frac{Wmin - Wmax}{MaxIter} * t \qquad (3.7)$$

Where: $w_{min} = 0.4$, and $w_{max} = 0.9$.

3.3.2 Agent displacement for collaborative path planning

There exists a wide range of optimization problems, from simple optimization problems to more challenging ones. In many fields, optimization is considered as essential process aiming to improve the solution and to reduce the costs, especially for complex problems. In other words, optimization concerns the process aiming to find the best solution among a set of feasible solutions. The main objective of the optimization is to determine the maximum or minimum value that the objective function may achieve while adhering to a set of constraints or limitations, as shown in equation (3.8):

$$Minimize\ or\ Maximize\ f(x),\ x \in A\ and\ x = (x1, x2..xN)$$

$$Subject\ to\ g(x) \leq 0\ and\ h(x) = 0 \qquad (3.8)$$

Where *f(x)* is the objective function; *A* is a set of findable solutions, which can be discrete (specific elements) or continuous (numbers inside a range); *g(x)* are the inequality constraints; *h(x)* are the equality constraints; N is the problem dimension size.

Recently, metaheuristic approaches are often used as optimization methods in complex problems. PSO [44] is considered a metaheuristic approach aiming to provide an equilibrium between the exploration and the exploitation of the search space. This method is based on a population of particles. Each particle represents an agent in MAS. The main goal of the particles is to reach the best possible solution in the search space. The particles update their positions $Xi = (x1, x2,... xN)$ in each iteration and navigate in their shared environment with the velocity $Vi = (v1, v2,...vN)$ established by their own experience or the experiences of the swarm's neighbors. The velocity and position equations are described in section 2 of Chapter 2 of this thesis (equation 2.1 and 2.2).

The primary goal of MAS path planning is to determine the most efficient trajectory for every agent in the environment (this is detailed in Section 3 of Chapter 1). To address this issue, PSO is used in order to represents each agent as a particle that indicates possible solutions through its dynamic positions. In other word, each particle (agent) iteratively updates its position based on its Cartesian coordinates and according to the PSO principles cited in the previous chapter as follows:

$$v_x(t + 1) = wv(t) + c_1r_1(pBest(t) — x(t)) + c_2r_2\ (gBest(t) - x(t)) \qquad (3.9)$$

$$v_y(t + 1) = wv(t) + c_1r_1(pBest(t) — y(t)) + c_2r_2\ (gBest(t) - y(t)) \qquad (3.9)$$

$$(x(t + 1), y\ (t + 1)) = \begin{cases} x(t\ +\ 1) = x(t) + v_x(t\ +\ 1) \\ y(t\ +\ 1) = y(t) + v_y(t\ +\ 1) \end{cases} \qquad (3.10)$$

(x, y) : the Cartesian coordinates of the concerned agent.

In order to move to its new environment position, the agent uses the magnitude of its velocity as follows:

$$Move\text{-}to(x, y) = \sqrt{v_x \times v_x + v_y \times v_y} \qquad (3.11)$$

The PSO takes the fitness score as a measure of a path's quality. Specifically, each position in the environment is assigned to a fitness value in the range of 0 to 1. The main objective of the PSO path planning is to allow the agents to achieve the positions with the highest fitness degree. Moreover, it is noted that only one environment position returns the maximum fitness degree. As a first step, a random fitness value in the range of 0 to 1 is generated in each environment position. After that, the fitness degree (f) of each environment position is updated as follows:

$$\theta = Maximize\ Val(x_i)$$

$$f(x_i) = max\_fitness * (x_i - min\_val) / (max\_val - min\_val) \qquad (3.12)$$

Where:

Θ: the objective function.

max_fitness: it is used to normalize the position value between 0 and 1 (*max_fitness = 1*).

min_val: the minimum fitness value randomly generated in the environment.

Max_val: the maximum fitness value randomly generated in the environment.

3.4 The proposed Multi-Agent Path Planning Algorithm

This section introduces the proposed MAS path planning based on the new PSO version. Specifically, it details how this approach is used in order to allow the agents to achieve the objective state in minimum time.

Algorithm 3.1: An Inversed Butterworth PSO algorithm for MAS path planning

Inputs: Found-Target = false;

setup-environment;

Initialize (population-size);

Initialize (target-position);

Initialize (number-iteration-max);

Initialize (IW,c1, r1, c2, r2);

Foreach $Agent_i$ **do**

Initialize ($position_i$);

Initialize ($velocity_i$);

$PBest_i \leftarrow position_i$;

End for;

searching-time ← 0;

target-value ← Evaluate-fitness($target\text{-}position_i$);

Gbest ← Max (PBest);

while number-iteration ≠ number-iteration-max **or** Found-Target ≠ true **do**

Foreach $Agent_i$ **do**

fitness-value$_i$ ← Evaluate-fitness(position$_i$);

PBest-value$_i$ ← Evaluate-fitness(PBest$_i$);

Communicate-with-the-environment(Agent$_i$);

If fitness-value$_i$ = target-value **then**

Found-Target ← true;

Elseif fitness-value$_i$ is better than PBest-value$_i$ **then**

Update PBest$_i$;

Gbest ← Max (Pbest);

End if;

If target-value$_i$ = false **and** number-iteration ≠ number-iteration-max **then**

Position$_i$ ← Move- toward(Agent$_i$, GBest);

IW ← Calculate-inertia (IW); ***//using the new equation***

update-velocity(IW, position$_i$);

End if;

End for;

searching-time ← searching-time + 1;

end while;

Outputs: searching-time, Found-Target;

The different algorithm steps can be explained as follows:

The main objective of this algorithm is to generate the trajectory for all agents in order to find the location of the global optimum, which is known as the location with the best fitness-value. At the beginning of the proposed path planning, the agents are created and placed randomly in the environment. In relation to the PSO principles, each agent represents a particle ($i \in [1, population\ size]$). Thus, the population size represents the number of existing agents. The population size is initialized and fixed at the beginning of each algorithm's episode. At each iteration, each agent updates its velocity

according to their cartesian coordinates as shown in equation (3.9). Known that, the initial velocity of the agents is assumed to be zero. According to the obtained velocity, the position of each agent in the environment is updated as detailed in equation (3.10). Moreover, it is noted that the constant parameters *c1* and *c2* used for calculating velocity are initialized at the beginning. However, the dynamic parameters *r1, r2,* as well as the *IW* are updated during each iteration.

During each iteration, *IW* is updated according to the proposed function known as the Inversed Butter worth *IW* (IBPSO), detailed in equation (3.13). The proposed *IW* calculation is inspired by the BPSO reflected in equation (3.6). However, an inversed process has been adapted in order to allow the exploitation of obtained solutions at the first iterations and the exploration of the search space in the last iterations. Specifically, in BPSO, the *IW* parameter decreases from 0.9 to 0.4 during the different iteration. On the other hand, in the proposed IBPSO, the *IW* parameter increases from 0.4 to 0.9 as follows:

$$W = w_{max} \div \left(\frac{1}{1 + (\frac{p1}{t})^{p2}} \right) + w_{min} \qquad (3.13)$$

Where: $p1 = (MaxIter)/3$, $p2 = 10$, $w_{max} = 0.5$; $w_{min} = 0.4$

Increasing the *IW* parameter during the algorithm's iterations improves the exploration of the search space. In the path planning case, the exploration principle allows the agents to move far away from their current positions to discover a broader range of possibilities. In [106], the authors indicated that an increasing *IW* parameter in PSO reflects an aggressive movement towards the solution region. In addition, a high *IW* parameter allows the particles to escape problems related to the local minima. Also, increasing *IW* improves the PSO convergence by preventing the algorithm from deviating too far from the optimal direction. Furthermore, the strategy of increasing *IW* parameter is generally competitive and outperforms a decreasing *IW* one in terms of convergence speed and solution precision [99,101,106]**.**

After the update of the agents' positions, the *PBest* position of each agent will be updated. In order to update its *PBest* position, each agent compares the fitness value of its current position with the fitness value of its *PBest* position according to equation (3.14). Knowing that, the initial position of each agent is considered as its initial *PBest* position. The *GBest* position is also updated after each iteration. The *GBest* position is chosen among the *PBest* positions found by the swarm. Precisely, the *Pbest* postion with the highest fitness degree is considered as the new *GBest* position according to equation (3.15).

$$PBest_i = \begin{cases} PBest_i \ \boldsymbol{if}\ (PBest - value_i > fitness - value_i) \\ position_i \ \boldsymbol{else} \end{cases} \qquad (3.14)$$

Where:

-*PBest-value*$_i$*:* represents the fitness value associated to the *PBest*$_i$ position.

-*position*$_i$*:* refers to the current position of the agent *i*.

$$GBest_i = \begin{cases} GBest \ \boldsymbol{if} \ (GBest - value \ > \ Max \ (Pbest - value)) \\ Max \ (Pbest) \ \boldsymbol{else} \end{cases} \quad (3.15)$$

Where:

-*GBest-value:* is the fitness value associated with the *GBest* position.

-*Max (PBest):* returns the position associated with the maximum *PBest* obtained by the swarm.

-*Max (Pbest-value):* the maximum value of *PBest* obtained by the swarm.

The algorithm will be executed until the stopping conditions are met. In other words, the algorithm will stop in the case of reaching the maximum number of iterations or finding the position with the highest fitness degree. At the end, the proposed algorithm returns the used number of iterations as well as if the position with the highest fitness degree was reached:

$$found\text{-}target = \begin{cases} True \ \boldsymbol{if} \ GBest - value \ = \ 1 \\ false \ \boldsymbol{else} \end{cases} \quad (3.16)$$

3.5 Simulation results

This section aims to compare the proposed path planning method with other approaches based on PSO principles. The simulations for this study were performed using the NetLogo platform [107], version 5.3.1, on a machine featuring an Intel Core i7-8665U @ 1.90GHz processor and16.0 GB of RAM (this data proves that this algorithm maintains low processing overhead and operates effectively in near real-time). NetLogo plays a crucial role in achieving robust simulation results, particularly in areas like agent-based modeling. It excels at simulating the behavior of individual agents and their interactions within a defined environment. This allows to model complex systems. NetLogo offers an intuitive graphical interface for real-time simulation observation, enabling the analysis of the system dynamics and make interactive adjustments during runtime for sensitivity analysis and hypothesis testing.

The studied case in this section is based on a population of 50 particles that must achieve a final optimal location by taking into account the number of used iterations. Noting that, each iteration represents a cycle of the proposed Algorithm 3.1. As shown in Figure 3.2, the initial state of the environment is a 2D grid of 100×100 of environmental positions, known as patches. Each patch is characterized by its cartesian coordinates as well as a fitness degree calculated according to equation (3.12). With the aim

of providing a visual approximation of the patches' fitness degree, the patch color provided by the NetLogo platform was used. Specifically, the brightness of the patch increases in connection with the increase of fitness degree, as shown in Figure 3.2. Knowing that, the agents' colors showed in Figure 3.2 are randomly chosen. During each iteration, each agent (particle) updates its cartesian coordinates (position) according to equation (3.10). In the initial case, the agents' positions are randomly initialized. Also, the fitness degrees of the patches are updated from a simulation episode to another. The constant parameters *c1* and *c2* are fixed at the beginning of each simulation episode. However, the parameters *r1* and *r2* are randomly updated before each iteration.

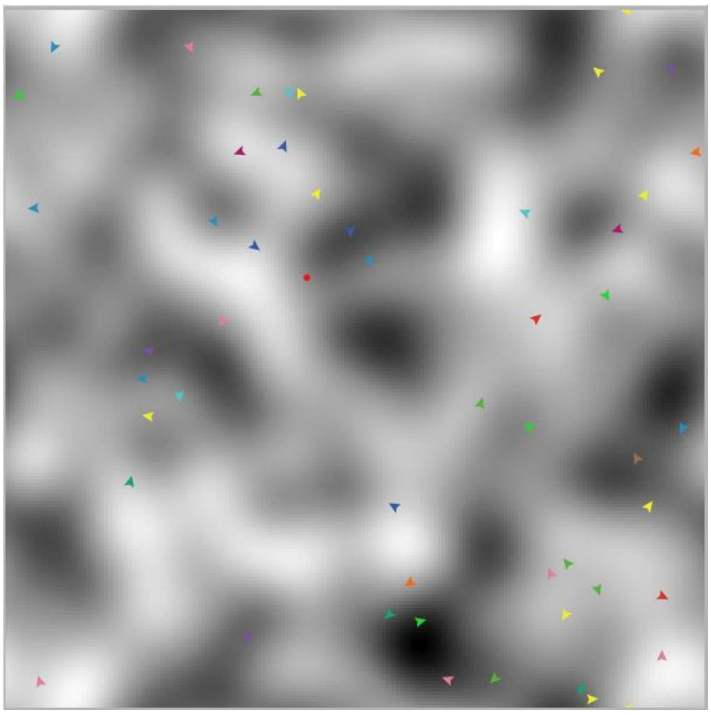

Figure 3.2: A representation of the environment

Knowing that the proposed PSO version is based on the improvement of the *IW* parameter updated before each iteration according to equation (3.13), the usefulness of comparing it with improved PSO approaches based on the same parameter has been seen. In other words, the proposed approach is compared with eight approaches based on different equations for calculating *IW* parameters. The principles of the 9 compared cases are well discussed in Sections 3 and 4 of this chapter:

- Constant Inertia Weight (CIW) [97,102]: in this case, the *IW* is during the different iterations (*IW =0.729*).
- Linearly Decreasing Inertia Weight (LDIW) [98]: In this case, the *IW* linearly reduces during iterations from 0.9 to 0.4 according to equation (3.2). LDIW prioritizes the search space exploration before the exploitation of the obtained solutions.

- Natural Exponent Strategy 1 Inertia Weight (NE1IW) [98]: In this case, the value of *IW* non-linearly decreases from 0.9 to 0.4 according to equation (3.3).
- Natural Exponent Strategy 2 Inertia Weight (NE2IW) [98]: this case has the same principle as the previous one (NE1IW), however, *IW* reduces according to equation (3.4) .
- Simulated Annealing Inertia Weight (SAIW) [98]: In this case, the *IW* decreases during the iterations according to equation (3.5).
- Butterworth Inertia Weight (BWIW) [102]: in this case, the *IW* parameter also decreases in a non-linear way according to equation (3.6).
- Standard Inertia Weight (SIW) [103,104]: this case is based on the original PSO version, in which the *IW* parameter is considered as a neutral element (*IW = 1*).
- Linearly Increasing Inertia Weight (LIIW) [99–101]: in this case, the *IW* parameter increases from 0.4 to 0.9 according to equation (3.7). LIIW aims to escape from local spaces in order to improve the search space exploration.
- Inversed Butterworth Inertia Weight (IBWIW): this case is based on the *IW* calculation proposed in this chapter according to equation (3.13).

Figure 3.3 represents the development of *IW* values during the execution of 100 iterations of the proposed algorithm by using the compared *IW* calculation approaches. As can be constated, the *IW* values remain constant in CIW (IW = 0.729) and SIW (IW = 1) cases. However, in LDIW, NE1IW, NE2IW, SAIW, and IBWIW, the *IW* values decrease from 0.9 to 0.4. In LAIW and IBWIW, the *IW* values increase from 0.4 to 0.9.

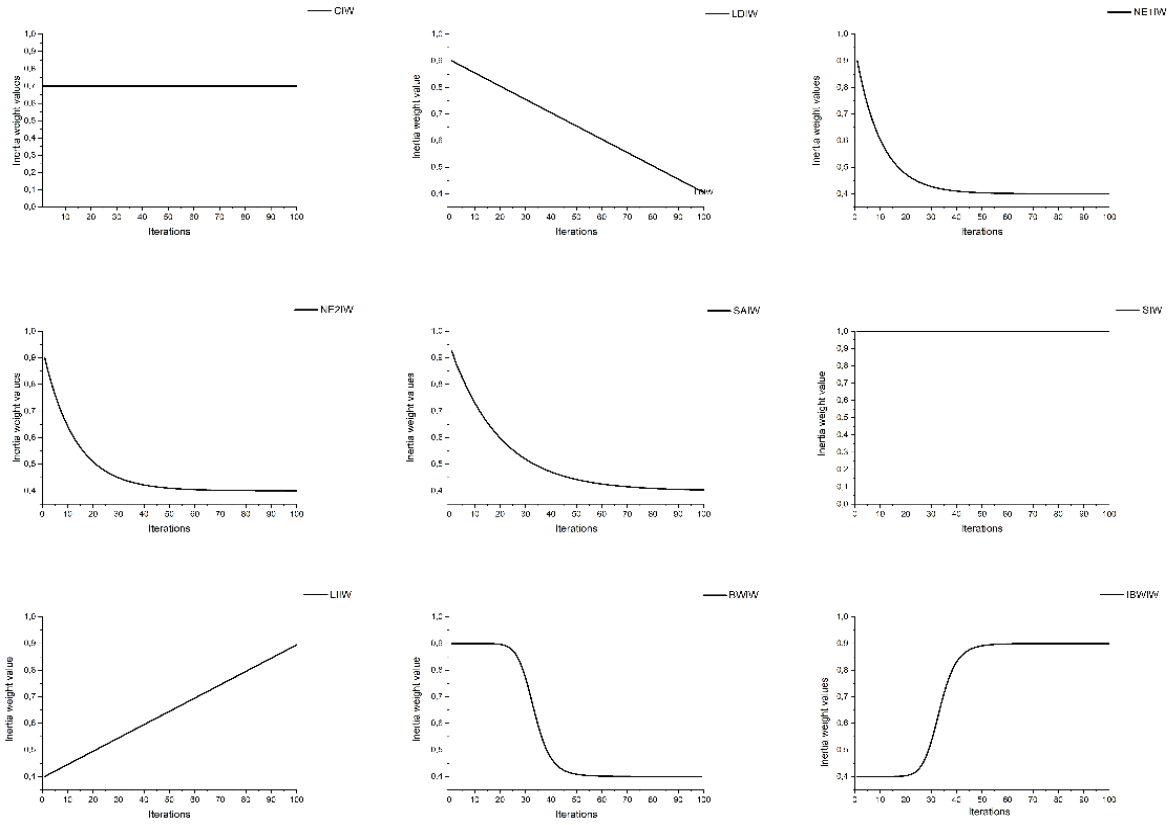

Figure 3.3: Different kinds of *IW* executed in 100 iterations

Figure 3.4 represents the searching time obtained by the execution of 100 episodes of the proposed algorithm through the use of the compared *IW* methods. The searching time is reflected by the number of iterations effectuated by the particles with the aim of finding the target position. According to the reflected results, it can be noted that the 8 algorithms have outperformed the SIW case by lower values in terms of searching time average. The highest average searching time is 92.6 iterations, which is obtained via the use of the SIW parameter. Furthermore, an important decrease of 44.51% can be noted in the average of BWIW (50.48 iterations) in relation to SIW case. In CIW, NE2IW, and SAIW cases the averages obtained were respectively 49.34, 49.05, and 49.47. In addition, an approximation between NE1IW 43.12 and LDIW 45.97 cases can be constated. According to the shown results, it is noted that the best average searching time 37.47 is obtained via the use of the proposed approach in this chapter (IBWIW). Also, an interesting average 38.79 is obtained through the utilization of the LLIW approach, which proves that the increasing *IW* methods have a positive impact on the MAS path planning problem.

Furthermore, a comprehensive analysis of the searching time obtained in 20 episodes is detailed in Figure 3.5. Knowing that in this case, each episode is composed of a maximum of 100 iterations. The searching time can be defined as the number of iterations generated by the particles with the aim of finding the goal position. In the case where the particles cannot reach the target at the end of an episode, the value 100 will be assigned as a result. The showcased results revealed that the SIW parameter significantly exhibits longer searching time in relation to the other studied cases, which is clearly represented by the bar chart in the figure. Across the majority of episodes, and in opposition to the other cases, the SIW based approach takes 100 iterations with an average of 91.65 without finding the target location. In the other hand, it is crucial to note that the IBWIW and LIIW take the shortest period with averages 37.47 and 38.79 respectively.

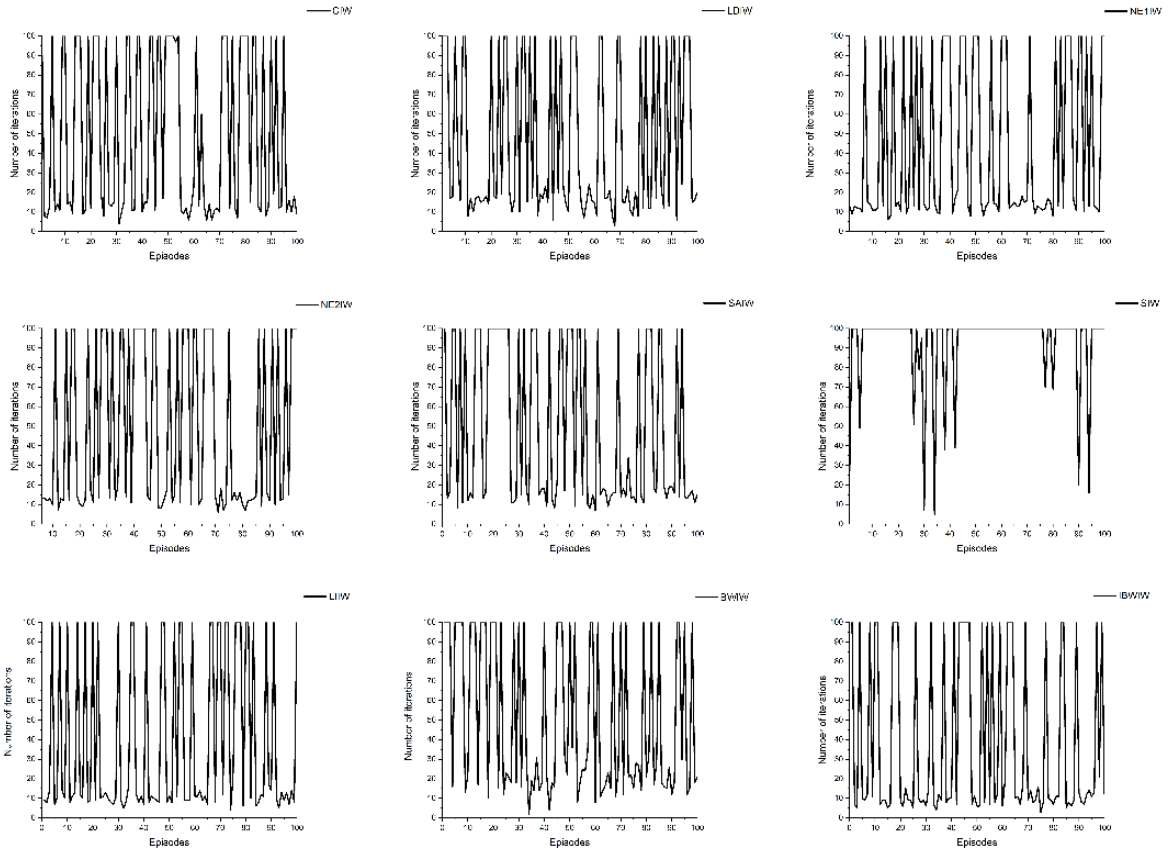

Figure 3.4: Average-searching time obtained (9 cases)

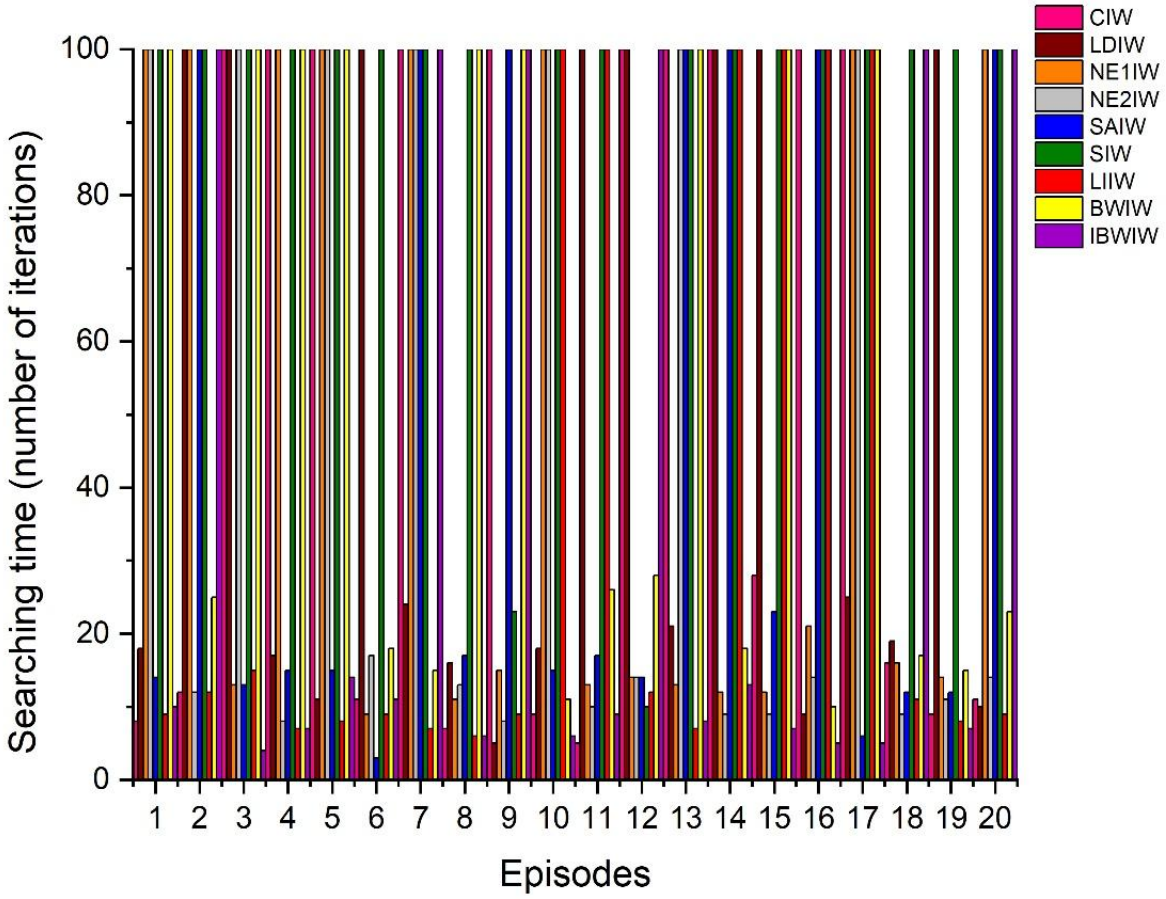

Figure 3.5: Average-searching time obtained in 20 episodes

Figure 3.6 represents the fitness development, which is indicated by the best value founded in a single episode. Each episode is composed of 100 iterations in order to find the desired location by utilizing the compared *IW* methods. Knowing that, the same agents' initial positions in the environment have been used as shown in Figure 3.8 in order to equitably compare the studied cases. Based on the obtained results, it is constated that the maximum fitness (f = 1) was obtained in the 9th iteration through the application of IBWIW parameter. In the LIW case, a fitness value of 0.95 was achieved after 13 iterations. In the CIW, SAIW, NE1IW and NE2IW cases, a fitness values between 0.82 and 0.92 were acquired in 10, 10, 19 and 13 iterations respectively. On the other hand, a fitness value of 0.75 was reached in the 41th iteration by the use of the SIW parameter. These results proved the goal orientation provided by the proposed *IW* calculation to MAS path planning problem. In other words, the proposed approach allows the agents to reach their objective position in minimum time in comparison with other improved PSO approaches.

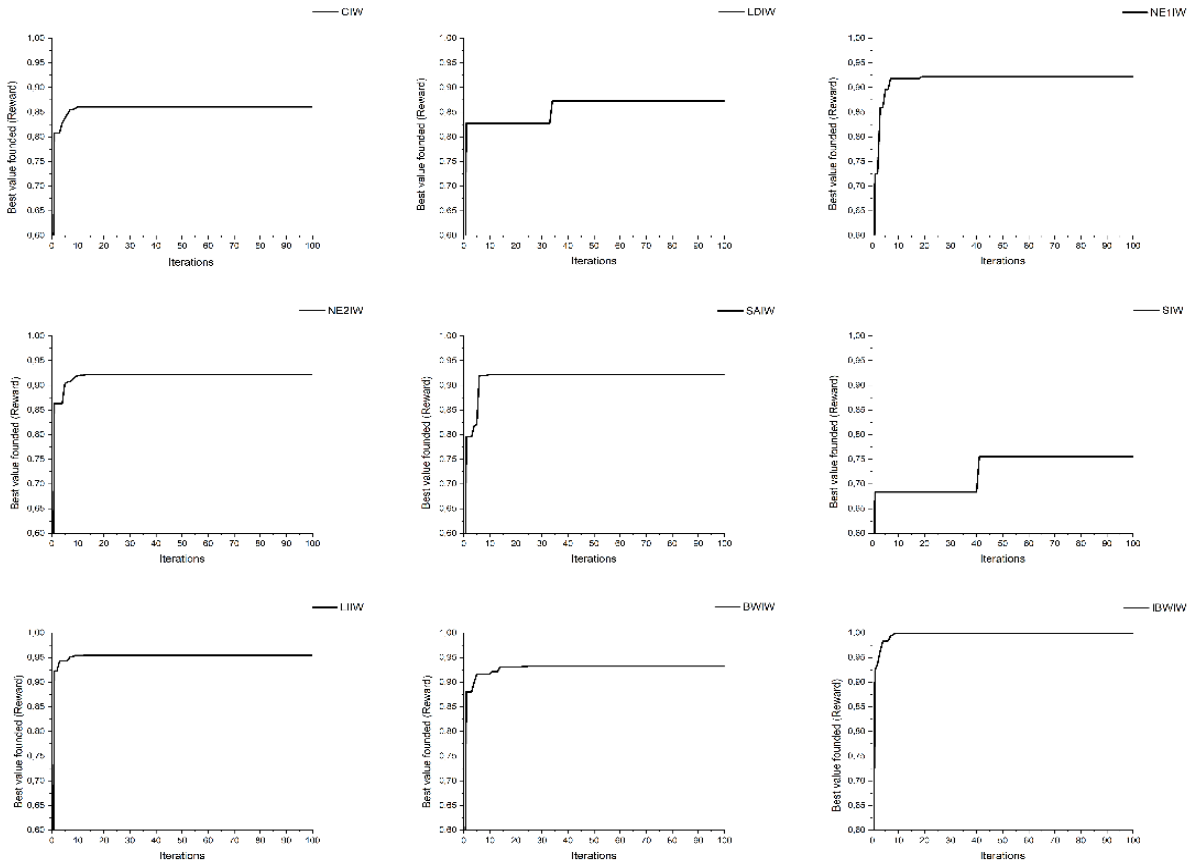

Figure 3.6: Fitness development during one episode of 100 iterations

Based on the results reflected in Figure 3.6, the usefulness of studying the difference between the fitness obtained in the *N+1* iteration and the *N* iteration has been seen during the complete episode, as shown in Figure 3.7. Generally, in the 8 compared cases, the difference increases in the first iterations. In particular, the highest difference that the IBWIW case produced is 0.9234 in the first iteration and 0.02 in the third iteration. In another studied case, the NE1IW produced a difference of 0.72 in the first iteration then 0.13 in the third one. Also, The SIW and the SAIW cases reflected close differences of 0.07 and 0.09, respectively in the 41st and 7th iterations. The highest differences obtained in the SIW and SAIW cases were in the first iteration (0.48 and 0.79 respectively). Furthermore, the differences obtained in the LDIW and the NE2IW methods were up to 0.82 and 0.86 in the first iteration. Also, these last two cases produced differences of 0.04 in 34th and 5th iterations. On the other hand, the differences that the BWIW, CIW, and LIW cases returned in first iteration were 0.88, 0.80 and 0.92, respectively. At a later time, they returned between the third and the fourth iterations difference values between 0.01 and 0.02.

In comparison with SIW case, all the *IW* based on dynamic calculations provided better results in terms of fitness development, especially the proposed IBWIW. Precisely, the obtained results proved the efficiency of the PSO based on dynamic *IW* methods regarding the convergence towards the optimal solution. Moreover, the results demonstrate that the proposed IBIW is the most efficient approach to resolve the MAS path-planning problem.

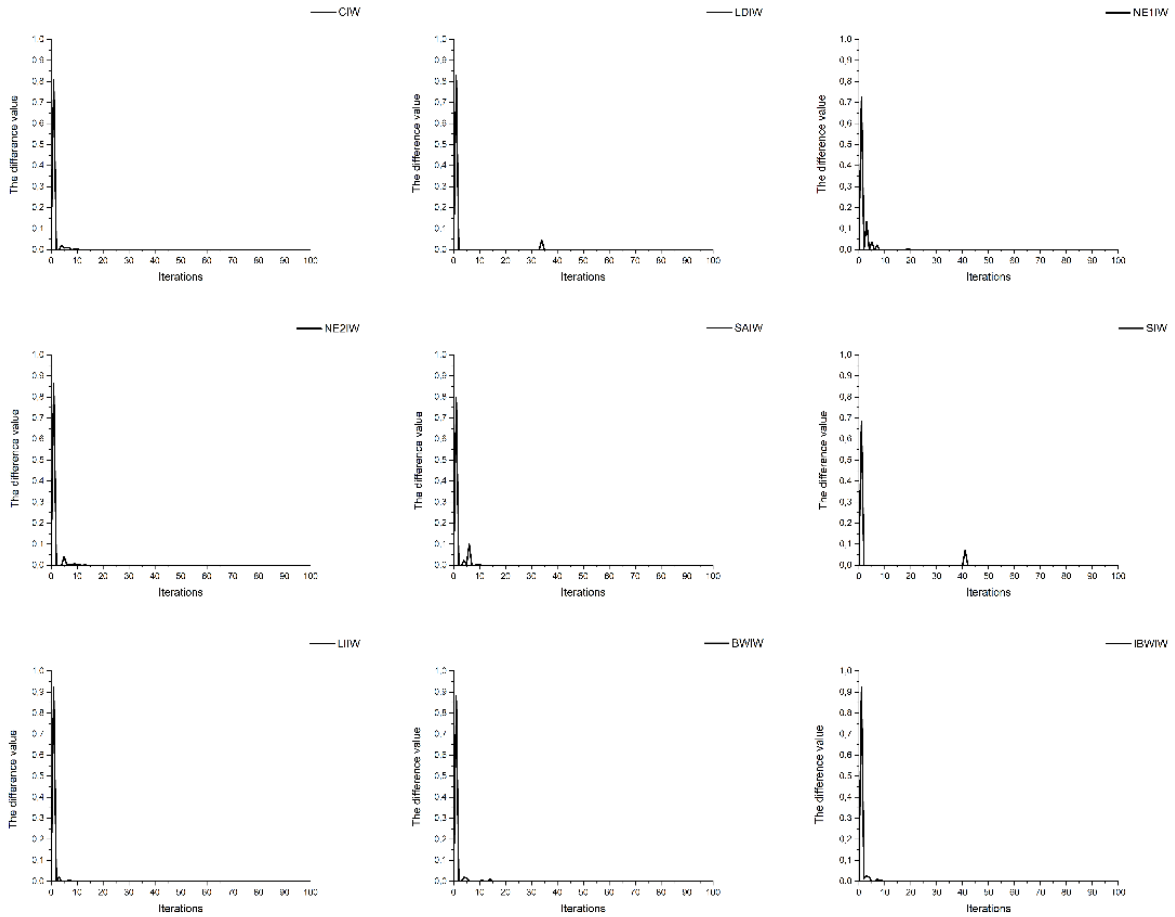

Figure 3.7: Difference between the N+1 iteration fitness and the N iteration fitness during one episode of 100 iterations

To verify the accuracy and reliability of the obtained results from the prior analysis, another simulation scenario is introduced. Figure 3.9 represents the searching time obtained through the application of the compared cases during 100 episodes as the first scenario. However, the same initial positions of particles are provided for all the *IW* variants (fixing the random initialization of positions) as shown in Figure 3.8. Moreover, it is noted that the remaining parameters are unchanged. From the reflected results, it can be constated that the dynamic *IW* based approaches achieved superior performances compared to

the SIW in terms of searching time's average. Specifically, an average searching time of 100 was obtained by the use of the SIW parameter. Otherwise, the searching time decreases in BWIW and NE2IW with the averages of 48.38 and 48.10 respectively. In addition, the average of NE1IW case (43.30) is approximate to the averages of LDIW (43.98), CIW (44.95), and SAIW (43.65). Subsequently, the IBWIW case reflects the lowest average searching time (32.14) in comparison with the other cases. In the same context, the LIIW returned an average approximately equal to the IBWIW with 34.38. According to the obtained results, it can be deduced that the increasing *IW* based methods are more efficient in the processing of the MAS path-planning problem.

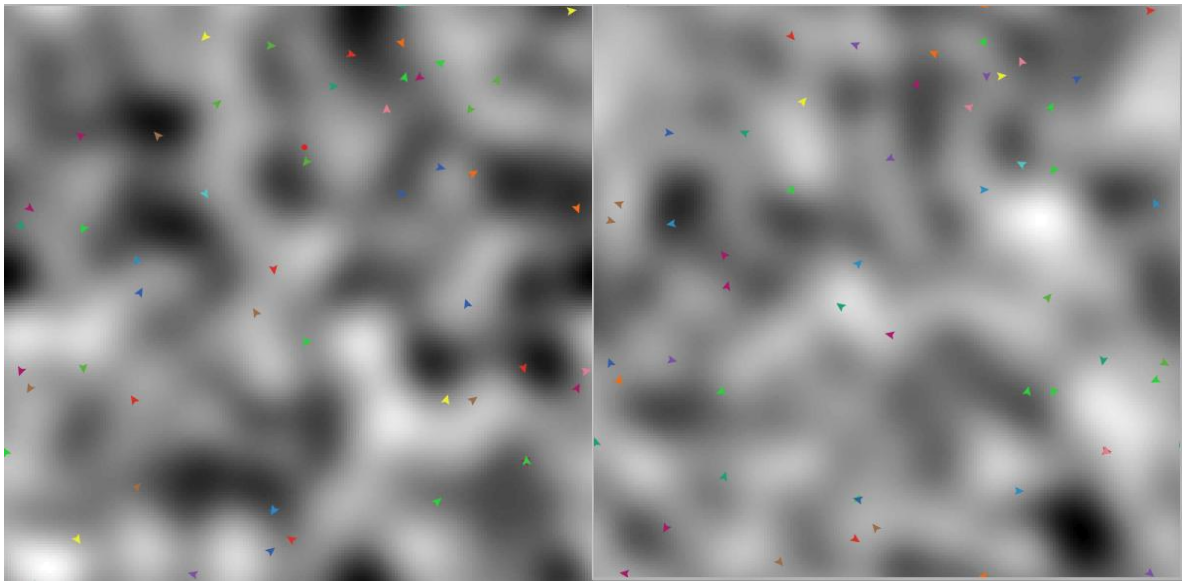

Figure 3.8: Example of two initializations with the same positions

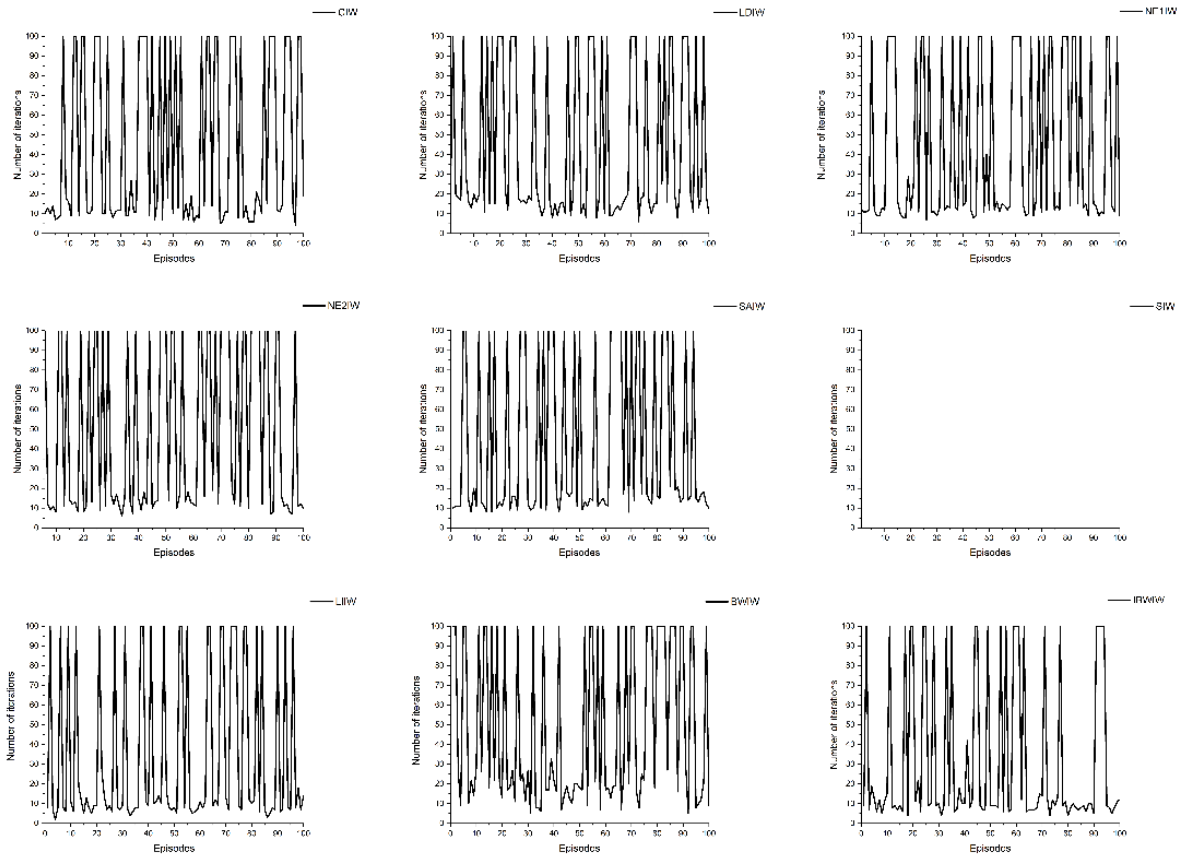

Figure 3.9: Average-searching time obtained by initializing agents with the same positions for the 9 cases

In this section, it was introduced a comparative analysis of the proposed approach (IBPSO) in comparison with recent PSO versions based on *IW* improvement. Through these simulations, it was studied the searching time obtained using the compared *IW* methods during 100 episodes by using random initial positions of the agents. Furthermore, it was studied the fitness development obtained in an episode of 100 iterations. From the obtained results, it was constated that all of the PSO improved versions, including the proposed one, outperformed the original PSO version. Moreover, it was noted that the increasing *IW* methods provide a significant improvement in path planning problems. Specifically, PSO versions based on increasing *IW* enable agents to achieve their objectives in minimum time and efficiently converge to their optimal solutions. Moreover, to validate the obtained results, it was studied the searching time through the use of the same the initial agents' positions. The results proved the effectiveness of the proposed approach as well as LIIW. A better understanding of the increasing *IW* role can be achieved by observing how solutions evolve during simulations. In other

words, increasing *IW* encourages initial exploitation of the best solutions found, followed by an exploration of the search space.

3.6 Summary

In this chapter, it was introduced a novel MAS path planning based on a modified PSO approach. As a metaheuristic approach, PSO allows the particles to move in the search space according to their personal best position as well as the global best position detected by the swarm. Moreover, PSO is based on the *IW* parameter, which allows to control the exploration and the exploitation of the search space. In the proposed PSO version, an increasing non-linearly *IW* control has been introduced, known as IBIW in order to favorize the exploitation of the found solutions at the first iterations, and progressively the exploration of the search space. To apply this version to the MAS path planning, each agent was considered a particle moving in the environment to find the location with the highest fitness value. To prove the feasibility of the proposed method, it has been compared to other PSO versions based on *IW* improvement. According to the showcased results, it has been constated that the IB-PSO allows to decrease the searching time and to increase the fitness acquisition in comparison with the other PSO versions. The key to this improvement lies in the IB-PSO's ability to facilitate a more aggressive exploration late in the optimization cycle, a mechanism critical for escaping local optima and ensuring convergence to a global solution. In future work, modification of the proposed approach will be attempted in order to be applicable in a 3D search space environment. This initiative will be taken with the aim of considering reel world application based on the use of UAVs. To propose a more robust solution, the next chapter will detail the implementation of an original algorithm, resulting from a hybridization of existing methods, with a particular focus on path planning with collision avoidance problem.

CHAPTER 4: MULTI-AGENT COLLABORATIVE PATH PLANNING BASED ON HYBRID SWARM INTELLIGENCE APPROACH

4.1 Introduction

The integration of autonomous agents in MAS into everyday environments faces significant challenges, particularly in path planning and collision avoidance. These challenges are critical for safe operations, especially in dynamic environments where agents must navigate obstacles including other agents. In safety-critical applications, errors can lead to equipment damage and risks to infrastructure. Robust collision avoidance is essential, shifting focus from merely optimizing travel time to safety mandates, thereby facilitating reliable MAS technology in real-world scenarios. Metaheuristics provide effective solutions for optimization challenges in MAS by harmonizing exploration and exploitation of search spaces and adapting to various problems through hybrid methodologies, leveraging strengths from multiple algorithms. From one perspective, the CSA, is a metaheuristic algorithm that simulates crow behavior where crows conserve excess food reserves for future retrieval. The CSA inherently demonstrates strong capabilities in exploration, derived from the way crows randomly search and revisit their memory of stored food, ensuring a wide coverage of the search space. However, from another perspective, PSO has become a common metaheuristic technique in the optimization community. The higher exploitation ability of PSO comes from the interaction of the particles influenced by both their own and global best positions. However, it is relatively weak in exploration. In addition, this algorithm may converge to local optima, and this is an important challenge.

Inspired by the earlier research activities, the main objective of this chapter is to propose a new hybrid CSA and PSO optimization-based path planning solution with collision avoidance known as CSO for crow swarm optimization. The aim of the suggested solution is to enhance optimization performance by combining the best features of both algorithms. The principal motivation for choosing these metaheuristics refers to the positive aspects of PSO in exploitation and that of CSA in exploration. Furthermore, it is imperative to address the adopted approach to avoid collisions. Therefore, this path-planning algorithm uses velocity regulation that implements velocity reduction when a collision is detected. In other words, the intellectual contribution of this work is the functionally segregated control mechanism, which addresses the core empirical challenge in maintaining the balance between search exploration and exploitation by utilizing a simple, empirically determined fixed transition factor (from the exploration phase to the exploitation), thereby reliably preventing premature convergence to suboptimal local minima. Consequently, to achieve this, the main goals of this chapter can be summarized as follows:

- The proposition of a MAS path-planning algorithm with collision avoidance, based on a hybridization of CSA-PSO algorithms.

- The testing of the impact of the α-constriction factor and the regulating velocity approach on ensuring effective safety while simultaneously achieving rapid convergence.
- The minimization of searching time and path length, allowing for fast convergence to optimal solutions.
- The escape from the trap of local minima, merging the exploration strengths of CSA with the exploitation capabilities of PSO. This allows enhancing its ability to traverse diverse solution spaces, overcoming local optima, and identifying global solutions, leading to improved optimization outcomes in complex problems.
- The enhancement of scalability via the minimization of the number of agents.
- The comparison of the proposed CSO method with recent approaches aiming to control the balance between exploration and exploitation through their application to the path-planning problem with collision avoidance.

The chapter is organized as follows: Section 1 introduces the core concepts and outlines the problem addressed by this research. Section 2 discusses recent investigations of MAS that utilize the CSA metaheuristic, focusing on hybridization, establishing the need for the novel approach. Section 3 provides descriptions of the CSA method, the PSO method, and the MAS path planning with the collision avoidance problem. Section 4 provides a detailed description of the proposed hybrid path-planning algorithm with collision avoidance. Section 5 conducts a comparative analysis of the primary simulation outcomes against related methodologies. Finally, the conclusion summarizes the main contributions of this chapter in Section 6.

4.2 Foundational Literature and Research Gap for Multi-agent systems path planning

This section aims to analyze prior research related to the work presented in this chapter, focusing particularly on the most significant and recent studies in MASs that utilize CSA and PSO metaheuristics. It leads directly to the necessity of the proposed approach.

In recent years, the CSA has garnered significant attention for its efficacy in solving various optimization problems. A study [108] aims to present a detailed overview of CSA, its fundamental principles, variations, and different applications in various fields. CSA effectively addresses path-planning issues in MAS. In addition, another work in [109] focuses on enhancing the CSA, particularly its exploration and exploitation capabilities, to address global numerical optimization challenges. Several enhanced versions of CSA have been developed to improve its performance, adaptability, and

convergence speed across different applications beyond the path-planning problem. These enhancements leverage various strategies, making the CSA a versatile tool to tackle increasingly complex optimization challenges. For example, the research [110] proposes a novel approach to data privacy preservation by utilizing a CSA enhanced with adaptive *AP*. Similarly, the study [111] introduces a modified version of the CSA specifically aimed at solving challenges within power systems, adjusting the *AP*. A key study in this area [112] introduces the Enhanced CSA (ECSA) based on Dynamic *AP*. It is specifically designed for feature selection in machine learning applications. These enhancements aim to improve the exploration capabilities of the algorithm. A conference paper [113] introduces enhancing CSA's capabilities. This enhancement improves the algorithm's ability to explore multiple optima within a complex search space. It modifies the traditional CSA, adjusting both the *AP* and the *fl* parameter. Another paper [114] focuses on optimizing fractional order controllers for Automatic Voltage Regulation (AVR) systems using a distance and Levy-Flight-based CSA. It combines traditional CSA with distance metrics and Levy-flight strategies to enhance the exploration capabilities of the algorithm. Similarly, another work [115] presents a Dynamic CSA (DCSA) that utilizes adaptive parameters to tackle large-scale global optimization challenges. By incorporating dynamic adaptations of its search parameters, the DCSA enhances its ability to explore complex solution spaces effectively and escape local optima. The study demonstrates the efficacy of their approach through extensive experiments.

The PSO has emerged as a powerful method in MAS for efficiently solving complex problems. Several studies, such as [116], it provide a detailed study of the advancements and applications of PSO methodology across various domains, highlighting its evolution and strengths. The study [92] presents a novel algorithm using discrete PSO to form coalitions among pursuers in dynamic multi-agent scenarios, enhancing their ability to capture evaders. In addition, recent developments in PSO have led to various improved versions. The paper [117] introduces the Inversed Butterworth PSO, which enhances the traditional PSO by drawing inspiration from Butterworth PSO [102] to improve convergence speed and path planning efficiency in MASs.

Moreover, it is imperative to recognize that use of metaheuristics extends far beyond foundational methods such as PSO and CSA. Consequently, to meet the demands for optimal, energy-efficient 3D path planning in complex environments, notably agricultural scenarios, this paper [118] showcases the necessity of advanced adaptive metaheuristics. Superior, real-world robotic performance is achieved through the utilization of techniques like the Incremental Grey Wolf Optimizer (I-GWO) and the Enhanced Grey Wolf Optimizer (Ex-GWO). Similarly, another paper [119] proposes novel algorithms

such as Multi-Domain Inversion-Based Genetic Algorithm (MDIGA) for Unmanned Surface Vehicle (USV) path-planning, which, by employing a strategy of increasing and filtering offspring, demonstrates a shorter optimal path, faster convergence speed, and superior robustness compared to conventional GAs. Additionally, research on UAV path planning [120] has focused on improving the Ant Colony Algorithm (ACA) to prevent it from falling into the local optimum by setting upper and lower limits for pheromones and their volatile factors. This enhanced ACA also incorporates multi-heuristic factors to further improve the algorithm's performance in finding optimal paths. The field of MAS collision avoidance has also seen substantial recent contributions from Machine Learning (ML), particularly through Deep Reinforcement Learning (DRL) framework [121].

Furthermore, the hybrid algorithms have emerged as a powerful strategy to further refine different optimization problems. By combining, for example, the strengths of CSA with techniques such as PSO [122] this study proposes a hybrid algorithm that integrates CSA with other optimization methods. It shows that the combination of PSO with CSA makes them complete each other. This approach aims to leverage the strengths of multiple optimization techniques to enhance performance. Moreover, it presents a novel approach to feature selection in machine learning by combining the strengths of PSO and CSA.

As deeper delving into the hybrid MAS path planning algorithms, it is notable how the modified and hybrid versions have been particularly effective. From these works, the research in [123] is introduced; it explores the application of CSO in determining optimal paths for robots in various environments, potentially enhancing their navigation capabilities. The comparison process illustrates that the CSA is better than the PSO and ACO in path planning, but compared to the hybrid method, CSO's performance improved with decreasing population size. In addition, the paper [124] provides the optimal path planning of the wheeled mobile robot with collision, using a hybridization of PSO and grey wolf algorithm (GWA) [125].

Consequently, machine learning and deep reinforcement learning (DRL) methods typically require extensive training data and sacrifice interpretability, justifying the use of a computationally metaheuristic that provides explicit safety constraints and guaranteed convergence without prior training. Moreover, single methods like PSO are too exploitative and suffer from problems, including local optima. This problem becomes especially noticeable when many agents are operating closely together (high-density MAS), potentially causing system-wide deadlocks or suboptimal path solutions. Conversely, CSA is too explorative, leading to impractically slow convergence. This high exploratory

tendency stems directly from the algorithm's core mechanics, which simulate the broad, cautious search behavior of crows. Furthermore, across all metaheuristic techniques and their improvements, including GA and ACA, the major research gap remains the tension between solution quality and computational efficiency when adapting from controlled simulations to dynamic and real-world scenarios. Additionally, existing hybrid methods often fail to fully integrate safety constraints with the optimization core. Consequently, the hybrid CSO is uniquely effective because it structurally combines the exploratory power of CSO to prevent the local stagnation common to PSO with the exploitative speed of PSO to overcome the slow convergence of pure CSA. The Hybrid CSO fills this gap by critically implementing the α-constriction factor, guaranteeing a superior balance of safe, rapid convergence under high pressure. Moreover, this CSO framework stems from the inherent limitations of existing metaheuristics in achieving safety and efficiency simultaneously.

The current state of path planning reveals significant limitations across various methodologies. Firstly, ML and DRL methods require extensive training data and sacrifice interpretability, justifying the use of a computationally lighter metaheuristic. Among single metaheuristics, PSO is too exploitative and suffers from problems, including local optima. This problem becomes especially noticeable when many agents are operating closely together (high-density MAS), potentially causing system-wide deadlocks or suboptimal path solutions. Conversely, CSA is too explorative, leading to impractically slow convergence. This high exploratory tendency stems directly from the algorithm's core mechanics, which simulate the broad, cautious search behavior of crows. Furthermore, across many metaheuristic improvements (including GA and ACA), the primary gap remains the tension between solution quality and computational efficiency when moving from simulations to dynamic, real-world scenarios. Additionally, existing hybrid methods often fail to fully integrate safety constraints with the core optimization. Consequently, the hybrid CSO is uniquely effective because it structurally combines the exploratory power of CSO to prevent the local stagnation common to PSO with its exploitative speed to overcome the slow convergence of pure CSA. The framework fills this gap by critically implementing the α-constriction factor, guaranteeing a superior balance of safety and rapid convergence under high pressure.

4.3 Problem description

This section outlines the principles versions of the CSA algorithm and its different equations, highlighting the most important parameters and spotlighting its strength in the exploration phase. Furthermore, the principles of the PSO method are elucidated, emphasizing its strength in exploiting

the environment. Moreover, it introduces the path-planning problem with collision avoidance, describing how agents avoid collisions.

4.3.1 The compared Crow Search algorithm versions

As the principles of the CSA are thoroughly discussed in Section 2 of chapter 2, there is no need to reiterate them here. The equations for the adaptation of a crow's position and memory are presented in equations (2.17) and (2.18).

In optimization using metaheuristics, it is essential to strike the right balance between exploration and exploitation [126]. In the early iterations, it is absolutely necessary to exploit the space; in contrast, during the last iterations, the focus shifts to exploitation. On this basis, and on the one hand, the exploration is vital to avoid local optima, as it enables the algorithm to discover globally optimal solutions. On the other hand, regarding the exploitation, the best resources need to be found. While the basic CSA provides the basic principles of crow behavior, it is important to highlight two key parameters that significantly influence the algorithm's performance by guaranteeing the balance between exploration and exploitation: these two key factors are *fl* and *AP* [126]. The intelligent adaptation of both parameters allows the algorithm to navigate complex search spaces more effectively. A longer *fl* allows crows to move from their current positions, enhancing their ability to encourage exploration of new areas. While high-*AP* leads to better exploitation. Indeed, from [127], the *fl* indicates the crow's maximum possible distance to steal food, with *AP* ranging from 0 to 1, with lower values indicating local searches. As a result of the work [110], using small values of *AP* intensifies the algorithm by increasing the *AP* value, the probability of searching the position of current better solutions decreases, and CSA tends to exploration.

To address the limitations of the basic CSA, it exists many improved versions that typically focuses on adjusting adaptive or dynamic *fl*, also *AP* parameters. Knowing that the main objective of this chapter is to propose a new improved hybrid CSA version, the usefulness of discussing the main improved calculation methods of both *fl* and *AP* of the literature review has been seen. In order to enhance the CSA, different extensions based on the *fl*, *AP* or both control were proposed:

- AAP-CSA: the crow *j* is chosen using equation 4.1 [110]:

$$APj = 1 - (iter/ itermax) \qquad (4.1)$$

Where: *iter* represents the current iteration; *itermax* refers to the maximum number of iterations.

- ICSA (improved CSA): it is based on Dynamic Awareness Probability (DAP); it is calculated by picking a fitness value of a random crow using equation 4.2 as follows [111]:

$$DAP = 0.9 * (F(X_{i,k}) / wV) + 0.1 \qquad (4.2)$$

Where: *wV* represents the worst fitness value seen sofar. Assuming a minimization problem, this value is calculated as follows in equation 4.3:

$$wV = max\ F(X_{i,k}) \qquad (4.3)$$

Under this probabilistic approach, promising solutions will have a high probability of being exploited.

- ECSA: an enhanced CSA, it is based on Dynamic Awareness Probability (DAP), which is based on fitness values. It is calculated using equation 4.4 [112]:

$$DAPi = APmin + (APmax - APmin)\ ranki/NP \qquad (4.4)$$

Where: *ranki* is the rank of the ith solution according to its fitness; *NP* refers to the size of the population. *APmin* and *APmax* are the minimum and maximum values of *AP*. In this work, *APmin* was experimentally set to 0.1 and *APmax* to 0.8.

- MCSA: a modified CSA. In the primary CSA, *fl* and *AP* remain constant throughout the whole search method. This is not suitable for creating a balance between exploration and exploitation. So the following equations (4.5 and 4.6) are proposed adjusting both *fl* and *AP* [113]:

$$fl = 2 * \frac{tmax - titr}{tmax} + 0.5 \qquad (4.5)$$

$$AP = 1 - \frac{tmax - titr}{tmax} \qquad (4.6)$$

Here, *tmax* denotes the maximum iteration number, and *titr* is the current iteration number. At the initial iteration, *fl* and *AP* start with 2.5 and 0, respectively.

- INC-CSA: by increasing the value of *AP*, CSA tends to conduct the search for finding a global optimum solution, while decreasing its value tends to find a local optimum solution. As a result, *AP* is calculated using equation 4.7 as follows [114]:

$$AP = \begin{cases} APmax \times \frac{t}{t\ tmax} & for\ (0 < AP < 2) \\ APmax \times ri & otherwise \end{cases} \qquad (4.7)$$

Where: *APmax = 0.9; tmax* is the maximum number of iterations; *ri* is a random number between 0 and 1; *t* represents the current iteration.

-DEC-DAPCSA: for decreasing dynamic awareness probability, it will be decreased linearly from *APmax* to *APmin* over iterations. High *AP* values stimulate randomization in the algorithm search

process, directing the global optimal solution. Low *AP* values promote the CSA exploitation phase to extract the best results. *AP* is calculated using the following equation 4.8 [115]:

$$APiter = \left(\frac{APmin - Apmax}{Iter-max}\right) \times iter + APmax \qquad (4.8)$$

Where: *Iter-max* refers to the maximum number of iterations; *iter* refers to the current iteration.

Noting that *APmax* and *APmin* are set experimentally (0.9 and 0.1, respectively).

Furthermore, it is essential to mention that several studies have demonstrated the strength of this algorithm in exploration. This is related to the integration of a crow-inspired food hiding mechanism and the randomization. The CSA method ensures an initially uniform population distribution across the search space [122].

4.3.2 The compared Particle Swarm Optimization versions

As the principles of PSO are well-established, they will not be redescribed here. A detailed description of the velocity and position equations is presented in equations (2.1) and (2.2), which are described in Section 2 of chapter 2 of this work.

As said previously, in the early version of PSO, the *IW* parameter is fixed to a constant value (typically 1 or 0.7). Moreover, the constant value proves insufficient to effectively balance exploration and exploitation in the PSO algorithm. This suggests the need to explore more alternative *IW* values. Knowing that, a higher value of *IW* leads to exploration. Conversely, a low value leads to exploitation. There is extensive literature on the objective of enhancing the PSO method by adjusting the *IW* parameter. In this work, two recent approaches for solving the path-planning problem are highlighted, which are the Butterworth IW [102] and the inversed Butterworth IW, a method proposed in Chapter 2 [117].

- Butterworth IW: this non-linearly decreasing *IW* version is calculated using equation 3.6 [102] in Chapter 3.

-Inversed Butterworth IW: this non-linearly increasing *IW* version calculated using equation 3.13 [117] in Chapter 3.

On the other hand, in the literature, many works concentrate on the exploration capability of PSO. The paper is a refined version of PSO that enhances the exploration capabilities while ensuring convergence to optimal solutions [128]. Another paper in [129] shows the exploitation capability of PSO against its exploration capability. This reinforces the intention to leverage this feature in this work.

Moreover, the proposed hybrid algorithm in this work involves both CSA and PSO. Consequently, it is necessary to incorporate certain improved versions from both the first and the second approaches.

4.3.3 The path-planning problem with collision avoidance

As discussed in Chapter 1, path planning [117] determines the best path for a moving object that can execute to traverse safely and without collision [130] between two known points. Solving this problem helps develop intelligent agents that can perform tasks autonomously in situations where human interaction is not possible or limited. In path planning, the mobile agent takes the optimal path based on the available knowledge regarding the environment in the scene of interest. Research on path planning and collision avoidance (COLAV) algorithms improves safety and reduces accidents, fuel costs, and operational costs. Moreover, collisions can occur in moving obstacles like other agents or static obstacle. When collisions are detected, subsequent particle action is crucial. Firstly, the system needs to detect if a particle's movement results in a collision. In this investigation, and to detect collisions, it is necessary to use communication firstly. The communication in MAS refers to how artificial agents interact and coordinate their actions in order to take decisions. Which allows direct or indirect information exchange.

One approach to handle collisions is regulating velocity [131], by multiplying a factor to the particle's velocity. This velocity regulation aims to achieve a balance between exploration and exploitation. In this case, the velocity-regulated method is used to avoid collision by limiting the speed of the agents or particles in case of detection of a collision. The most common method to regulate the velocity is the constriction factor [131]. It is reassessing surroundings post-collision to facilitate effective path planning. Moreover, other work enhances optimization performance by dynamically adjusting the search space and imposing velocity limits in [132].

Upon detecting a collision, the particle's i velocity, represented as vi, is adjusted using a constriction factor α based on equation 4.9 [131] as follows:

$$vi(t+1) = \alpha \times vi(t) \qquad (4.9)$$

Where: $vi(t+1)$ is the new velocity after collision detection; α is a constriction factor, effectively reducing the speed of the particle; in this case, it is set to a constant float value between *0 and 1*.

4.4 The proposed algorithm for Optimal Multi-Agent Path planning with Collision Avoidance

This section introduces the proposed MAS path planning with collision avoidance based on the new CSO version. Specifically, it details how this approach is used to optimize the agents' convergence to their objective.

Algorithm 4.1: A hybrid CSO algorithm for multi-agent path planning with collision avoidance

```
Inputs: Found-Target = false;
setup-environment;
Initialize (population-size, target-position, number-iteration-max, c1, r1, c2, r2, fl, AP); //
AP fixed at 0.1 and fl fixed at 0.5

Foreach Agent_i do
   Initialize (position_i,  velocity_i, memory_i );
   memory_i ← position_i;
End for;
number-iteration ← 0;
searching-time ← 0;
target-value ← Evaluate-fitness(target-position_i);
Exploitation-Phase ← False; // Control flag for phase transition
Initialize(Change-Phase-Factor, α); // α  is a random float number between 0 and 1
                    //PHASE 1: EXPLORATION (CSA-BASED)
while (number-iteration ≠ number-iteration-max or  Found-Target ≠ true) and
(Exploitation-Phase = False) do
        Write ''Exploration phase using CSA''; //CSA phase
        Foreach Agent_i do
            fitness-value_i ← Evaluate-fitness(position_i); //knowing the current
numerical quality score of the current position of the agent i
            If fitness-value_i = target-value then
               Found-Target ← true;
            Elseif fitness-value_i is better than memory_i then
               Update memory_i;
               Gbest ← Max (memory); //global target that all agents will follow
during the subsequent Exploitation Phase
            End if;
            If target-value_i = false  and number-iteration ≠ number-iteration-max then
                Generate (r); //random number between 0 and 1
                Communicate-with-environment(agent_i);
                Pick-random (crow_j);
                Calculate(position_j);
                 // CSA Movement Logic
                 //Collision avoidance covers both the Seeking and the Tracking
state
                 If r < AP then // Seeking Mode
                    If Detect-Collision = True then
                       Update-velocity(α, velocity_i); // α-constriction for collision
avoidance
                       Position_i ← Move- randomly(memory_i , fl, velocity_i);
                    Else
```

```
                        Position_i ← Move- randomly(memory_i , fl, velocity_i);
                    End if;
                Else // Tracking Mode
                    If Detect-Collision = True then
                        Update-velocity(α, velocity_i); // α-constriction for collision
avoidance
                        Position_i ← Move- toward(memory_j; fl, velocity_i);
                    Else
                        Position_i ← Move- toward(memory_j; fl, velocity_i);
                    End if;
                End if;
            End if;
        End for
        searching-time ← searching-time + 1;
        number-iteration ← number-iteration + 1;
        If number-iteration = ( number-iteration-max  DIV Change-Phase-Factor) then
          Exploitation-Phase = True;
//it is fixed at 4 to ensure the transition to the Exploitation Phase is strategically
triggered exactly at the first quarter of the total iterations
          Write "passing to Exploitation phase''; //PSO phase
        End if
end while;          //ENDING THE PHASE 1: EXPLORATION (CSA-BASED)
                    //PHASE 2: EXPLOITATION (PSO-BASED)
                //INITIALIZE PSO ELEMENTS FOR EXPLOITATION
Gbest ← Max (memory); // Global Best determined from exploration results
fl← 0;
AP← 0;
Initialize(IW); // IW fixed at 0.4
Foreach Agent_i do
velocity_i ← velocity_i
Pbest_i ← memory_i; // PBest inherited from Exploration memory
End for;
    while number-iteration ≠ number-iteration-max or  Found-Target ≠ true do
        Foreach Agent_i do // Update PBest and GBest
            fitness-value_i ← Evaluate-fitness(position_i);
            PBest-value_i ← Evaluate-fitness(Pbest_i);
            Communicate-with-the-environment(Agent_i);
            If fitness-value_i = target-value then
              Found-Target ← true;
            Elseif fitness-value_i is better than PBest-value_i then
              Update PBest_i;
              Gbest ← Max (Pbest);
            End if;
            If target-value_i = false  and number-iteration ≠ number-iteration-max then
              If Detect-Collision = True then // PSO Movement Logic (Always
using update velocity method in collision case)
                Update-velocity(IW, position_i);
                Update-velocity(α, velocity_i); // α-constriction for collision
avoidance
```

```
                    Position_i ← Move- toward(Agent_i , GBest);
                Else
                    Update-velocity(IW, position_i);
                    Position_i ← Move- toward(Agent_i , GBest);
                End if;
            End if;
        End for;
        searching-time ← searching-time + 1;
        number-iteration ← number-iteration + 1;
    end while;
Outputs: searching-time, Found-Target;
```

The steps of the algorithm are explained as follows:

The principal objective of this algorithm is to compute the trajectory of each agent. The computation's duration, or search time, is reflected in the number of iterations per episode of execution. This process enables the identification of the optimal location, characterized by the highest fitness value, within the solution space. This location is generated to be a unique global optimum. In the context of this MAS path planning problem, the Objective Function defines the primary optimization goal: to maximize the predefined static value assigned to a location, formally stated as $MaximizeV\,alue(x)$, where x represents the target coordinate set (x, y). Since the problem is inherently structured as a maximization search for the highest possible value (between 0 and 1), the Fitness Function ($F(xi)$) used by the CSO algorithm is mathematically identical to the objective Function. The fitness is therefore calculated simply as $F(xi)=Value(xi)$, where xi is the agent's current location, and $Value(xi)$ is the fixed, predefined fitness score at that coordinate.

It is crucial to note that the agents must traverse the environment without encountering each other. At the beginning of the proposed CSO algorithm for MAS path planning with collision avoidance, the agents are randomly generated and positioned in the environment. In alignment with the core principles of both CSA and PSO, each agent assumes the functional role of a crow firstly. Subsequently, it transitioned into a particle.

For a more comprehensive explanation, initially, each agent i (where $i \in [1, N]$, where N refers to the population size, or the number of crows/particles), characterized by its initial position that gives it the first fitness value represented in its own memory at the beginning. In addition, the velocities of all entities are initialized as a preliminary step. In this case, it can be initialized with a simple constant value. Each agent's environmental coordinates are updated using its velocity, based on equation (4.10) as follows:

$$(x_{i,iter+1}, y_{i,iter+1}) = \begin{cases} x_{i,iter} + r_i \times fl_{i,iter} \times (m_{j,iter} - x_{i,iter}), \\ y_{i,iter} + r_i \times fl_{i,iter} \times (m_{j,iter} - y_{i,iter}), & r_j \geq AP_{j,iter} \\ A\ random\ position & otherwise \end{cases} \quad (4.10)$$

Where: *(x, y)* : the Cartesian coordinates of the concerned agent.

It is significant that the parameters *AP* and *fl* are initialized in this case as constant values in the beginning. However, this only happens if the agent does not collide with another one in the search area. Knowing that, each agent has a collision detection field. Every location within this field is characterized by the agent's signature, which is imprinted upon arrival and subsequently removed after displacement. The field consists of the agent's location and their 8 neighboring cells' locations. This method aims to communicate indirectly via the environment. Moreover, the signature of the agent is marked using equation 4.11. Then, the neighboring cells are marked using equation 4.12. Lastly, the collision based on both equations 4.11 and 4.12 is established using equation 4.13. Furthermore, Figure 4.1 shows a collision detection field of an agent in the environment.

$$Signature\text{-}t_i(x,y) = \begin{cases} -1 \ for\ (x = xt_i)\ and\ (y = yt_i) \\ 0\ Otherwise \end{cases} \quad (4.11)$$

Where: *Signature-t_i(x,y)* is the signature marked by the agent *i* (xt_i, yt_i) at the iteration *t* on the location with *x* and *y* Cartesians coordinates.

$$Neighbors\text{-}t_i(xt_i, yt_i) = \begin{cases} -1 \ for\ Signature - t_i(xt_i, yt_i) = -1 \\ 0\ Otherwise \end{cases} \quad (4.12)$$

Where: *Neighbors-t_i(xt_i, yt_i)* is the signature marked by the agent *i* on the 8 neighboring cells of agent *i* location with Cartesians coordinates xt_i and yt_i.

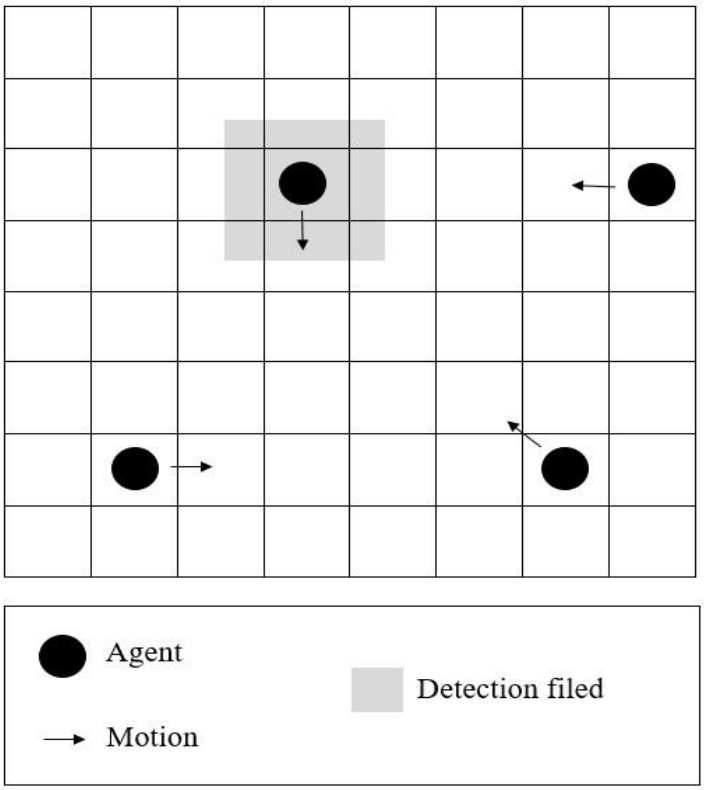

Figure 4.1: Collision detection filed of an agent in the environment

$$Collision(A_i t+1) = \begin{cases} yes \ \ for\ Signature - t_i(xt+1_i, yt+1_i) = -1 \\ No\ Otherwise \end{cases} \quad (4.13)$$

Where: A_i is the agent i; $Collision(A_i t+1)$ is the collision detected for the next position calculated at time $t+1$; $xt+1_i$ *and* $yt+1_i$ are Cartesians coordinates of the calculated position at time $t+1$.

When a collision is detected for the agent, the process uses equation (4.9) to decrease the $velocity_i$. Given that the parameter α is initialized with a random value between 0 and 1 at the beginning. Then it updates the $position_i$, consistently using equation (4.10). Moreover, in both cases, after the update of the agents' positions, the memory of each agent will be updated. To perform this action, each agent compares the fitness value of its current position with the fitness value of its memory according to equation (2.18) of the second chapter. Knowing that, the initial position of each agent is considered its initial memory position. Furthermore, it must update the *gBest* position, taking the position of the maximum fitness value of all memories. This is the process of exploration in this proposed approach.

To proceed with the exploitation process, a condition must be verified. This condition allows the agents, acting as crows, to distribute themselves throughout the environment. Precisely, the exploitation phase starts when the number of iterations becomes equal to the maximum number of iterations divided by a

factor (in this case, it is 4, because, experimentally, a quarter of the maximum number of iterations is adequate for the agents to achieve optimal distribution throughout the environment). This factor is a constant natural value initialized at the beginning. Considering that, the exploitation process means the transition from the CSA method to the PSO one. Specifically, transitioning from agents acting as crows to agents acting as particles. To start with, the proposed algorithm takes the position of the maximum value of memory given by the exploration phase as the initial *gBest*. In addition, it affects each agent's *i* velocity with the last velocity obtained in the exploration process. Moreover, the *pBest* position is reflected initially by the best position in the memory of the agent when it was crow. In other words, each particle (agent) iteratively updates its position based on its Cartesian coordinates *x* and *y* and according to the PSO principles, using its velocity calculated based on the detailed equations (3.9) and (3.10) of the previous chapter.

The agent's movement to its new position is governed by its velocity magnitude calculated in both exploration and exploitation phases using this equation (3.11) detailed in the chapter 3.

Moreover, it is noted that the constant parameters *IW* (0.4), c1, and c2 used for calculating velocity are initialized at the beginning. However, the dynamic parameters r1 and r2 are updated during each iteration. Additionally, in the case of a collision, the velocity calculated will be multiplied using equation (4.9), if not, the process works normally. Once the agents' positions are updated, the *pBest* position of each agent is then adjusted. This adjustment is performed by each agent, which evaluates the fitness of its current position against the fitness of its existing *pBest* position. Subsequently, and iteratively, the *gBest* position is revised. This revision involves selecting the *pBest* position that exhibits the greatest fitness value across the entire swarm.

The algorithm's execution continues until one of two stopping conditions is satisfied: either the maximum number of iterations is reached, or the optimal fitness position is identified. Upon completion, the algorithm provides the number of iterations performed and a Boolean indicating whether the optimal fitness position was found. It uses the equation (3.16) detailed in the chapter 3.

$$found\text{-}target = \begin{cases} True \ \boldsymbol{if} \ GBest - value = 1 \\ false \ \boldsymbol{else} \end{cases} \quad (3.16)$$

4.5 Simulation results

This section presents the simulation results obtained from the proposed algorithm, which was conducted to evaluate the performance of the algorithm, comparing the proposed path planning method with collision avoidance with other approaches based on both CSA and PSO principles. Consistent with the simulations detailed in Chapter 3 (Section 5), the current simulations employ NetLogo platform [107], version 5.3.1, utilizing its robust capabilities for agent-based modeling

To execute the various scenarios examined in this proposed method, each crow/particle was represented as an autonomous agent within this simulation environment. Moreover, each iteration represents a cycle of the proposed Algorithm 1. In addition, each episode is represented by the number of times the algorithm is executed. The static environment is initialized as a 100x100 2D grid of patches; each one has Cartesian coordinates, a fitness value, and a signature value that indicates the presence of a moving agent on that patch. It can be 0 if the patch contains no agents; otherwise, it can be -1. The target position has 1 as a fitness value. It is a unique value in the entire environment. To implement a visual approximation of the degree of fitness of the patches, this work used the native color palette of the NetLogo platform. An increase in fitness degree was represented by a corresponding increase in patch brightness. All agents are treated as point endowed with a defined signature (representing their presence) and a detection field. This detection field is specifically modeled to cover the 8-cell neighborhood surrounding the agent's current position, ensuring immediate threat detection for local path planning. Also, agents are initialized with a random velocity vector to ensure a non-deterministic start to the search process. Each agent is assigned a starting position and fitness value related to its position. This section examines a population of 50 agents aiming to optimize their location, taking into account the number of iterations. Noting that each agent must detect and avoid collision with the others. In each iteration, the position of each agent, represented by Cartesian coordinates, is updated according to the principle of CSA, then PSO using Algorithm 1, taking into account a condition of avoiding collisions. Initially, agent positions are randomly assigned. Furthermore, the fitness degrees of the patches are dynamically updated between simulation episodes. The following Figure 4.2 shows an initial state of the environment, noting that each agent is represented with a circle with a random color. However, the constant parameters *c1, c2*, and α are fixed at the beginning of each episode. In addition, the parameters *r1* and *r2* are initially generated randomly.

Furthermore, the selection of the parameters *AP*, *fl*, and *IW* is detailed in the proposed algorithm section. Finally, the transition from the CSA to the PSO method will be effected after a quarter of the maximum number of iterations. The main safety system works by doing a detection field check to spot any collision risk at the agent's next position. When a risk is found (for example, agent A2 detects A1, as shown in

Figure 4.3), this immediately turns on the α-constriction factor. This factor smoothly and dynamically reduces the agent's speed in the direction of the conflict, forcing it to slow down and adjust its path. This is why the velocity of the A2 line gets shorter. Paths created using this controlled method are much smoother and more efficient than other methods, which may take longer and result in suboptimal paths or just stop the agent completely (set velocity to zero), which is an inefficiency often seen in Figure 4.14. Such hard stops waste time and result in poor path quality; conversely, the α-method keeps the agents moving safely forward.

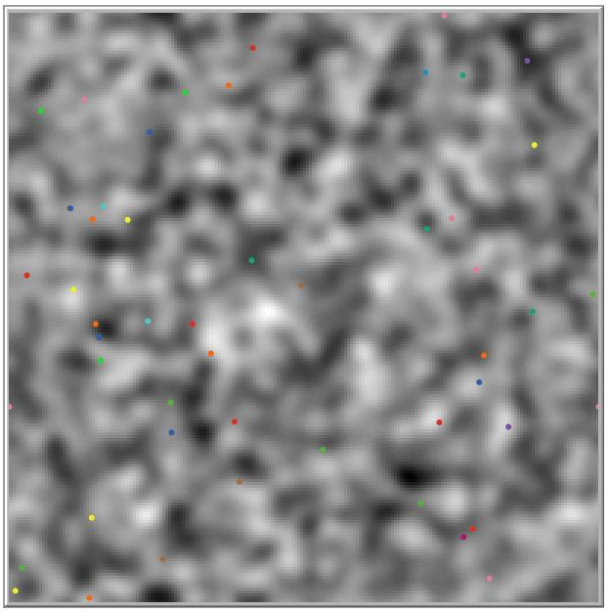

Figure 4.2: Initial state of the environment

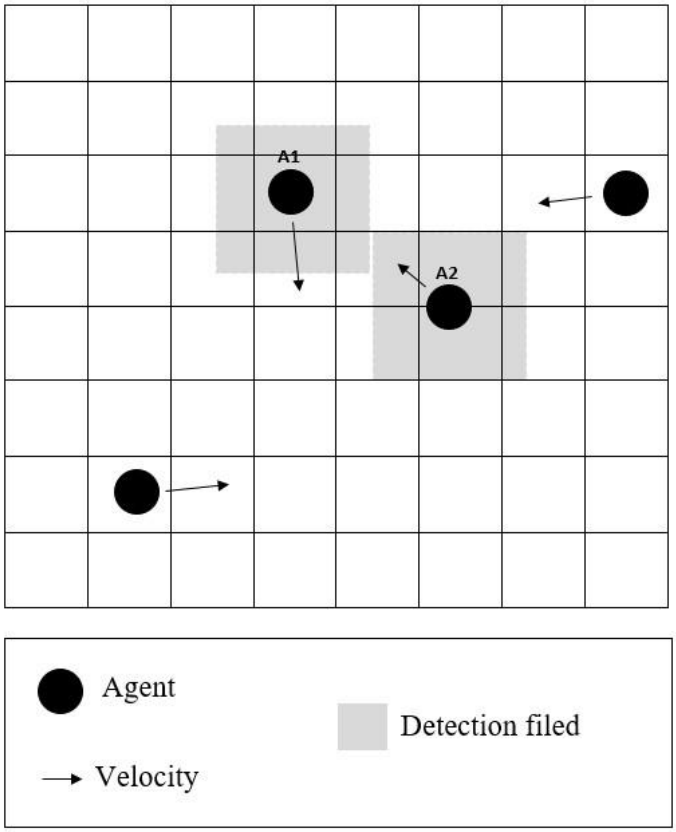

Figure 4.3: Collision avoiding case

Knowing that the proposed CSO version is a hybrid approach that controls the balance between exploration and exploitation. It will be compared with improved versions of both CSA and PSO in terms of efficiency and searching time. Noting that the proposed hybrid version is considered an improved version of the CSA's exploitation. In this regard, many different versions of improved CSA are chosen as compared to cases. They are based on the adjustment of *AP* or *fl* parameters. Besides this, two other improved versions of PSO are chosen based on the adjustment of the *IW* parameter. With the aim of providing an equitable comparison of the two algorithms' capabilities. Remembering that the *AP*, *fl* in CSA and *IW* in PSO, are parameters that can strike the balance between exploration and exploitation phases. Furthermore, the principles of the compared cases are well discussed in Sections 3 and 4 of this chapter.

- Adaptive Awareness Probability-based CSA (AAP-CSA) [110]**:** in this case, the *AP* is changing during the different iterations using equation (4.1) (decrease).
- Improved CSA (ICSA) [111]: in this case, the Dynamic the *AP* is changing during the different iterations using equation (4.2).

- Enhanced CSA (ECSA) [112] **:** in this case, the Dynamic the *AP* which is based on fitness values, is calculated using equation (4.4). The *AP* is changing during the different iterations between 0.2 and 0.8 (fixed experimentally)
- Modified CSA (MCSA) [113]: in this case, *fl* and *AP* are adjusted using equations (4.5 and 4.6). The *AP* value starts from 0 and increase to 1. The *fl* value starts from 2.5 and decreases.
- Increasing CSA (INC-CSA) [114]: in this case, the value of *AP* is increased from 0.01 (experimentally) to 0.9 using equation (4.7).
- Decreasing dynamic awareness probability CSA (DEC-DAPCSA) [115]**:** for decreasing dynamic awareness probability, it will be decreased linearly from 0.1 to 0.9 over iterations using equation (4.8).
- Butterworth Inertia Weight PSO (BWIWPSO) [102]: in this case, the *IW* parameter decreases in a non-linear way according to equation (3.6).
- Inversed Butterworth Inertia Weight PSO (IBWIWPSO) [117]: this case is based on the *IW* increases in a non-linear way according to equation (3.13).
- Crow Swarm Optimization (CSO): this case is based on the hybrid proposed version in this chapter according to Algorithm (4.1).

Figure 4.4 represents the development of *AP* and *fl* values during the execution of 100 iterations of each compared CSA approach. As can seen;

- The *fl* values remain constant in all cases *(fl = 0.5)*, except in the MCSA case, they decrease from 2.5 to 0.5. However, it is different for *AP* values that change in all cases.
- In AAP-CSA, DEC-DAPCSA, the *AP* values decrease linearly from 1 to 0 and from 0.9 to 0.1, respectively.
- In the cases of INC-CSA and MCSA, the values of *AP* parameters increase from 0 to 0.99 and 0.9, respectively.
- In the ICSA and ECSA cases, it can be seen that the *AP* values are fluctuating between increases and decreases. This is found logical, due to the reliance on the fitness value in the calculation equations. It is noted that these values change between 0.2 and 0.8 in ECSA.

Figure 4.5 represents the development of *IW* values during the execution of 100 iterations of each compared PSO approach. As it can be seen, the *IW* values in BWIWPSO, the *IW* values decrease from 0.9 to 0.4. In IBWIWPSO, the *IW* values increase from 0.4 to 0.9.

Figure 4.6 represents the development of the values of the parameters *AP*, *fl*, and *IW* during the execution of 100 iterations of the proposed method. As is evident:

- The *AP* values are fixed to 0.1 during the first 25 iterations (the quarter of 100 iterations).
- The values of *fl* are also fixed to 0.5 during 25 iterations (The choices for *AP* and *fl* are fixed empirically to maximize the exploration phase during the initial 25 iterations).
- A low *AP* means the ”crows” are less cautious about others, and most of them are dedicated to the seeking mode (exploration). Coupling this with a moderate *fl* ensures the searching agents can take sufficiently wide steps across the search space.
- Both parameters are subsequently set to 0 to facilitate the transition.

These values enforce the principles of the CSA algorithm initially. Consequently, the *IW* values are first fixed at 0, and they become 0.4 from the 26th iteration to the end of the episode. The choice of the value 0.4 is based on the knowledge that in enhanced PSO algorithms, the *IW* is typically confined between 0.4 and 0.9. Knowing that higher values (closer to 0.9) encourage exploration of the search space, while lower values (closer to 0.4) encourage exploitation of existing solutions. Therefore, considering that the aim is to to exploit the power of the PSO algorithm in exploitation, choosing the value 0.4. Consequently, this demonstrates that the proposed algorithm first uses the principles of CSA and then changes to those of PSO. This confirms the assertion and successful proposition of a MAS path-planning algorithm with collision avoidance, which is based on a hybridization of the CSA and PSO algorithms, as hypothesized in the introduction.

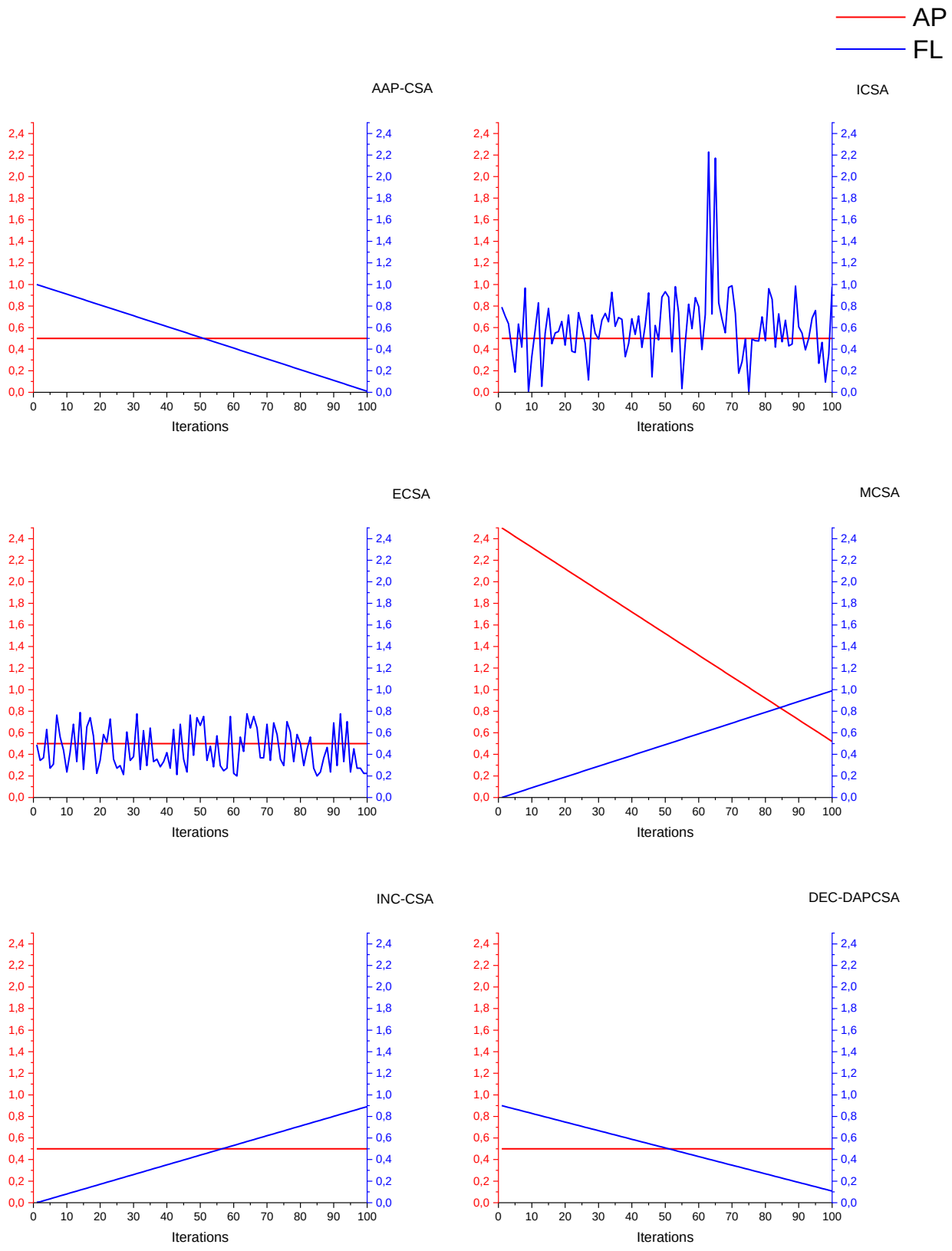

Figure 4.4: The development of *AP* and *fl* during an episode of 100 iterations in the 6 CSA's cases

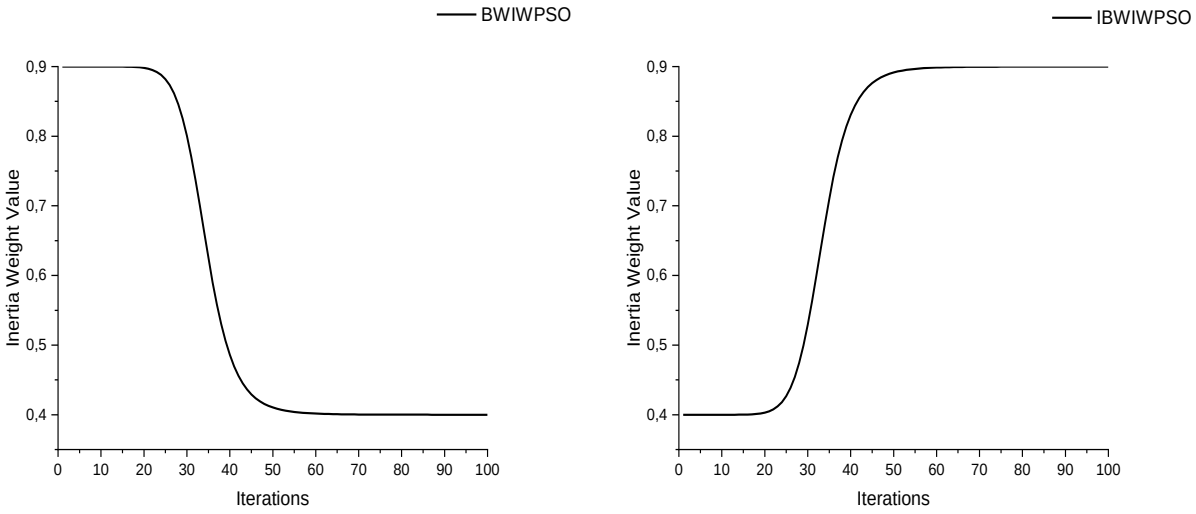

Figure 4.5: The development of *IW* during an episode of 100 iterations in the 2 PSO's cases

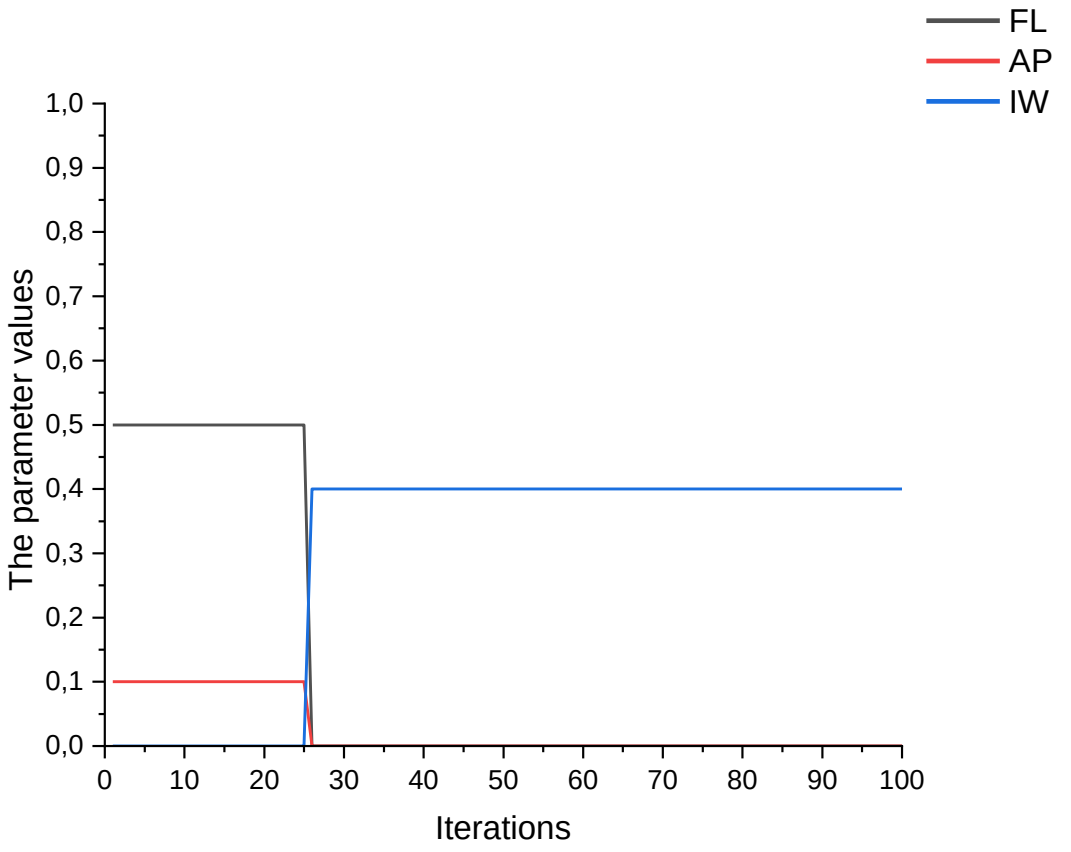

Figure 4.6: The development of CSO's parameters during an episode of 100 iterations

Figure 4.7 illustrates searching time obtained by the execution of 100 episodes of the proposed algorithm using the various methods compared. The searching time is determined by the number of iterations of the agents to find the target position without collision. The results are then summarized in

Table 4.1, which presents the global average searching time (calculated across all agents) achieved during 100 episodes for every compared algorithm. From the results, it is observed that:

- The highest average searching time is 98.38 iterations, which is obtained using the ECSA method.
- In terms of AAP-CSA, ICSA, MCSA, INC-CSA, and DEC-DAPCSA, the values obtained were close to those provided by ECSA, with a range of 90.41 to 97.17.
- An important decrease of 43.65% and 49.08% is noted in the average of BWIWPSO (54.73 iterations) and IBWIWPSO (49.3 iterations), respectively, in relation to the ECSA case.
- The proposed approach (CSO) surpasses all other approaches regarding searching time, by the best average rate of 43.22

Consequently, the results shown prove that the CSA' versions are the slowest. This is logical and confirms the previous assertion that the CSA method is strong in exploration; it struggles with exploitation. However, PSO methods showed improvement because of their ability to exploit. As a result, the proposed hybrid method has a positive impact on the MAS path planning problem with collision avoidance, minimizing the searching time while balancing the exploration and exploitation phases, thus validating the objectives stated in the introductory section.

	AAP-CSA	**ICSA**	**ECSA**	**MCSA**	**INC-CSA**	**DEC-DAPCSA**	**BW-IWPSO**	**IBW-IWPSO**	**CSO**
Global Avg search time	92.17	94.05	98.38	96.5	97.17	90.41	54.73	49.3	43.22

Table 4.1: The global average searching time achieved during the 100 episodes for every compared algorithm

Figure 4.8 represents the fitness development, which is determined by the best value found during a single episode. Each episode is composed of 100 iterations to locate the desired location using all the compared methods. Upon examination of the results obtained, it is apparent that:

- The maximum fitness found was obtained (0.98) in the 97th iteration through the application of MCSA and DEC-DAPCSA methods; this is, on the one hand. On the other hand, the two methods find the first best value (0.73, 0.96), respectively.
- The ICSA and INC-CSA found their maximum fitness value after 69 and 67 iterations, with values of 0.98 and 0.99, respectively. Noting that their first best values are 0.84 and 0.85, respectively.

- The ECSA case has first obtained 0.72, and then its best value is fixed at 0.91 after 16 iterations.
- The AAP-CSA method obtained 0.79 as the first best value and 0.97 in 38 iterations.
- In IBWPSO and BWPSO, the best values achieved are 0.99 and 0.98 only after 11 and 17 iterations, respectively. They both start with 0.76 and 0.55.
- The proposed method CSO obtained the highest value of 0.98 after 34 iterations with 0.81 as an initial best value found.

These results proved that the strength of the CSA versions in exploration is shown in the high initial best value, and then they are different in finding their highest value (some are fast and others are slow). Unlike PSO methods that start with low values because of their weakness in exploration, they are fast in finding the highest value. Therefore, the proposed approach allows the agents to reach their objective position in a minimum of time, starting with a high fitness value.

Consequently, the CSO is very efficient with respect to the convergence towards the optimal solution, achieving rapid convergence to resolve the MAS path-planning problem with collision avoidance, consistent with the premise outlined in the introduction.

Based on the results reflected in Figure 4.8, it is useful to study the difference between the fitness obtained in the $N + 1$ iteration and the N iteration during a complete episode, as shown in Figure 4.9.

- Generally, in the compared cases, the difference increases in the first iterations.
- The highest difference that the ECSA and INC-CSA cases produced is 0.72 and 0.85 in the first iteration and 0.8 in the 6th and 9th iterations, respectively.
- The AAP-CSA produced a difference of 0.79 in the first iteration; then the second highest one is 0.05 in the 8th iteration.
- The ICSA, MCSA, and DEC-DAPCSA cases reflected differences of 0.03, 0.09, and 0.01, respectively, in the 65th, 97th, and 27th iterations.
- The highest differences obtained in the BWIWPSO and IBWIWPSO cases were in the first iteration (0.55 and 0.76, respectively); then they changed to 0.13 and 0.07 in the 12th and 5th iterations.
- The difference obtained in the proposed case was up to 0.81 in the first iteration.

Consequently, the obtained results proved and confirmed the efficiency of the proposed method regarding the convergence towards the optimal solution. Compared to the proposed case, there are better and worse cases in terms of fitness development. The methodology is not hindered by this, allowing

one to state that this approach is highly effective in resolving the MAS path-planning problem with collision avoidance.

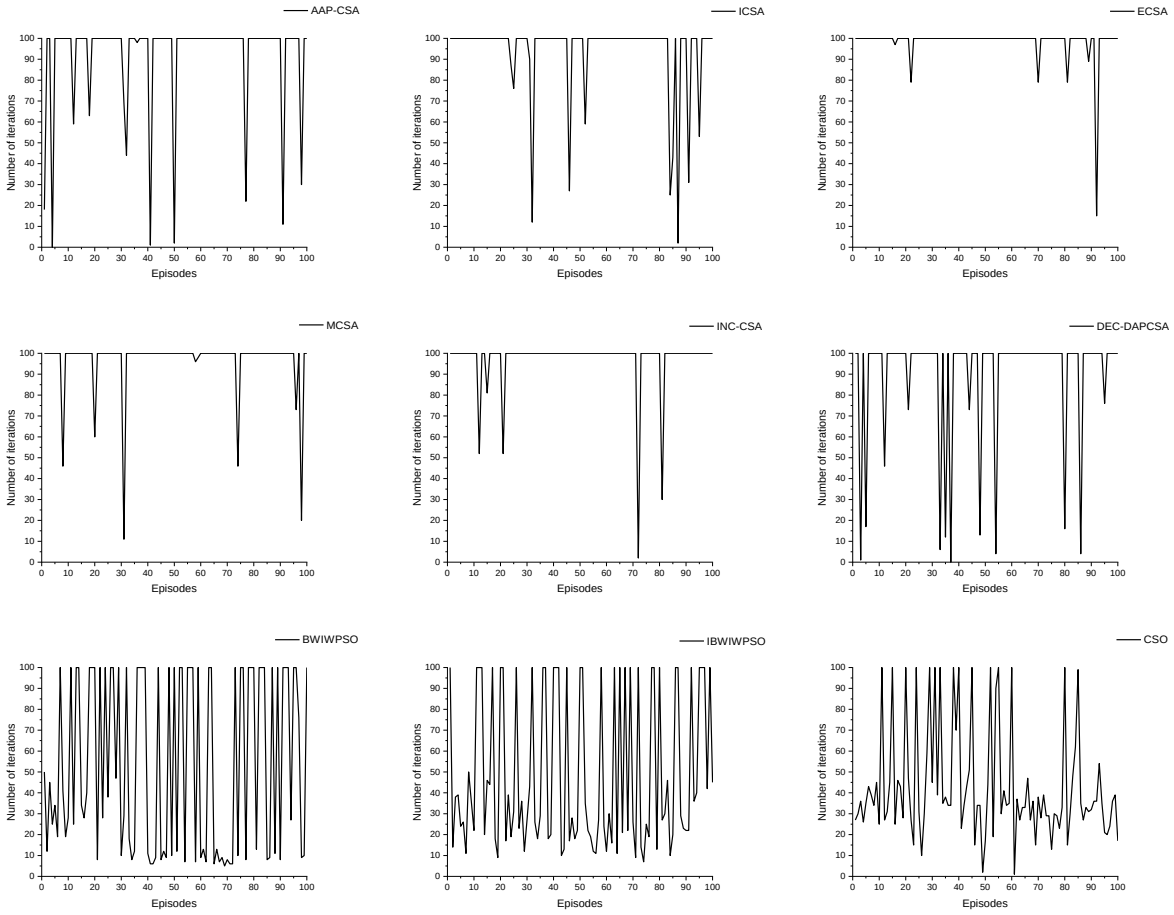

Figure 4.7: The searching time obtained by the execution of 100 episodes (9 cases)

To confirm the validity of the results of the previous analysis, an alternative simulation scenario is now introduced and characterized by initializing the agents with the same positions during each episode as shown in Figure 4.10. This idea was conceived to establish a fair comparison between all methods with the aim of eliminating the possibility of randomness to influence results in favor of any one algorithm. Subsequently, Figure 4.11 represents the searching time obtained by applying the compared cases during 100 episodes as the first scenario. The results are then summarized in Table 4.2, which presents the global average searching time achieved during 100 episodes for every compared algorithm, initializing the agents with the same positions during each episode. Furthermore, it is crucial to highlight that the parameters not explicitly mentioned have been kept the same as in the first scenario. Based on the results observed it is deduced that:

- The PSO-based approaches achieved better performance compared to the CSA-based approaches in terms of searching time's average.
- An average searching time between 90.76 and 96.01 was obtained using the 6 CSA cases. Noting that previously (results from Figure 4.7); the searching time obtained was between 90.41 and 98.38.
- The average of BWIWPSO cases (56.98) is approximately the same as the average of IBWIWPSO (51.88). It should be noted that these two values are close to their previous values in the first scenario (54.73 and 49.56, respectively).
- The CSO case reflects the lowest average searching time (45.56) compared to the other cases.

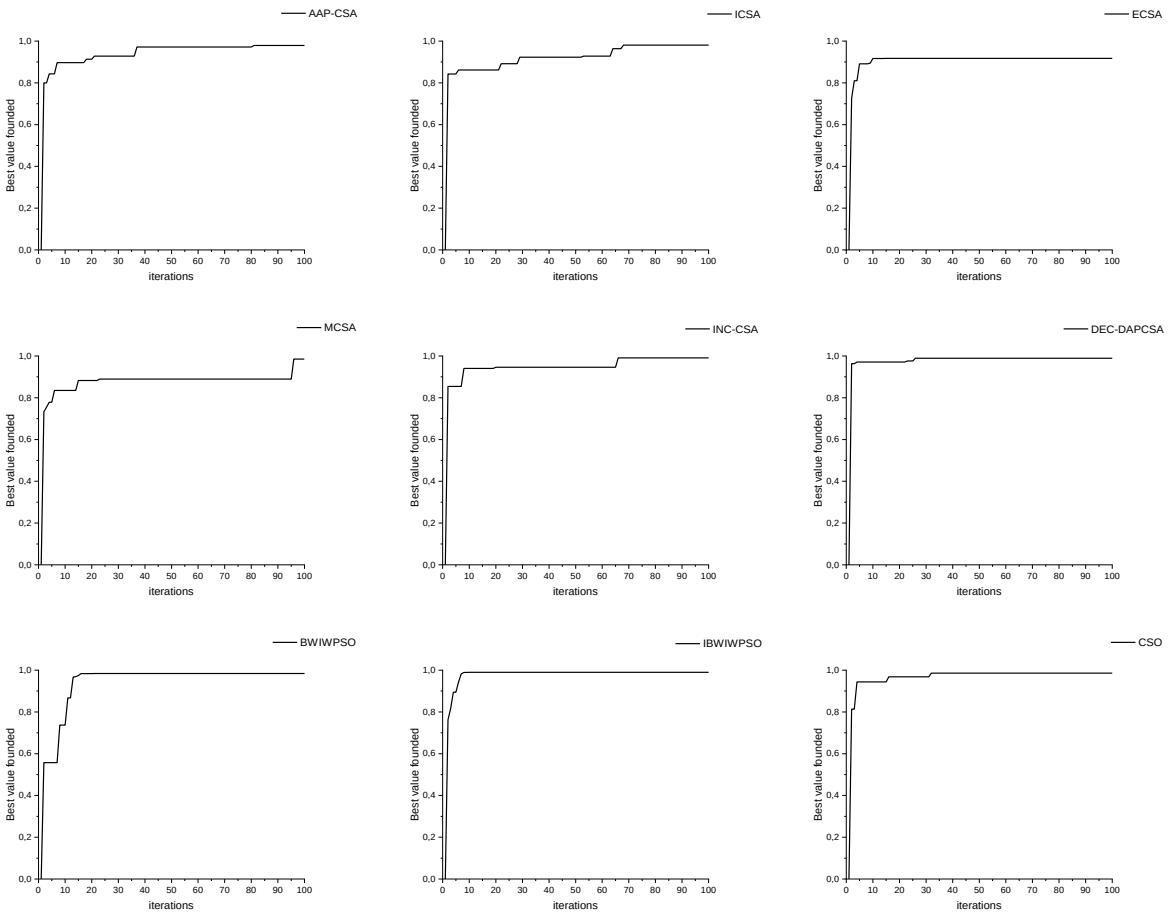

Figure 4.8: Fitness development during one episode of 100 iterations

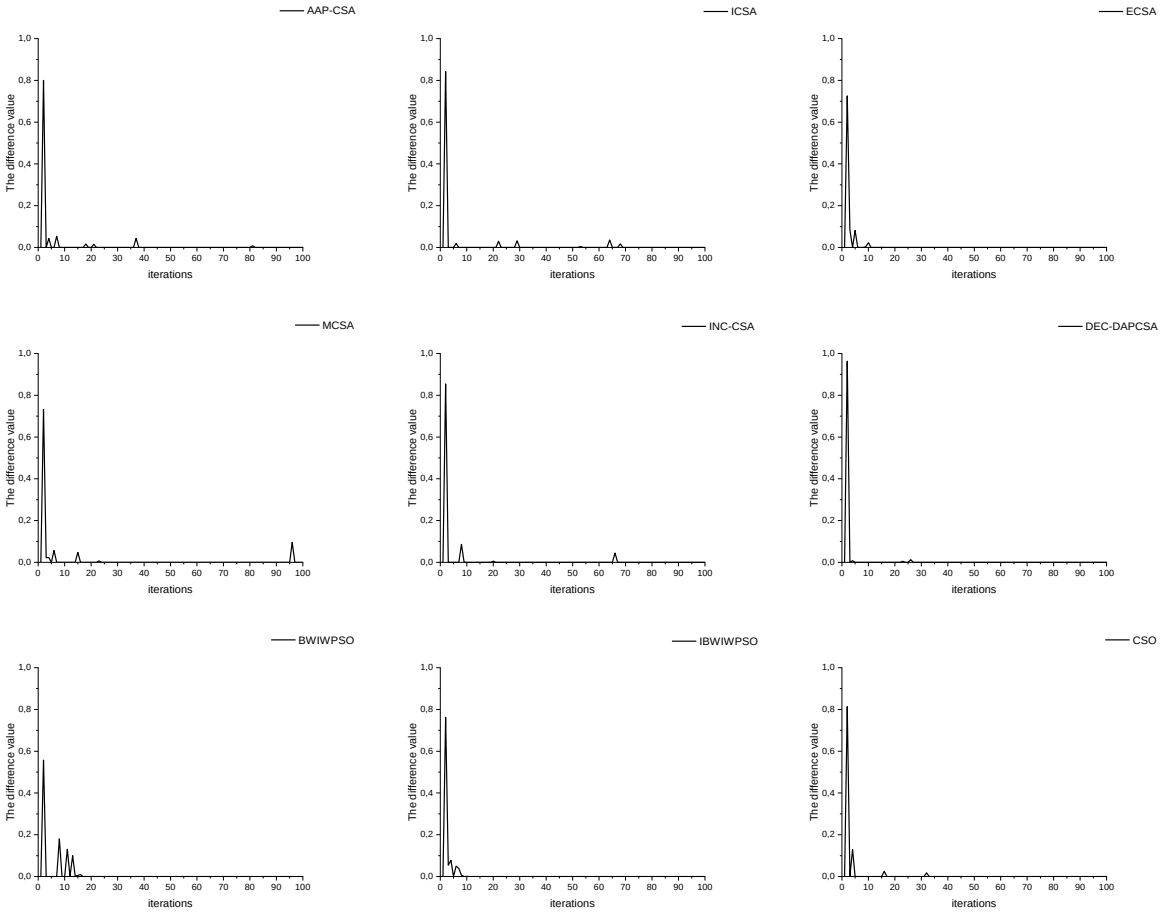

Figure 4.9: Difference between the *N+1* iteration fitness and the *N* iteration fitness during one episode of 100 iterations

	AAP-CSA	**ICSA**	**ECSA**	**MCSA**	**INC-CSA**	**DEC-DAPCSA**	**BW-IWPSO**	**IBW-IWPSO**	**CSO**
Global avg search time (same positions)	90.84	95.53	95.94	94.03	90.76	96.01	56.98	51.88	45.56
Previous global avg search time	92.17	94.05	98.38	96.5	97.17	90.41	54.73	49.3	43.22

Table 4.2: The global average searching time achieved during the 100 episodes for every compared algorithm (with same initial positions of agents)

Based on the results obtained, the proposed hybrid method has a positive impact on the MAS path planning problem with collision avoidance. It minimizes the searching time while balancing the exploration and exploitation phases, thus validating the objectives stated in the introductory section,

which was achieved through a fair comparison between all methods to eliminate the possibility of randomness influencing results in favor of any one algorithm.

Analysis of the data from figures 4.8 and 4.11 reveals that the cases exhibit significant delays in achieving the objective, as evidenced by their elevated average rates. Consequently, implementation involved an alternative scenario to validate that the proposed method demonstrates significant efficiency, exhibiting both optimized search time and scalability by employing a reduced agent population. This scenario builds upon the same configuration of the first scenario (shown in Figure 4.7), with an increase in the number of searching agents to 200.

Furthermore, it is important to note that this scenario applies only to the 6 CSA's compared methods, while the other cases (CSO, BWIWPSO, and IBWIWPSO) are well-established and do not require an increase in the number of agents to demonstrate their effectiveness. Figure 4.12 presents the searching time obtained through the application of the six compared cases over 100 episodes. This scenario intentionally increases the population size to 200 agents (building upon the configuration established in Figure 4.7). The results are then summarized in Table 4.3, which presents the global average searching time (calculated across all agents) achieved during 100 episodes for the 6 compared algorithms, increasing the number of agents during each episode. It can be concluded from the results that:

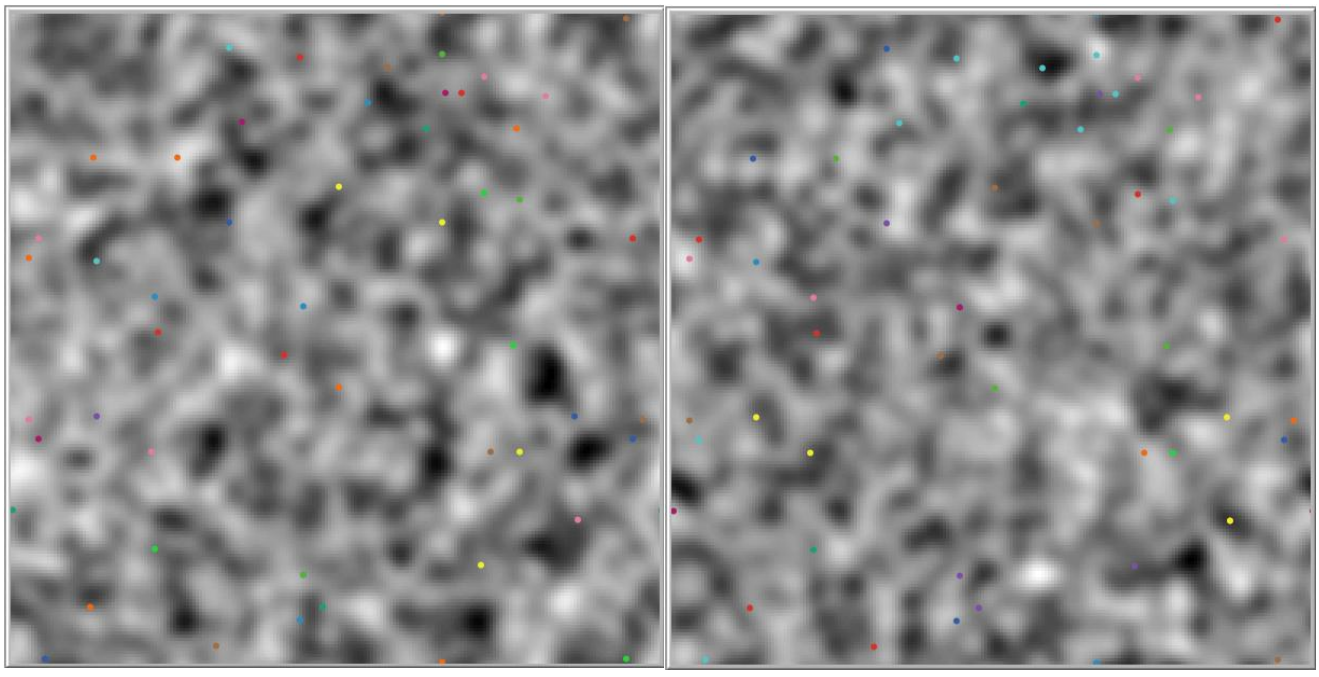

Figure 4.10: Example of two initializations

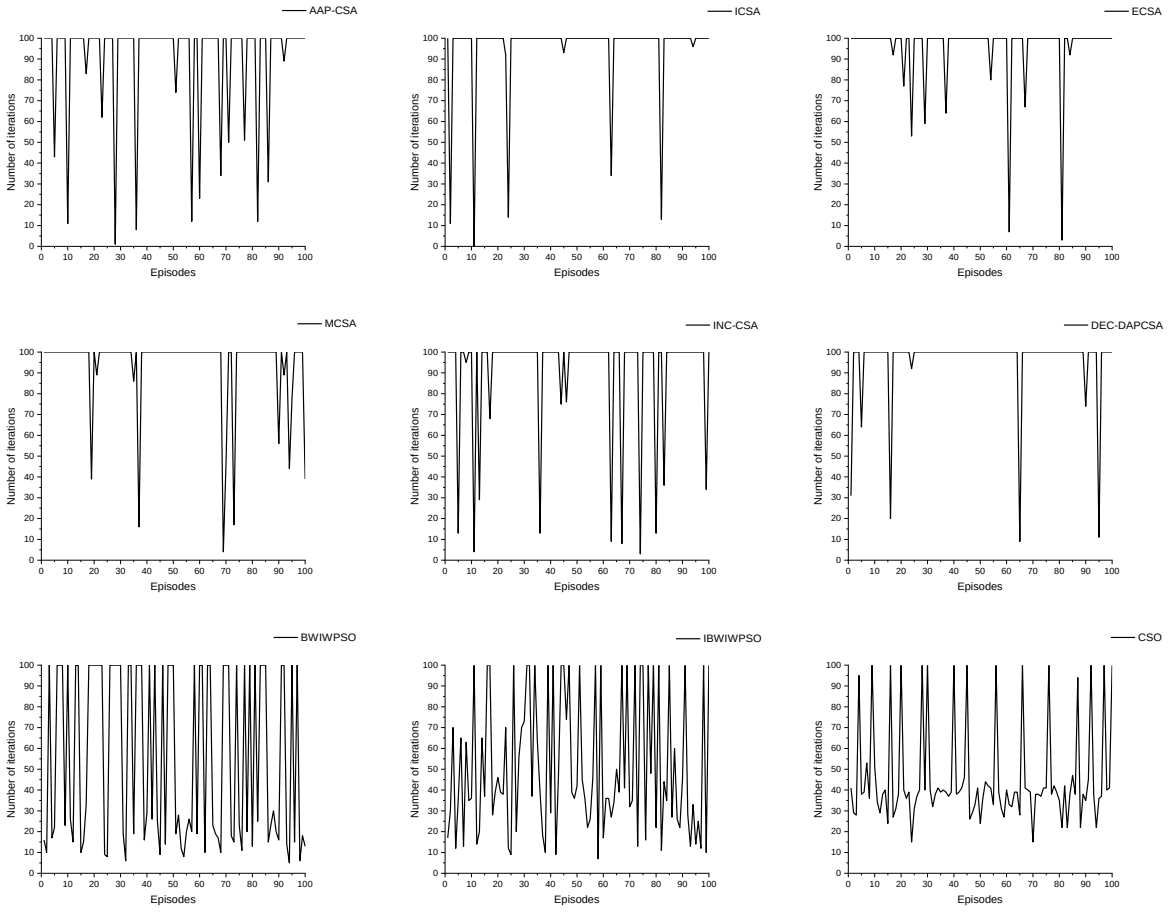

Figure 4.11: The searching time obtained by the execution of 100 episodes initializing agents with the same positions for the 9 cases

- The analysis observes a decrease in the average rate of AAP-CSA from 92.17 to 81.8.
- The ICSA and INC-CSA gave closed results (77.8 and 78.94) after giving (94.5 and 97.17) in the first scenario.
- ECSA and DEC-DAPCSA reflected (85.47 and 84.02) they have obtained previously (98.38 and 90.41).
- MCSA gave 80.08; this value was decreased from 96.5.
- The best searching time is obtained by the ICSA method.
- The results from the other algorithms are relatively close, it is also important to highlight that the decrease rate varies from 6.39% to 18.23%, which is considered a highly satisfactory improvement over the previous results.

Consequently, the obtained results proved the efficiency of the proposed hybrid method in enhancing scalability by employing a minimal agent set compared to the other methods, consistent with the premise outlined in the introduction.

	AAP-CSA	**ICSA**	**ECSA**	**MCSA**	**INC-CSA**	**DEC-DAPCSA**
Global avg search time (augmenting the agents's number)	81.8	77.8	85.47	80.08	78.94	84.02
Previous global avg search time	92.17	94.05	98,38	96.5	97.17	90.41

Table 4.3: The global average searching time achieved during the 100 episodes for every compared algorithm (increasing the number of agents)

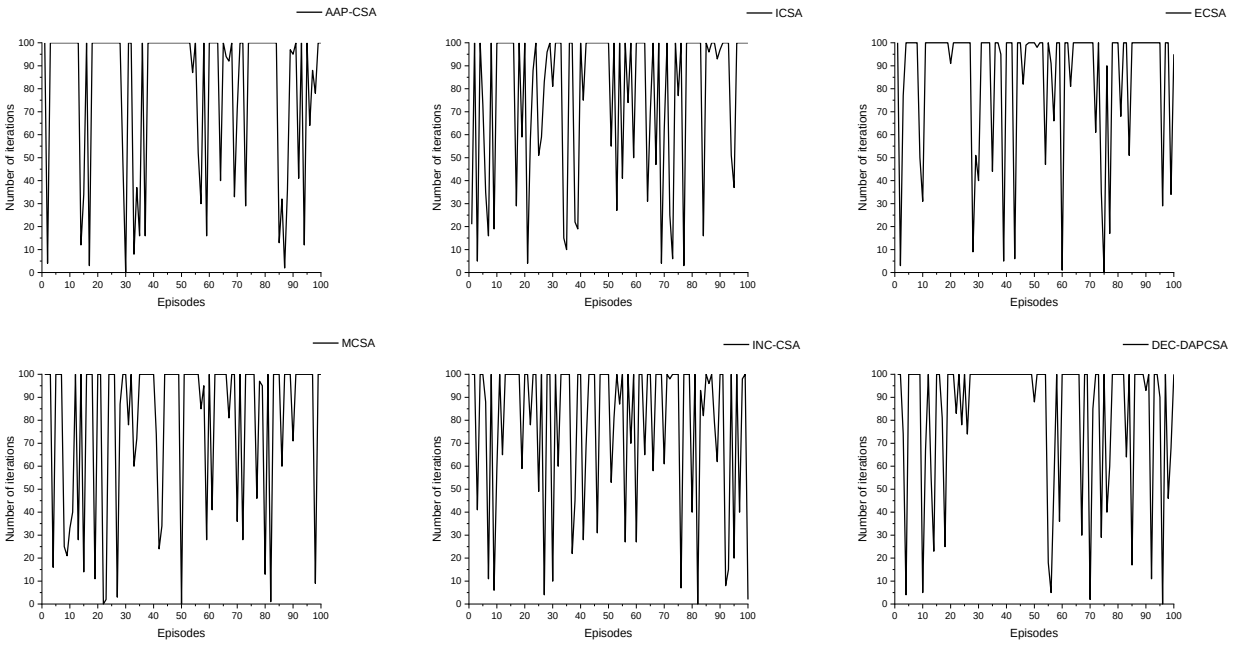

Figure 4.12: The searching time obtained by the execution of 100 episodes (6 cases) by 200 agents

To validate the results obtained in the previous analysis, another simulation scenario is introduced. This new scenario involves the direct calculation of the path length, which represents the accumulated distance traveled per agent over a single episode of 100 iterations, and is evaluated across all the compared algorithms, as shown in Figure 4.13. The results are then summarized in Table 4.4, which presents the global average path length (calculated across all agents) achieved during 100 episodes for every algorithm compared. Consequently, it was found that:

- All the CSA improvement versions (AAP-CSA, ICSA, ECSA, MCSA, INC-CSA, and DEC-DAPCSA) obtain closed high average values (65.28, 80.32, 67.95, 64.19, 62.52, and 60.63, respectively) compared to the proposed CSO that obtains 37.80.
- The PSO-improved versions, BWIWPSO and IBWIWPSO, achieve the lowest average path length compared to all other methods, including the CSO.

Consequently, the high path length confirms that the CSA exploration phase is inefficient; it is necessary to decrease the exploration to improve target finding. This is from the one hand. However, the analysis interprets this finding as a negative result that indicates that the PSO-improved versions struggled with exploration. The extremely low path length signifies that the agents did not traverse a sufficient distance to thoroughly search the global space, leading the swarm to get stuck in suboptimal local areas (local optima).

Therefore, the CSO algorithm satisfies three key objectives: it strikes a better compromise between exploration and exploitation than both the highly exploratory CSA variants and the highly exploitative PSO variants, thereby minimizing the path length, and effectively preventing premature convergence and trapping in local optima.

	AAP-CSA	**ICSA**	**ECSA**	**MCSA**	**INC-CSA**	**DEC-DAPCSA**	**BWIWPSO**	**IBWIWPSO**	**CSO**
Global Average Path Length	65.28	80.32	67.95	64.19	62.52	60.63	20.90	13.54	37.80

Table 4.4: The global average path length achieved during the 100-iteration episode for every compared algorithm

To confirm the strategic validity and necessity of the α constriction factor in preserving convergence efficiency under collision constraints, the study conducted an ablation study comparing two CSO variants that both utilize a zero-velocity punishment (v_i=0) in collision detection cases. The results, illustrated in Figure 4.14, clearly reflect the impact of the α factor on the primary metric, Searching Time (number of iterations in one episode of 100 iterations). The proposed CSO method (which uses the α constriction factor) achieved an average searching time of 48.1 iterations, significantly outperforming the second case (without the α), which recorded 76.63 iterations. This substantial difference in computational effort demonstrates that the presence of the α constriction factor significantly enhances the algorithm's global search capability, thereby minimizing the computational effort required for convergence. Consequently, this validation confirms that the α constriction factor is a high-performing regulation method that effectively prevents collisions by stabilizing the agent's

velocity and preserving search integrity, which is crucial for achieving rapid and safe convergence within the CSO framework, in accordance with the objectives stated in the Introduction.

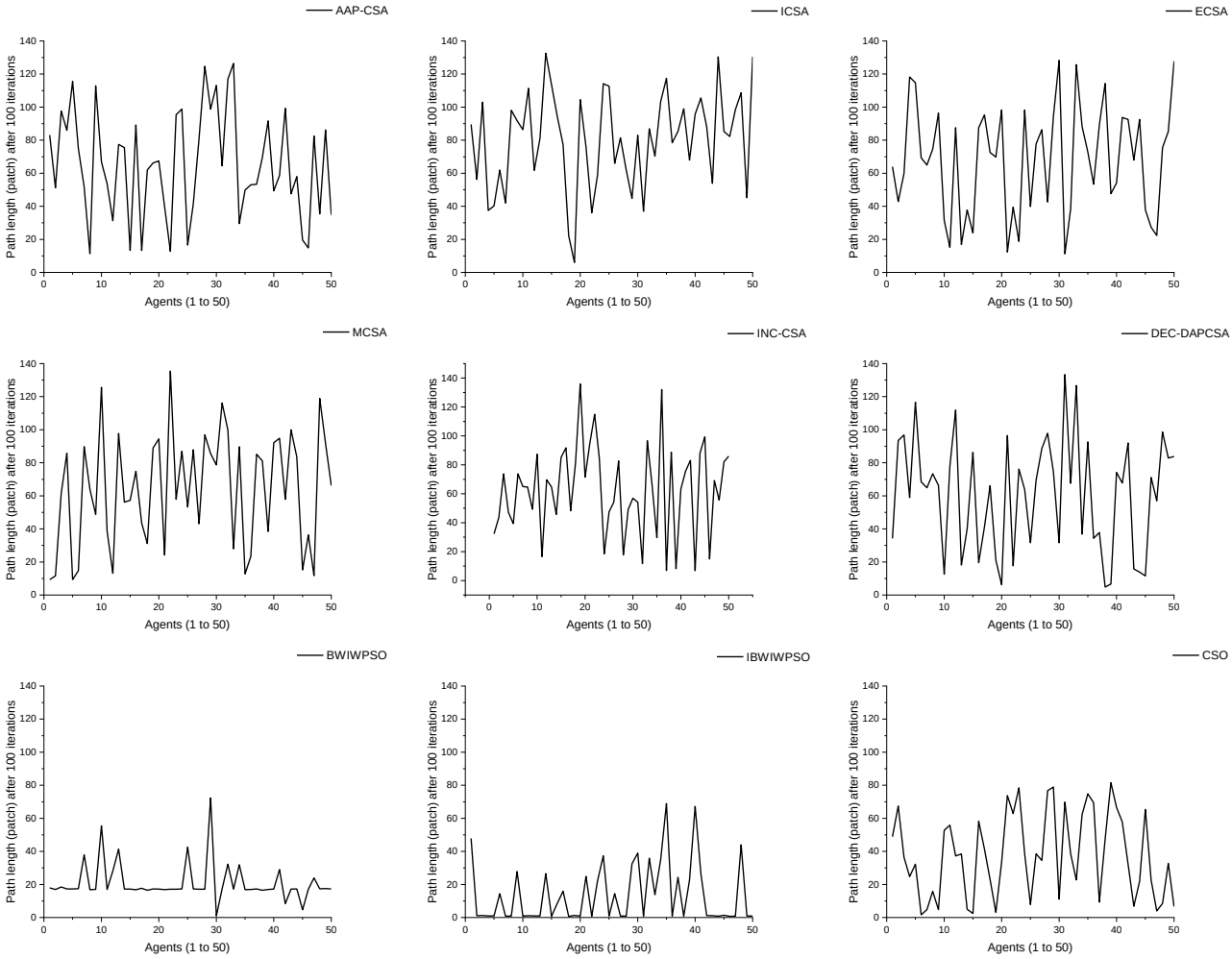

Figure 4.13: Path-length obtained by the execution of 100 episodes (9 cases)

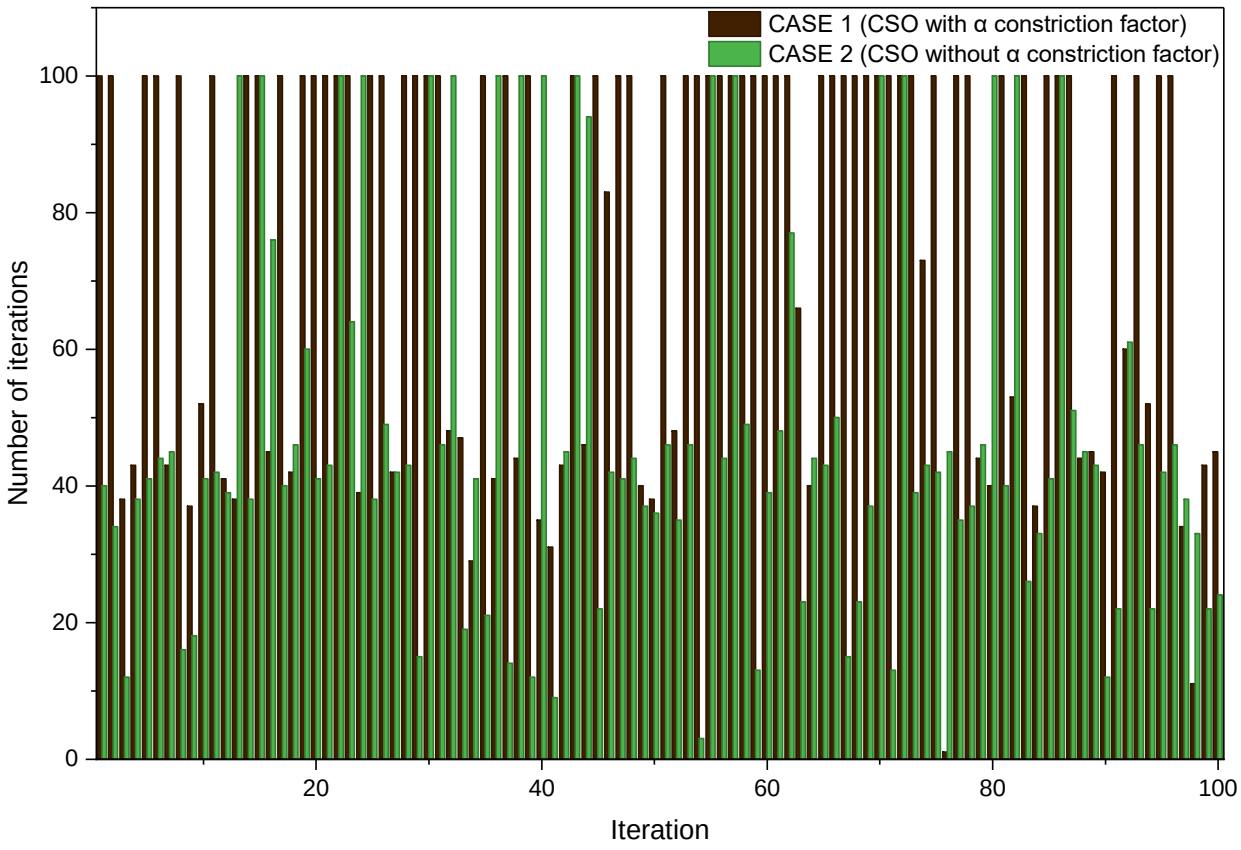

Figure 4.14: The impact of the α factor on the primary metric, Searching Time

This section introduced a comparative analysis of the proposed hybrid approach (CSO) in comparison with recent CSA and PSO versions based on *AP*, *fl* and *IW* improvement for solving the path-planning problem with collision avoidance. This proposed method is considered an improvement of CSA. Through these simulations, it was studied the searching time obtained using the compared methods during 100 episodes by using random initial positions of the agents. Furthermore, it was studied the fitness development obtained in an episode of 100 iterations. From the obtained results, it was observed that the proposed method (CSO) outperformed all the other methods. Moreover, it is noted that this hybrid method provides a significant improvement in path planning problems, due to its strong balance between the exploration and exploitation phases. Unlike other methods that have proven their strength in only the exploration or the exploitation. Specifically, the CSA improved methods are the slowest in terms of searching time, in contrast to the PSO ones that are better in this context. In addition, CSO enables agents to achieve their objectives in minimum time and efficiently converge to their optimal solutions. Moreover, to validate the obtained results, it was studied the searching time with the same

initial agents' positions to reduce the effect of randomness on the results. Secondly, the same scenario is repeated with augmenting the number of agents only for the slowed methods (CSA's improvement methods); they realize better results than in the first scenario. Furthermore, another metric, Path Length, is also studied; therefore, the CSO algorithm appears to strike a better compromise between exploration and exploitation than both the highly exploratory CSA variants and the highly exploitative PSO variants. Finally, the last scenario shows that the application of the α constriction factor in the proposed CSO method is empirically validated by its significantly lower searching time against the simple CSO one (that doesn't use the regulation velocity method). From this, the results proved the effectiveness of the proposed approach using the strength of CSA and PSO approaches. Specifically, it encourages initial exploration of the search space using CSA principles. Followed by an exploitation of the best solutions found using PSO principles. In addition, it optimizes scalability.

Briefly, the primary contribution of this research is the successful proposition and validation of the novel CSOMAS hybrid algorithm for efficient MAS path planning with collision avoidance. The core novel insight is that the CSO's superior capability to maintain a better compromise between exploration and exploitation than both CSA and PSO variants is fundamental, as it effectively eliminates the issue of local optima trapping and results in a significant minimization of searching time and path length. Critically, this performance translates directly to significant practical applications in real-world scenarios, such as enabling faster decision-making and resource conservation in autonomous vehicle fleets, and ensuring enhanced mission reliability and local optima avoidance in search-and-rescue operations. Consequently, this study advances the field of swarm optimization by establishing a robust framework for autonomous navigation. However, a limitation involves the computational overhead when extending to large-scale 3D dynamic environments, which future work will address by concentrating on developing a parallelized CSO framework and integrating self-adaptive parameter control mechanisms.

4.6 Summary

This chapter introduced a novel MAS path planning with collision avoidance based on a hybrid CSA approach with PSO. As a metaheuristic approach, CSA allows crows to move in their environment according to their memories, which helps them to save the best positions found. The algorithm, governed by the main parameters *AP* and *fl*, achieves a balance between the exploration and exploitation phases due to the temporal influence of these parameters. Despite this, CSA is known for its ability to explore the searching space; this is one side. On the other hand, the PSO allows the particles to move according to their own *pBest* and the *gBest* position detected by the swarm. Moreover, PSO is based on

the *IW* parameter, the controller of exploration and exploitation. Knowing that the PSO is strong in exploiting the existing solutions in the searching space.

This proposed algorithm provides a potentially effective solution to MAS path planning with collision avoidance, by capitalizing on the respective advantages of both algorithms – CSA's exploration and PSO's exploitation – the hybrid method can efficiently navigate complex environments, finding optimal or near-optimal paths while ensuring agents avoid collisions. To apply this version, the current study considered each agent as a crow/particle moving in the environment to find the location with the highest fitness value. To prevent collisions, agents mark their current and neighboring positions, creating a detectable field. This allows agents to recognize each other's presence, leading to velocity reduction and smooth movement without collision. Upon detection of collision, the velocity is attenuated by a multiplicative factor called the constriction factor α. To prove the feasibility of the proposed method, this study compared it with other CSA and PSO versions based on the improvements in *AP*, *fl* and *IW*. Based on the results presented, it was observed that the CSO allows a decrease in the searching time and an increase in fitness acquisition compared to the other versions. In addition, the proposed version aimed to increase the population size and enhance scalability. Additionally, the proposed CSO algorithm proves its superior performance and balance by achieving a shorter path length across varied test cases, which strikes a better compromise between exploration and exploitation than its base variants. Furthermore, the algorithm also demonstrates a significantly lower searching time, which empirically validates the α-constriction factor.

Consequently, the results empirically confirm the efficacy of the functionally segregated exploration/exploitation phases, showing this structural approach is key to achieving both fast global convergence and safety. Nevertheless, the present study acknowledges that the current model uses a simplified, discrete collision mechanism within a 100x100 grid, which differs from real-world continuous environments involving kinematics. However, the core principles of the hybrid CSO, particularly the velocity-based updates and the α constriction factor, are robust and scalable.

To build upon the validated success of the CSO-MAS hybrid algorithm, future research will concentrate on enhancing its operational capabilities and real-world applicability by first extending the framework to 3D and high-dimensional spaces—potentially through hybridization with improved PSO variants to ensure robust performance and replacing discrete sensing checks with the Time-to-Collision (TTC) prediction metric for continuous space validation. A crucial next step involves integrating the path planning with low-level control to ensure kinematically feasible trajectories, specifically by adapting the α-constriction factor to govern actual control inputs (like steering or thrust), effectively transforming

CSO into a dynamic constraint controller capable of generating safe and feasible paths that respect the agent's physical limits in complex, dynamic environments. Finally, to maximize efficiency and resilience, a crucial next step involves exploring adaptive scheduling of the α-constriction factor and *AP/fl* parameters, introducing mechanisms that dynamically adjust these values based on observed environmental complexity; alongside this, will incorporate practical constraints such as noise sensitivity and energy consumption directly into the optimization objective function to ensure the generated paths are not only mathematically optimal but also physically executable and sustainable.

GENERAL CONCLUSION

This research contributes to addressing the complex challenge of MAS path planning and task coordination in complex and dynamic environments by using and enhancing metaheuristic algorithms. This study was prompted by the limitations of traditional optimization methods and the need to capitalize on the capabilities of metaheuristics, specifically their ability to efficiently search vast, large, and complicated solution spaces. To do this, this work begins by introducing key concepts and then explores and analyzes different recent applications. Focus was placed on the use of metaheuristics in path and task coordination to solve different optimization problems. In this analysis, the study highlighted the key strengths and weaknesses of each metaheuristic studied, such as parameter sensitivity, the balance between exploration and exploitation, local minima, scalability, and computational capability. Their effectiveness was found to be highly dependent on the specific problem. Based on these findings, these two contributions aim to expand on these identified limitations to solve especially the MAS path-planning problem.

Starting with the first contribution, which provides an enhanced PSO algorithm named IB-PSO. It presents a new version of the PSO algorithm that uses an inversed *IW* calculation method, inspired by the Butterworth function. This version gives an increasing function that prevents this algorithm from getting trapped in the local minima problem. To apply it to the MAS path planning, each agent was considered a particle moving in the environment to find the location with the highest fitness value. It improved the algorithm's balance between exploration and exploitation in the search space, enhancing its ability to find optimal paths more efficiently. The performance of the IB-PSO method was thoroughly evaluated and outperformed other improved methods by allowing a decrease in the searching time and an increase in the fitness acquisition in comparison with them.

Furthermore, the second contribution provides a hybrid approach named CSO. It combines the CSA and the PSO algorithms. This strategy effectively integrates the strong exploration capabilities of CSA with the robust exploitation capabilities of PSO. Its process starts by discovering the search space using agents as crows (CSA phase), then it turn to PSO phase using agents as particles that must navigate to achieve the target location. Furthermore, this approach elevates the challenge of path planning by successfully incorporating collision avoidance. A key feature for achieving this goal integrated velocity regulation mechanism, which successfully prevents agents from colliding, by decreasing the velocity in detecting collisions cases. To prove the performance of the proposed method, it has been compared to other CSA and PSO versions based on *AP*, *fl* and *IW* parameters improvement. According to the showcased results, it has been constated that the CSO allows a decrease in the searching time and an increase in the fitness acquisition in comparison with the other versions.

In essence, this work focused on both theoretical and practical key principles for applying metaheuristics to MAS. The proposed algorithms, IB-PSO and CSO, provide significant enhancement in the MAS field. They offer more robust solutions for coordination problems. Consequently, this thesis highlights the critical need for a new perspective on exploring mechanisms for dynamically adjusting the parameters of the proposed algorithms in real-time. Also, future research could hybridize the proposed CSO approach with other improved versions of PSO to enable operation in a 3D search space to ensure safety. Nevertheless, its applicability may extend to path planning scenarios requiring obstacle avoidance.

REFERENCES

REFERENCES

1. Ahmad, H. F. (2002). Multi-agent systems : Overview of a new paradigm for distributed systems. *7th IEEE International Symposium on High Assurance Systems Engineering, 2002. Proceedings.*, 101-107. https://doi.org/10.1109/HASE.2002.1173110

2. Wooldridge, M. J. (2009). *An introduction to multiagent systems* (2nd ed). John Wiley & Sons.

3. Dorri, A., & Jurdak, R. (2018). *Multi-Agent Systems : A Survey*. *6*.

4. Jennings, N. R. (1993). Commitments and conventions : The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, *8*(3), 223-250. https://doi.org/10.1017/S0269888900000205

5. Wooldridge, M., & Jennings, N. R. (s. d.). *Intelligent agents : Theory and practice*.

6. Nwana, H. S. (1996). Software agents : An overview. *The Knowledge Engineering Review*, *11*(3), 205-244. https://doi.org/10.1017/S026988890000789X

7. Weiss, G. (Éd.). (2001). *Multiagent systems : A modern approach to distributed artificial intelligence* (3. print). MIT Press.

8. *Holons and agents*. (s. d.).

9. Van Der Hoek, W., & Wooldridge, M. (2008). Chapter 24 Multi-Agent Systems. In *Foundations of Artificial Intelligence* (Vol. 3, p. 887-928). Elsevier. https://doi.org/10.1016/S1574-6526(07)03024-6

10. Da Silva, P. S., & De Melo, A. C. V. (2011). A Formal Environment Model for Multi-Agent Systems. In J. Davies, L. Silva, & A. Simao (Éds.), *Formal Methods : Foundations and Applications* (Vol. 6527, p. 64-79). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-19829-8_5

11. Caridi, M., & Cavalieri, S. (2004). Multi-agent systems in production planning and control : An overview. *Production Planning & Control*, *15*(2), 106-118. https://doi.org/10.1080/09537280410001662556

12. Alshabi, W., Ramaswamy, S., Itmi, M., & Abdulrab, H. (2007). Coordination, Cooperation and Conflict Resolution in Multi-Agent Systems. In T. Sobh (Éd.), *Innovations and Advanced Techniques in Computer and Information Sciences and Engineering* (p. 495-500). Springer Netherlands. https://doi.org/10.1007/978-1-4020-6268-1_87

13. Beigi, A., & Mozayani, N. (2016). Dialogue strategy for horizontal communication in MAS organization. *Computational and Mathematical Organization Theory*, *22*(2), 161-183. https://doi.org/10.1007/s10588-015-9201-1

14. Valckenaers, P., Hadeli, Germain, B. S., Verstraete, P., & Van Brussel, H. (2007). MAS coordination and control based on stigmergy. *Computers in Industry*, *58*(7), 621-629. https://doi.org/10.1016/j.compind.2007.05.003

15. Hennet, J.-C., & Arda, Y. (2008). Supply chain coordination : A game-theory approach. *Engineering Applications of Artificial Intelligence*, *21*(3), 399-405. https://doi.org/10.1016/j.engappai.2007.10.003

16. Teixeira, M., d'Orey, P. M., & Kokkinogenis, Z. (2019). Autonomous Vehicles Coordination Through Voting-Based Decision-Making. In M. Lujak (Éd.), *Agreement Technologies* (Vol. 11327, p. 199-207). Springer International Publishing. https://doi.org/10.1007/978-3-030-17294-7_15

17. Xie, J., & Liu, C.-C. (2017). Multi-agent systems and their applications. *Journal of International Council on Electrical Engineering*, *7*(1), 188-197. https://doi.org/10.1080/22348972.2017.1348890

18. Mariani, S., & Omicini, A. (s. d.). *Multi-paradigm Coordination for MAS: Integrating Heterogeneous Coordination Approaches in MAS Technologies*.

19. Almeida, F., Lau, N., & Reis, L. P. (s. d.). *A Survey on Coordination Methodologies for Simulated Robotic Soccer Teams*.

20. Hong, Y., Pearson, J. N., & Carr, A. S. (2009). A typology of coordination strategy in multi-organizational product development. *International Journal of Operations & Production Management*, *29*(10), 1000-1024. https://doi.org/10.1108/01443570910993465

21. Hernández, J. E., Mula, J., Poler, R., & Lyons, A. C. (2014). Collaborative Planning in Multi-tier Supply Chains Supported by a Negotiation-Based Mechanism and Multi-agent System. *Group Decision and Negotiation*, *23*(2), 235-269. https://doi.org/10.1007/s10726-013-9358-2

22. Hassan, I. A., Abed, I. A., & Al-Hussaibi, W. A. (2023). Navigation and Path Planning of Mobile Robots. *Journal Européen Des Systèmes Automatisés*, *56*(5), 751-756. https://doi.org/10.18280/jesa.560505

23. Mac, T. T., Copot, C., Tran, D. T., & De Keyser, R. (2016). Heuristic approaches in robot path planning : A survey. *Robotics and Autonomous Systems*, *86*, 13-28. https://doi.org/10.1016/j.robot.2016.08.001

24. Fines, K., Sharpanskykh, A., & Vert, M. (2020). Agent-Based Distributed Planning and Coordination for Resilient Airport Surface Movement Operations. *Aerospace*, *7*(4), 48. https://doi.org/10.3390/aerospace7040048

25. Parker, L. E. (2009). Multiple Mobile Robot Teams, Path Planning and Motion Coordination in. In R. A. Meyers (Éd.), *Encyclopedia of Complexity and Systems Science* (p. 5783-5800). Springer New York. https://doi.org/10.1007/978-0-387-30440-3_344

26. Zhao, J., Zhao, W., Deng, B., Wang, Z., Zhang, F., Zheng, W., Cao, W., Nan, J., Lian, Y., & Burke, A. F. (2024). Autonomous driving system : A comprehensive survey. *Expert Systems with Applications*, *242*, 122836. https://doi.org/10.1016/j.eswa.2023.122836

27. Wahab, M. N. A., Nefti-Meziani, S., & Atyabi, A. (2020). A comparative review on mobile robot path planning : Classical or meta-heuristic methods? *Annual Reviews in Control*, *50*, 233-252. https://doi.org/10.1016/j.arcontrol.2020.10.001

28. Tan, C. S., Mohd-Mokhtar, R., & Arshad, M. R. (2021). A Comprehensive Review of Coverage Path Planning in Robotics Using Classical and Heuristic Algorithms. *IEEE Access*, *9*, 119310-119342. https://doi.org/10.1109/ACCESS.2021.3108177

29. Dong Eui Chang, Shadden, S. C., Marsden, J. E., & Olfati-Saber, R. (2003). Collision avoidance for multiple agent systems. *42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)*, *1*, 539-543. https://doi.org/10.1109/CDC.2003.1272619

30. Katona, K., Neamah, H. A., & Korondi, P. (2024). Obstacle Avoidance and Path Planning Methods for Autonomous Navigation of Mobile Robot. *Sensors*, *24*(11), 3573. https://doi.org/10.3390/s24113573

31. *284745_615758*. (s. d.).

32. Yazdanpanah, V., Dastani, M., Fatima, S., Jennings, N. R., Yazan, D. M., & Zijm, H. (2020). Task Coordination in Multiagent Systems. *New Zealand*.

33. Bassiliades, N., Chalkiadakis, G., & De Jonge, D. (Éds.). (2020). *Multi-Agent Systems and Agreement Technologies : 17th European Conference, EUMAS 2020, and 7th International Conference, AT 2020, Thessaloniki, Greece, September 14-15, 2020, Revised Selected Papers* (Vol. 12520). Springer International Publishing. https://doi.org/10.1007/978-3-030-66412-1

34. Guo, M., & Dimarogonas, D. V. (2017). Task and Motion Coordination for Heterogeneous Multiagent Systems With Loosely Coupled Local Tasks. *IEEE Transactions on Automation Science and Engineering*, *14*(2), 797-808. https://doi.org/10.1109/TASE.2016.2628389

35. Abdel-Basset, M., Abdel-Fatah, L., & Sangaiah, A. K. (2018). Metaheuristic Algorithms : A Comprehensive Review. In *Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications* (p. 185-231). Elsevier. https://doi.org/10.1016/B978-0-12-813314-9.00010-4

36. Gogna, A., & Tayal, A. (2013). Metaheuristics : Review and application. *Journal of Experimental & Theoretical Artificial Intelligence*, *25*(4), 503-526. https://doi.org/10.1080/0952813X.2013.782347

37. *Metaheuristics.pdf*. (s. d.).

38. Yang, T., Yi, X., Wu, J., Yuan, Y., Wu, D., Meng, Z., Hong, Y., Wang, H., Lin, Z., & Johansson, K. H. (2019). A survey of distributed optimization. *Annual Reviews in Control*, *47*, 278-305. https://doi.org/10.1016/j.arcontrol.2019.05.006

39. Hills, T. T., Todd, P. M., Lazer, D., Redish, A. D., & Couzin, I. D. (2015). Exploration versus exploitation in space, mind, and society. *Trends in Cognitive Sciences*, *19*(1), 46-54. https://doi.org/10.1016/j.tics.2014.10.004

40. Fadhil, H. (2025, février 4). Metaheuristic Algorithms in Optimization and its Application : A Review. *Proceedings of the 3rd International Conference on Engineering and Innovative Technology*. The 3rd International Conference On Engineering And Innovative Technology. https://doi.org/10.31972/iceit2024.013

41. Talbi, E.-G. (2009). *Metaheuristics : From design to implementation*. John Wiley & Sons.

42. Raidl, G. R., Puchinger, J., & Blum, C. (2019). Metaheuristic Hybrids. In M. Gendreau & J.-Y. Potvin (Éds.), *Handbook of Metaheuristics* (Vol. 272, p. 385-417). Springer International Publishing. https://doi.org/10.1007/978-3-319-91086-4_12

43. Ting, T. O., Yang, X.-S., Cheng, S., & Huang, K. (2015). Hybrid Metaheuristic Algorithms : Past, Present, and Future. In X.-S. Yang (Éd.), *Recent Advances in Swarm Intelligence and Evolutionary Computation* (Vol. 585, p. 71-83). Springer International Publishing. https://doi.org/10.1007/978-3-319-13826-8_4

44. Eberhart, R., & Kennedy, J. (s. d.). *Zyzxywxv ANew OptimizerUsingParticleSwarmTheory*.

45. Gupta, A., & Srivastava, S. (2020). Comparative Analysis of Ant Colony and Particle Swarm Optimization Algorithms for Distance Optimization. *Procedia Computer Science*, *173*, 245-253. https://doi.org/10.1016/j.procs.2020.06.029

46. DORIGO, M. (1992). Optimization, Learning and Natural Algorithms. *Ph.D. Thesis, Politecnico di Milano*. https://cir.nii.ac.jp/crid/1573950400977139328

47. Guntsch, M., & Middendorf, M. (2002). Applying Population Based ACO to Dynamic Optimization Problems. In M. Dorigo, G. Di Caro, & M. Sampels (Éds.), *Ant Algorithms* (Vol. 2463, p. 111-122). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-45724-0_10

48. Kuan Yew Wong & Komarudin. (2008). Parameter tuning for ant colony optimization : A review. *2008 International Conference on Computer and Communication Engineering*, 542-545. https://doi.org/10.1109/ICCCE.2008.4580662

49. Karaboga, D. (s. d.). *AN IDEA BASED ON HONEY BEE SWARM FOR NUMERICAL OPTIMIZATION*.

50. Wei-feng Gao, San-yang Liu, & Ling-ling Huang. (2013). A Novel Artificial Bee Colony Algorithm Based on Modified Search Equation and Orthogonal Learning. *IEEE Transactions on Cybernetics*, *43*(3), 1011-1024. https://doi.org/10.1109/TSMCB.2012.2222373

51. Sampson, J. R. (1976). Adaptation in Natural and Artificial Systems (John H. Holland). *SIAM Review*, *18*(3), 529-530. https://doi.org/10.1137/1018105

52. Lipowski, A., & Lipowska, D. (2012). Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and Its Applications*, *391*(6), 2193-2196. https://doi.org/10.1016/j.physa.2011.12.004

53. Bazi, S., Benzid, R., Bazi, Y., & Rahhal, M. M. A. (2021). A Fast Firefly Algorithm for Function Optimization : Application to the Control of BLDC Motor. *Sensors*, *21*(16), 5267. https://doi.org/10.3390/s21165267

54. Kumar, V., & Kumar, D. (2021). A Systematic Review on Firefly Algorithm : Past, Present, and Future. *Archives of Computational Methods in Engineering*, *28*(4), 3269-3291. https://doi.org/10.1007/s11831-020-09498-y

55. Yang, X.-S., & Deb, S. (2009). *Cuckoo Search via Le´vy Flights*.

56. Akbari, M., & Rashidi, H. (2016). A multi-objectives scheduling algorithm based on cuckoo optimization for task allocation problem at compile time in heterogeneous systems. *Expert Systems with Applications*, *60*, 234-248. https://doi.org/10.1016/j.eswa.2016.05.014

57. Joshi, A. S., Kulkarni, O., Kakandikar, G. M., & Nandedkar, V. M. (2017). Cuckoo Search Optimization- A Review. *Materials Today: Proceedings*, *4*(8), 7262-7269. https://doi.org/10.1016/j.matpr.2017.07.055

58. Mirjalili, S., & Lewis, A. (2016). The Whale Optimization Algorithm. *Advances in Engineering Software*, *95*, 51-67. https://doi.org/10.1016/j.advengsoft.2016.01.008

59. Gharehchopogh, F. S., & Gholizadeh, H. (2019). A comprehensive survey : Whale Optimization Algorithm and its applications. *Swarm and Evolutionary Computation*, *48*, 1-24. https://doi.org/10.1016/j.swevo.2019.03.004

60. Hussien, A. G., Amin, M., Wang, M., Liang, G., Alsanad, A., Gumaei, A., & Chen, H. (2020). Crow Search Algorithm : Theory, Recent Advances, and Applications. *IEEE Access*, *8*, 173548-173565. https://doi.org/10.1109/ACCESS.2020.3024108

61. Król, D., & Drożdżowski, M. (2010). Use of MaSE methodology for designing a swarm-based multi-agent system. *Journal of Intelligent & Fuzzy Systems*, *21*(3), 221-231. https://doi.org/10.3233/IFS-2010-0453

62. Milano, M., & Roli, A. (2004). MAGMA : A Multiagent Architecture for Metaheuristics. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, *34*(2), 925-941. https://doi.org/10.1109/TSMCB.2003.818432

63. Houssein, E. H., Saeed, M. K., Hu, G., & Al-Sayed, M. M. (2024). Metaheuristics for Solving Global and Engineering Optimization Problems : Review, Applications, Open Issues and Challenges. *Archives of Computational Methods in Engineering*. https://doi.org/10.1007/s11831-024-10168-6

64. Yahia, H. S., & Mohammed, A. S. (2023). Path planning optimization in unmanned aerial vehicles using meta-heuristic algorithms : A systematic review. *Environmental Monitoring and Assessment*, *195*(1), 30. https://doi.org/10.1007/s10661-022-10590-y

65. Dhulkefl, E. J., Durdu, A., & Electrical and Electronic Engineering. (2019). Path Planning Algorithms for Unmanned Aerial Vehicles. *International Journal of Trend in Scientific Research and Development, Volume-3*(Issue-4), 359-362. https://doi.org/10.31142/ijtsrd23696

66. Saeed, R. A., Omri, M., Abdel-Khalek, S., Ali, E. S., & Alotaibi, M. F. (2022). Optimal path planning for drones based on swarm intelligence algorithm. *Neural Computing and Applications*, *34*(12), 10133-10155. https://doi.org/10.1007/s00521-022-06998-9

67. Biswas, S., Anavatti, S. G., & Garratt, M. A. (2017). Particle swarm optimization based co-operative task assignment and path planning for multi-agent system. *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, 1-6. https://doi.org/10.1109/SSCI.2017.8280872

68. Nayeem, G. M., Fan, M., & Akhter, Y. (2021). A Time-Varying Adaptive Inertia Weight based Modified PSO Algorithm for UAV Path Planning. *2021 2nd International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST)*, 573-576. https://doi.org/10.1109/ICREST51555.2021.9331101

69. Sun, B., & Niu, N. (2024). Multi-AUVs cooperative path planning in 3D underwater terrain and vortex environments based on improved multi-objective particle swarm optimization algorithm. *Ocean Engineering*, *311*, 118944. https://doi.org/10.1016/j.oceaneng.2024.118944

70. Najm, H. T., Ahmad, N. S., & Al-Araji, A. S. (2024). Enhanced path planning algorithm via hybrid WOA-PSO for differential wheeled mobile robots. *Systems Science & Control Engineering*, *12*(1), 2334301. https://doi.org/10.1080/21642583.2024.2334301

71. Trujillo-Romero, F. (s. d.). *Planificación de trayectorias usando metaheurísticas*.

72. Zheng, Y., Wang, L., & Xi, P. (2018). Improved Ant Colony Algorithm for Multi-Agent Path Planning in Dynamic Environment. *2018 International Conference on Sensing,Diagnostics, Prognostics, and Control (SDPC)*, 732-737. https://doi.org/10.1109/SDPC.2018.8664885

73. Huang, S., Yang, D., Zhong, C., Yan, S., & Zhang, L. (2021). An Improved Ant Colony Optimization Algorithm for Multi-Agent Path Planning. *2021 International Conference on Networking Systems of AI (INSAI)*, 95-100. https://doi.org/10.1109/INSAI54028.2021.00028

74. Bahabadi, M. H. J., Mahdavi, A., & Khankalantary, S. (2024). Heterogeneous Coverage Path Planning for Multi-Agent Systems with ACO and GA. *2024 32nd International Conference on Electrical Engineering (ICEE)*, 1-6. https://doi.org/10.1109/ICEE63041.2024.10667967

75. Majumder, C. G., Kumar, L. R., & Philip, N. K. (2016). Bee foraging inspired multi-agent optimal motion planning analysis in a simulated-mobile environment. *2016 Indian Control Conference (ICC)*, 39-46. https://doi.org/10.1109/INDIANCC.2016.7441103

76. Lamini, C., Benhlima, S., & Elbekri, A. (2018). Genetic Algorithm Based Approach for Autonomous Mobile Robot Path Planning. *Procedia Computer Science*, *127*, 180-189. https://doi.org/10.1016/j.procs.2018.01.113

77. Rath, A. K., Parhi, D. R., Das, H. C., Kumar, P. B., & Mahto, M. K. (2021). Design of a hybrid controller using genetic algorithm and neural network for path planning of a humanoid robot. *International Journal of Intelligent Unmanned Systems*, *9*(3), 169-177. https://doi.org/10.1108/IJIUS-10-2019-0059

78. Sood, M., & Panchal, V. K. (2024). Optimal path planning with hybrid firefly algorithm and cuckoo search optimisation. *International Journal of Advanced Intelligence Paradigms*, *27*(3/4), 223-248. https://doi.org/10.1504/IJAIP.2024.138565

79. Ma, X., Zhang, C., Yao, F., & Li, Z. (2022). Multi-Agent Task Allocation Based on Discrete DEPSO in Epidemic Scenarios. *IEEE Access*, *10*, 131181-131191. https://doi.org/10.1109/ACCESS.2022.3228918

80. Long, J., Wu, S., Han, X., Wang, Y., & Liu, L. (2023). Autonomous Task Planning Method for Multi-Satellite System Based on a Hybrid Genetic Algorithm. *Aerospace*, *10*(1), 70. https://doi.org/10.3390/aerospace10010070

81. Wang, Y., Yang, R. R., Xu, Y. X., Li, X., & Shi, J. L. (2021). Research on Multi-Agent Task Optimization and Scheduling Based on Improved Ant Colony Algorithm. *IOP Conference Series: Materials Science and Engineering*, *1043*(3), 032007. https://doi.org/10.1088/1757-899X/1043/3/032007

82. Wang, S., Liu, Y., Qiu, Y., Zhang, Q., Huo, F., Huangfu, Y., Yang, C., & Zhou, J. (2022). Cooperative Task Allocation for Multi-Robot Systems Based on Multi-Objective Ant Colony System. *IEEE Access*, *10*, 56375-56387. https://doi.org/10.1109/ACCESS.2022.3165198

83. ANNU PRIYA & SUDIP KUMAR SAHANA. (2022). An Optimized Modelling and Simulation on Task Scheduling for Multi-Processor System using Hybridized ACO-CVOA. *Journal of Information Science and Engineering*, *38*(5). https://doi.org/10.6688/JISE.202209_38(5).0001

84. Evaznia, N., & Ebrahimi, R. (2023). Providing a Solution for Optimal Management of Resources using the Multi-objective Crow Search Algorithm in Cloud Data Centers. *2023 9th International Conference on Web Research (ICWR)*, 179-184. https://doi.org/10.1109/ICWR57742.2023.10139192

85. Lopes Silva, M. A., De Souza, S. R., Freitas Souza, M. J., & De França Filho, M. F. (2018). Hybrid metaheuristics and multi-agent systems for solving optimization problems : A review of frameworks and a comparative analysis. *Applied Soft Computing*, *71*, 433-459. https://doi.org/10.1016/j.asoc.2018.06.050

86. Lopes Silva, M. A., De Souza, S. R., Freitas Souza, M. J., & Bazzan, A. L. C. (2019). A reinforcement learning-based multi-agent framework applied for solving routing and scheduling problems. *Expert Systems with Applications*, *131*, 148-171. https://doi.org/10.1016/j.eswa.2019.04.056

87. Kamali, S. R., Banirostam, T., Motameni, H., & Teshnehlab, M. (2023). An immune-based multi-agent system for flexible job shop scheduling problem in dynamic and multi-objective environments. *Engineering Applications of Artificial Intelligence*, *123*, 106317. https://doi.org/10.1016/j.engappai.2023.106317

88. Kaur, M., & Saini, S. (2021). A Review of Metaheuristic Techniques for Solving University Course Timetabling Problem. In V. Goar, M. Kuri, R. Kumar, & T. Senjyu (Éds.), *Advances in Information Communication Technology and Computing* (Vol. 135, p. 19-25). Springer Singapore. https://doi.org/10.1007/978-981-15-5421-6_3

89. Total Interaction Algorithm : A Metaheuristic in which Each Agent Interacts with All Other Agents. (2023). *International Journal of Intelligent Engineering and Systems*, *16*(1), Article 1. https://doi.org/10.22266/ijies2023.0228.20

90. Nouiri, M., Bekrar, A., Jemai, A., Niar, S., & Ammari, A. C. (2018). An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem. *Journal of Intelligent Manufacturing*, *29*(3), Article 3. https://doi.org/10.1007/s10845-015-1039-3

91. Dai, Q., Liu, J., & Wei, Q. (2019). Optimal Photovoltaic/Battery Energy Storage/Electric Vehicle Charging Station Design Based on Multi-Agent Particle Swarm Optimization Algorithm. *Sustainability*, *11*(7), Article 7. https://doi.org/10.3390/su11071973

92. Souidi, M. E. H., Haouassi, H., Ledmi, M., Maarouk, T. M., & Ledmi, A. (2023). A discrete particle swarm optimization coalition formation algorithm for multi-pursuer multi-evader game. *Journal of Intelligent & Fuzzy Systems*, *44*(1), Article 1. https://doi.org/10.3233/JIFS-221767

93. Khan, I., Maiti, M. K., & Maiti, M. (2017). Coordinating Particle Swarm Optimization, Ant Colony Optimization and K-Opt Algorithm for Traveling Salesman Problem. In D. Giri, R. N. Mohapatra, H. Begehr, & M. S. Obaidat (Éds.), *Mathematics and Computing* (Vol. 655, p. 103-119). Springer Singapore. https://doi.org/10.1007/978-981-10-4642-1_10

94. Biswas, S., Anavatti, S. G., & Garratt, M. A. (2017). Obstacle Avoidance for Multi-agent Path Planning Based on Vectorized Particle Swarm Optimization. In G. Leu, H. K. Singh, & S. Elsayed (Éds.), *Intelligent and Evolutionary Systems* (Vol. 8, p. 61-74). Springer International Publishing. https://doi.org/10.1007/978-3-319-49049-6_5

95. Shao, Z., Yan, F., Zhou, Z., & Zhu, X. (2019). Path Planning for Multi-UAV Formation Rendezvous Based on Distributed Cooperative Particle Swarm Optimization. *Applied Sciences*, *9*(13), Article 13. https://doi.org/10.3390/app9132621

96. Li, X., Wu, D., He, J., Bashir, M., & Liping, M. (2020). An Improved Method of Particle Swarm Optimization for Path Planning of Mobile Robot. *Journal of Control Science and Engineering*, *2020*, 1-12. https://doi.org/10.1155/2020/3857894

97. Wenhua Han, Ping Yang, Haixia Ren, & Jianpeng Sun. (2010). Comparison study of several kinds of inertia weights for PSO. *2010 IEEE International Conference on Progress in Informatics and Computing*, 280-284. https://doi.org/10.1109/PIC.2010.5687447

98. Kumar, N., Pal, N., Kumar, P., & Kumari, A. (2018). Impact of different inertia weight functions on particle swarm optimization algorithm to resolve economic load dispatch problems. *2018 4th International Conference on Recent Advances in Information Technology (RAIT)*, 1-5. https://doi.org/10.1109/RAIT.2018.8389065

99. Bansal, J. C., Singh, P. K., Saraswat, M., Verma, A., Jadon, S. S., & Abraham, A. (2011). Inertia Weight strategies in Particle Swarm Optimization. *2011 Third World Congress on Nature and Biologically Inspired Computing*, 633-640. https://doi.org/10.1109/NaBIC.2011.6089659

100. Taherkhani, M., & Safabakhsh, R. (2016). A novel stability-based adaptive inertia weight for particle swarm optimization. *Applied Soft Computing*, *38*, 281-295. https://doi.org/10.1016/j.asoc.2015.10.004

101. Yong-ling Zheng, Long-hua Ma, Li-yan Zhang, & Ji-xin Qian. (2003). Empirical study of particle swarm optimizer with an increasing inertia weight. *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, 221-226. https://doi.org/10.1109/CEC.2003.1299578

102. Zhu, X., & Wang, H. (2018). *A new inertia weight control strategy for particle swarm optimization*. 040095. https://doi.org/10.1063/1.5033759

103. Jain, M., Saihjpal, V., Singh, N., & Singh, S. B. (2022). An Overview of Variants and Advancements of PSO Algorithm. *Applied Sciences*, *12*(17), 8392. https://doi.org/10.3390/app12178392

104. Marini, F., & Walczak, B. (2015). Particle swarm optimization (PSO). A tutorial. *Chemometrics and Intelligent Laboratory Systems*, *149*, 153-165. https://doi.org/10.1016/j.chemolab.2015.08.020

106. Malik, R. F., Rahman, T. A., Hashim, S. Z. M., & Ngah, R. (s. d.). *New Particle Swarm Optimizer with Sigmoid Increasing Inertia Weight*.

107. Tisue, S., & Wilensky, U. (s. d.). *NetLogo : A Simple Environment for Modeling Complexity*.

108. Meraihi, Y., Gabis, A. B., Ramdane-Cherif, A., & Acheli, D. (2021). A comprehensive survey of Crow Search Algorithm and its applications. *Artificial Intelligence Review*, *54*(4), 2669-2716. https://doi.org/10.1007/s10462-020-09911-9

109. Sheta, A., Braik, M., Al-Hiary, H., & Mirjalili, S. (2023). Improved versions of crow search algorithm for solving global numerical optimization problems. *Applied Intelligence*, *53*(22), 26840-26884. https://doi.org/10.1007/s10489-023-04732-z

110. Mandala, J., & Chandra Sekhara Rao, M. V. P. (2019). Privacy preservation of data using crow search with adaptive awareness probability. *Journal of Information Security and Applications*, *44*, 157-169. https://doi.org/10.1016/j.jisa.2018.12.005

111. Cuevas, E., Barocio Espejo, E., & Conde Enríquez, A. (2019). A Modified Crow Search Algorithm with Applications to Power System Problems. In E. Cuevas, E. Barocio Espejo, & A. Conde Enríquez, *Metaheuristics Algorithms in Power Systems* (Vol. 822, p. 137-166). Springer International Publishing. https://doi.org/10.1007/978-3-030-11593-7_6

112. Ouadfel, S., & Abd Elaziz, M. (2020). Enhanced Crow Search Algorithm for Feature Selection. *Expert Systems with Applications*, *159*, 113572. https://doi.org/10.1016/j.eswa.2020.113572

113. Islam, J., Vasant, P. M., Negash, B. M., & Watada, J. (2019). A modified crow search algorithm with niching technique for numerical optimization. *2019 IEEE Student Conference on Research and Development (SCOReD)*, 170-175. https://doi.org/10.1109/scored.2019.8896291

114. Bhullar, A. K., Kaur, R., & Sondhi, S. (2022). Optimization of Fractional Order Controllers for AVR System Using Distance and Levy-Flight Based Crow Search Algorithm. *IETE Journal of Research*, *68*(5), 3900-3917. https://doi.org/10.1080/03772063.2020.1782779

115. Necira, A., Naimi, D., Salhi, A., Salhi, S., & Menani, S. (2022). Dynamic crow search algorithm based on adaptive parameters for large-scale global optimization. *Evolutionary Intelligence*, *15*(3), 2153-2169. https://doi.org/10.1007/s12065-021-00628-4

116. Priyadarshi, R., & Kumar, R. R. (2025). Evolution of Swarm Intelligence : A Systematic Review of Particle Swarm and Ant Colony Optimization Approaches in Modern Research. *Archives of Computational Methods in Engineering*. https://doi.org/10.1007/s11831-025-10247-2

117. Laassami, F., Souidi, M. E. H., & Ledmi, A. (2024). IB-PSO : An inversed Butterworth particle swarm optimisation algorithm for multi-agent path planning. *International Journal of Systems, Control and Communications*, *15*(4), 387-410. https://doi.org/10.1504/IJSCC.2024.143853

118. Kiani, F., Seyyedabbasi, A., Nematzadeh, S., Candan, F., Çevik, T., Anka, F. A., Randazzo, G., Lanza, S., & Muzirafuti, A. (2022). Adaptive Metaheuristic-Based Methods for Autonomous Robot Path Planning : Sustainable Agricultural Applications. *Applied Sciences*, *12*(3), 943. https://doi.org/10.3390/app12030943

119. Xin, J., Zhong, J., Yang, F., Cui, Y., & Sheng, J. (2019). An Improved Genetic Algorithm for Path-Planning of Unmanned Surface Vehicle. *Sensors*, *19*(11), 2640. https://doi.org/10.3390/s19112640

120. Song, H., Jia, M., Lian, Y., Fan, Y., & Liang, K. (s. d.). *UAV Path Planning Based on an Improved Ant Colony Algorithm*.

121. Meng, F., Chen, L., Ma, H., Wang, J., & Meng, M. Q.-H. (2024). NR-RRT : Neural Risk-Aware Near-Optimal Path Planning in Uncertain Nonconvex Environments. *IEEE Transactions on Automation Science and Engineering*, *21*(1), 135-146. https://doi.org/10.1109/TASE.2022.3215562

122. Osei-kwakye, J., Han, F., Amponsah, A. A., Ling, Q.-H., & Abeo, T. A. (2023). A diversity enhanced hybrid particle swarm optimization and crow search algorithm for feature selection. *Applied Intelligence*, *53*(17), 20535-20560. https://doi.org/10.1007/s10489-023-04519-2

123. Directorate of Regions and Governorates Affairs, Ministry of Youth & Sport, Iraq., Yousif, M., Salim, A., & K. Jummar, W. (2021). A Robotic Path Planning by Using Crow Swarm Optimization Algorithm. *International Journal of Mathematical Sciences and Computing*, *7*(1), 20-25. https://doi.org/10.5815/ijmsc.2021.01.03

124. Abaas, T., & Shabeeb, A. (2022). Obstacle Avoidance and Path Planning of a Wheeled Mobile Robot Using Hybrid Algorithm. *Engineering and Technology Journal*, *40*(12), 1-12. https://doi.org/10.30684/etj.2022.132929.1154

125. Sharma, I., Kumar, V., & Sharma, S. (2022). A Comprehensive Survey on Grey Wolf Optimization. *Recent Advances in Computer Science and Communications*, *15*(3), e180322186729. https://doi.org/10.2174/2666255813999201007165454

126. Braik, M., Al-Zoubi, H., Ryalat, M., Sheta, A., & Alzubi, O. (2023). Memory based hybrid crow search algorithm for solving numerical and constrained global optimization problems. *Artificial Intelligence Review*, *56*(1), 27-99. https://doi.org/10.1007/s10462-022-10164-x

127. Subbaraj, S., Thiyagarajan, R., & Rengaraj, M. (2023). A smart fog computing based real-time secure resource allocation and scheduling strategy using multi-objective crow search algorithm. *Journal of Ambient Intelligence and Humanized Computing*, *14*(2), 1003-1015. https://doi.org/10.1007/s12652-021-03354-y

128. Chen, X., & Li, Y. (2007). A Modified PSO Structure Resulting in High Exploration Ability With Convergence Guaranteed. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, *37*(5), 1271-1289. https://doi.org/10.1109/TSMCB.2007.897922

129. Jia, Y.-H., Qiu, J., Ma, Z.-Z., & Li, F.-F. (2021). A Novel Crow Swarm Optimization Algorithm (CSO) Coupling Particle Swarm Optimization (PSO) and Crow Search Algorithm (CSA). *Computational Intelligence and Neuroscience*, *2021*(1), 6686826. https://doi.org/10.1155/2021/6686826

130. Tam, C., Bucknall, R., & Greig, A. (2009). Review of Collision Avoidance and Path Planning Methods for Ships in Close Range Encounters. *Journal of Navigation*, *62*(3), 455-476. https://doi.org/10.1017/S0373463308005134

131. Barrera, J., Álvarez Bajo, O., Flores, J. J., & Coello Coello, C. A. (2016). Limiting the Velocity in the Particle Swarm Optimization Algorithm. *Computación y Sistemas*, *20*(4). https://doi.org/10.13053/cys-20-4-2505

132. Adewumi, A. O., & Arasomwan, A. M. (2015). Improved Particle Swarm Optimizer with Dynamically Adjusted Search Space and Velocity Limits for Global Optimization. *International Journal on Artificial Intelligence Tools*, *24*(05), 1550017. https://doi.org/10.1142/S0218213015500177