UNIVERSITÀ DEGLI STUDI FIRENZE

PhD Program in Smart Computing

CYCLE XXXIV

Coordination of swarms of robots in target search: from bio-inspired heuristics to hyper-heuristics

Academic Discipline (SSD) ING-INF/05

PhD Candidate

Manilo Monaco

Manilo Monaco

Advisor

Prof. Gigliola Vaglini

[illegible]

Prof. Mario G.C.A. Cimino

[illegible]

Head of PhD Program

Prof. Stefano Berretti

[illegible]

Evaluation Committee

Prof. Giovanna Castellano, *University of Bari Aldo Moro*

Prof. Adrian Groza, *Technical University of Cluj-Napoca*

Years 2018/2021

To Sofia Elena,

who fills my life with light.

Manilo Monaco

Acknowledgments

First and foremost, I would like to dedicate a special thanks to my wife Angela for having encouraged me to open the horizon to the possibility of undertaking this adventure when I had never thought about it. Then, I would like to thank my parents for always being by my side, even when we were not in the same place, as if space and time did not exist. I thank my brother Mario, who always helped me with his advice and never made me feel alone.

My deep gratitude goes to my advisors, Prof. Gigliola Vaglini and Prof. Mario G. C. A. Cimino, who believed in me when neither I believed in myself and gave me the opportunity to live this amazing experience. I would like to thank Prof. Mario G. C. A. Cimino, who contributed to light in me the Love for research, because *"it is Love that moves the sun and the other stars..."*.

I would also like to express my appreciation to the reviewers of my thesis, Prof. Giovanna Castellano and Prof. Adrian Groza, for helping to improve this work with their useful and constructive suggestions.

Finally, yet importantly, I would like to thank all the colleagues, Professors, researchers, students, collaborators, and friends, with whom I shared part of this wonderful journey. I will always keep in my heart the very long afternoons spent in the Department with Federico A. Galatolo discussing about our research, trying to connect physics and metaphysics.

Thank you all for the time you dedicated to me.

Abstract

In recent years, the problem of *target search* has received much attention in the research community due to the wide range of applications domains involved, such as environmental monitoring, precision agriculture, surveillance, or search and rescue. Essentially, it concerns the search for stationary or dynamic targets in unstructured environments, aiming to minimize the overall discovery time. In order to tackle this problem, solutions based on collective search are currently of great interest in robotics. However, coordinating a multi-robot system is a challenging problem, particularly in unstructured areas, as for example hazardous and post-disaster scenarios where direct communication is limited. *Swarm robotics* is a new and disruptive research field that studies how to manage and coordinate large groups (swarms) of mostly simple physical robots, getting inspiration from *swarm intelligence* to model the behavior of the robots.

For centuries, the concept of intelligence has been linked exclusively to human beings. However, a simple observation of nature shows that other creatures can also develop behaviors that are sophisticated enough to be considered intelligent. The mysterious dance of honeybees to communicate the location of promising food sources, the amazing floating shapes drawn in the sky by a flock of birds while foraging, the creation of impressive cathedral mounds by termites, the trail followed by ants to quickly reach the nest from a food source are all good examples of complex collective behaviors, unknown to individual members of the swarm. These sophisticated collective behaviors emerge from a relatively small set of rather simple rules, where single individuals exploit only low-level local interactions with each other and with the environment to gain *decentralized control* and *self-organization*. For example, in ant societies, a key factor of self-organization is the indirect communication between individuals through changes in the environment, a process known as *stigmergy*. Specifically, at the beginning the ants search for new food sources moving randomly. However, when an ant finds a potential food source, it takes a piece and returns to the nest, leaving pheromone trails on the way back. Other ants, while perceiving the pheromones, follow the trail until the food source and come back to the nest, releasing themselves new pheromones, thus reinforcing the specific route. On the other hand, these pheromone trails evaporate over time, reducing their attractive strength. Obviously, shorter paths are less affected by this evaporation process in short term, so they are more likely to be eventually visited more frequently than the longer ones. In this way, nature provide a solution to the problem of finding the shortest path between two points: the ant colony and the food source. This and other biological mechanisms have been the inspiration for efficient optimization methods (*bio-inspired heuristics*). In the context of swarm robotics, a virtual representation of the pheromone can be used to steer the swarms towards the most favorable areas of an application scenario, e.g. the regions with the highest probability of the presence of targets.

The main drawbacks of bio-inspired heuristics are related to the selection of the most suitable algorithm for the specific scenario and the parametrization costs to adapt it to new type of missions. In fact, the hypothesis space of a bio-heuristic, i.e. the space in which to search for a good algorithm configuration, is constrained by models of biological species. In order to generate more adaptable logics, in this thesis it is proposed a novel design approach based on *hyper-heuristics*. For a given application domain, hyper-heuristics aim to provide more generalized solutions to optimization problems, rather than deriving techniques that perform well for just a few problem instances. In order to achieve this result, they either select or generate low-level heuristics, which are used to solve the problem at hand. In this thesis, two fundamental components are considered as constructive low-level heuristics for building decentralized and self-organized robot swarm coordination logics, i.e. stigmergy and flocking. Moreover, *Differential Evolution* is used to optimize the aggregation and tuning of these modular heuristics over realistic real-world scenarios. The experimental results acquired from extensive simulations are promising and show the convenience of using hyper-heuristics as a novel design methodology compared to simple bio-inspired heuristics.

Contents

List of Figures

List of Tables

Chapter 1

1 Introduction

Optimization has become increasingly important in a wide range of applications, from engineering design and business planning to data mining and machine learning. Minimizing energy consumption, costs, waste, travel time, and environmental impact, as well as maximizing profit, outputs, performance, efficiency, and sustainability, are all real-world problems that are concerned with optimization. Clearly, any application is subject to a limited amount of resources of many kinds, such as time and money, and the goal becomes to find optimal solutions to various design problems, which have a broad range of complex constraints (Yang & Deb, 2014). Mathematical optimization or mathematical programming can be used to properly formulate and model any kind of optimization problem. The construction of the cost function or objective function represents the most important part of the model and, usually, several design options are evaluated and compared. After formulating the optimization problem, the goal is to find the optimal solution or a set of optimal solutions, in order to minimize or maximize the objective function. Most real-world problems are nonlinear in terms of both objective function and constraints, and only sophisticated optimization algorithms can deal with such problems. Moreover, the evaluations of objective functions can be time-consuming, particularly for problems related to data mining and machine learning.

Despite the increasing computational power of modern computers, brute force approaches to finding optimal solutions are not feasible. Therefore, the use of efficient algorithms is critical in almost all applications. Algorithms for efficient optimization include both traditional techniques, such as gradient-based algorithms, and evolutionary approaches, such as genetic algorithms (GA) and evolutionary algorithms (EA). In recent years, this list has grown with interesting new algorithms inspired by nature and swarm intelligence, including ant colony optimization (ACO) (Dorigo, Birattari, & Stützle, 2006), particle swarm optimization (PSO) (Kennedy & Eberhart, 1995), the firefly algorithm (FA) (Yang, 2009), cuckoo search (CS) (Yang & Deb, 2014), and many others.

Among the applications where optimization can play an important role, there are multi-robot systems. Multi-robot systems have a great potential in a variety of critical missions, such as surveillance, environmental monitoring, search and rescue. A relevant issue is the coordination of swarms for an increasing number of robots in order to achieve pre-defined global objectives. Regarding this issue, today biological swarms are much more effective with respect to the artificial counterpart. Considering that in complex or open environments robots cannot exploit static information on layout and targets locations, their coordination is fundamental for an efficient target discovery: the key problem is how to specify the individual robot behavior for an effective interaction at the swarm level.

A target search mission is usually organized into *environmental exploration*, i.e., to search targets, and *targets resolution*, i.e., to collect sufficient target information. In the literature of biological models, a fundamental strategy of exploration is carried by ants. While on the move, ants deposit in the terrain a chemical substance called pheromone. As an example, in Ant Colony Optimization (ACO) (Dorigo, Birattari, & Stützle, 2006) artificial ants release pheromones while exploring the environment to temporarily mark the visited places. Different types of pheromones, related to different meanings, enable ants to make different decisions. Digital versions of pheromones are commonly used to orientate robots' exploration (Alfeo, et al., Swarm coordination of mini-UAVs for target search using imperfect sensors, 2018). Robots move according to the sensed pheromone; specifically, a robot begins to coordinate the resolution of the target detected during exploration, by attracting other robots towards the position indicated by the pheromone. When the recruited robots are sufficient in number, they perform the target resolution. In the literature, three major bio-inspired meta-heuristics are considered for recruitment: (i) the Firefly-based Team Strategy (FTS) (Yang, 2009), an algorithm derived from swarms of fireflies; (ii) Particle Swarm Optimization (PSO) (Kennedy & Eberhart, 1995), modelled from schools of fishes and flocks of birds; (iii) Artificial Bee Colony (ABC) (Karaboga & Akay, 2009), based on behavior of honey bees. The problem of coordinating swarms of robots has received attention by many research areas, due to its potential impact on real-world applications. Swarm coordination strategies can be divided into two categories. *Explicit coordination* is based on the direct exchange of messages between robots, according to a detailed orchestration among swarm members (Senanayake, et al., 2016). This many-to-many communication strategy causes a poor performance of large swarms of robots. In contrast, with implicit coordination each robot makes simple behavioral decisions, based on information gathered through its indirect perception mediated by an environmental mechanism. Although the single piece of information obtained by perception is not completely accurate, the robustness of the swarm can be sensibly improved by the collective contribution (Senanayake, et al., 2016).

1.1 Swarm intelligence in natural systems

For centuries, the notion of intelligence has been linked exclusively to human beings, without considering the possibility that other natural creatures could also develop sophisticated behaviors. However, this assumption is contradicted by multiple examples observed in nature, such as the collective behavior of colonies of many social animals (ants, termites, birds, bees, and fireflies). These swarms exhibit global behaviors that go far beyond the simple aggregations of individual behaviors. The mysterious dance of honeybees to communicate the location of promising food sources, the amazing floating shapes drawn in the sky by a flock of birds while foraging, the creation of impressive cathedral mounds by termites, the trail followed by ants to quickly reach the nest from a food source are all good examples of complex collective behaviors, unknown to individual members of the swarm. Pierre-Paul Grassé in 1959 first investigated the nest-building behavior of termite colonies. Termites occasionally pick up a ball of soil that is then covered with pheromone and dropped at random. If there is already a pheromone covered mud ball nearby, there is a greater likelihood that a second one will be placed next to it. As this stack increases in size and the corresponding amount of pheromone increases, the chances of more pheromones being added increase, leading to the result of creating a structured termite mound complete with arches and chambers. This action of cascading mud balls until a complete termite mound is built is an example of a positive feedback loop. Although a colony is composed of individual termites apparently pursuing their own interests, the shape and state of their local environment affects their actions and allows coordination at the swarm level (Howden, 2013).

In nature swarms can have very different sizes, from a few individuals that live in a restricted area to wide colonies spread over a large area and made up of thousands or millions of individuals. The fundamental feature of the swarms is that they are decentralized, in other words, the members of the swarm are not led by a leader to complete the prefixed activities. In general, a swarm is composed of a set of homogeneous (or quasi-homogeneous) individuals that move in a non-synchronized way. Each member has limited intelligence and is not able to achieve the goals of the swarm alone. It has been shown that the individuals are able to produce sophisticated collective behaviors without having any global understanding of the swarm. Each member has no information about the overall status of the swarm. On the other hand, interactions between individuals occur only at the local level. A paradigmatic example is represented by the flocking behavior of birds: each bird is aware only of its local neighbors, but despite this the flock is capable of migrating thousands of kilometers toward a destination, following a common orientation in its movement. Another example is fish schooling: individual fish keep track only of their close neighbors by following with their eyes some marks on the bodies of other fishes and ignore the overall evolution of the swarm. However, the school as a whole can change its shape and direction of movement at amazing speed, and without collisions between its members.

It is hard to understand how such complex behaviors can result from swarms of simple individuals with limited cognitive abilities. Surprisingly, these sophisticated collective behaviors emerge from a relatively small set of rather simple rules, where single individuals exploit only low-level local interactions with each other and with the environment to gain decentralized control and self-organization. The self-organization is based on the combination of four fundamental rules: positive feedback, negative feedback, randomness, and multiple interactions (Bonabeau, Theraulaz, Deneubourg, Aron, & Camazine, 1997). A key factor is the indirect communication between individuals through changes in the environment, a process known as stigmergy. Stigmergy is a mechanism of indirect coordination, mediated by the environment, by which the swarm is able to self-organize, generating intelligent behavioral patterns without any form of planning, control, or even direct communication between agents. Thanks to this indirect communication mechanism, extremely simple agents, although with low memory and intelligence, and without awareness of each other, achieve an efficient level of collaboration. For example, ants communicate by releasing pheromones into the environment. Specifically, at the beginning the ants search for new food sources moving randomly. However, when an ant finds a potential food source, it takes a piece and returns to the nest, leaving pheromone trails on the way back. Other ants, while perceiving the pheromones, follow the trail until the food source and come back to the nest, releasing themselves new pheromones, thus reinforcing the specific route. On the other hand, these pheromone trails evaporate over time, reducing their attractive strength. Obviously, shorter paths are less affected by this evaporation process in short term, so they are more likely to be eventually visited more frequently than the longer ones. In this way, nature provide a solution to the problem of finding the shortest path between two points: the ant colony and the food source. This mechanism has inspired the well-known Ant Colony Optimization method (Dorigo, Birattari, & Stützle, 2006).

Another important property sometimes observed in natural swarms is the eusociality, which represents the highest level of organization in social animals. A distinctive feature of eusociality is an efficient division of labor among the members of the swarm, where several specialized groups can be found. In many cases, this division of labor is characterized by the existence of subgroups of individuals who have lost a behavioral skill that is instead a property of other members of the swarm. This behavioral specialization makes swarms more efficient but also more complex, resulting in the emergence of different subgroups exhibiting different behavioral roles, usually labeled as castes. Examples of eusociality in animal swarms include most species of ants and termites, as well as many species of bees and wasps.

1.2 Computational swarm intelligence

The expression swarm intelligence was first used by Gerardo Beni and Jing Wang in 1989 in the context of a project on cellular robotic systems (Beni & Wang, 1989). Swarm intelligence is a branch of artificial intelligence concerned with the collective behavior of decentralized and self-organized systems. Swarm intelligence systems typically consist of a population of relatively simple agents that interact locally with each other and with the environment, as well as natural swarms actually do. Indeed, several biological systems observed in nature show very similar features (Yang, 2016). Since its introduction, swarm intelligence has increasingly attracted the attention of the scientific community in several fields, from computer science to artificial intelligence, from engineering to economics, and many others. As a result, a great number of techniques based on swarm intelligence have been developed in order to solve complex problems, such as optimization problems.

Practically all swarm behaviors observed in nature have been suitably modeled to develop different swarm intelligence methods. For example, optimization methods were proposed in the 1990s that are now well known, such as Ant Colony Optimization (Dorigo, Birattari, & Stützle, 2006), Particle Swarm Optimization (Kennedy & Eberhart, 1995), and Differential Evolution (Storn & Price, 1997). Methods based on swarm intelligence are designed to exploit mainly the local interactions among the agents of the swarm, rather than focusing on the structure of the agents as in the traditional approaches in artificial intelligence. Typically, the agents have poor intelligence and are driven by a small set of rules. However, the swarm as a whole is able to complete difficult tasks through strong cooperation based on local interactions and division of labor, leading to the emergence of complex behaviors never observed in a single agent. Ant algorithms represent one of the first applications of the computational stigmergy (Dorigo, Bonabeau, & Theraulaz, 2000). These approaches have been used in solving several computational problems, such as the traveling salesman problem, scheduling and routing problems, and problems related to structural engineering and digital image processing (Mohan & Baskaran, 2012).

1.3 Swarm robotics

In recent years, the research community has become increasingly interested in swarm intelligence because of its potential applications in multiple fields. The decentralized coordination of groups of very simple self-organized robots is one of its most important applications. In this way, complex and expensive robots can be replaced by simple and inexpensive drones that can potentially perform highly sophisticated tasks (Arvin, Murray, Shi, Zhang, & Yue, 2014). This interesting research area is generally referred to as swarm robotics. Swarm robotics can be briefly defined as the discipline that studies how to handle and coordinate large groups (swarms) of relatively simple autonomous robots by using local rules and interactions. The design of physical components (sensors and actuators) and controlling behaviors of the robots is also of interest for this research area (Sauter, Matthews, Parunak, & Brueckner, 2007).

The main challenge of swarm robotics applications is that the swarm of robots must be designed in such a way that local interactions among agents and between agents and the environment led to the emergence of the desired collective behavior, according to the concepts of swarm intelligence. However, in order to avoid falling into other approaches of coordinating multi-robot systems, the design of robotic swarms must also require the following criteria:

- The robots of the swarm are usually small and low cost, so that they can be manufactured and deployed in large numbers.

- The robots must be autonomous and able to sense and work in a real-world environment.

- The robots of the swarm should be homogeneous. Heterogeneous subgroups of robots are allowed only in limited number.

- The robots are very simple and cannot solve the problem individually, or they do so by exhibiting very poor performance. As a result, they need to cooperate in order to solve the problem more efficiently.

- Only local sensing and communication capabilities are allowed to the robots of the swarm. The local communication ensures that the swarm is scalable and robust and can overcome failures of individual members.

- The members of the swarm are usually controlled by very simple behavioral rules, performed at the individual level, according to the cooperative behavior commonly observed in natural swarms. The interaction of these rules produces a wide set of complex collective behaviors.
- The robotic swarm is decentralized, distributed, and self-organized. As a result, it shows high efficiency, parallelism, scalability, and robustness.

1.3.1 Advantages and limitations

Most of the advantages manifested by robotic swarm systems are similar to those observed in natural swarms. A single robot that has to perform a difficult task requires a sophisticated design and configuration, involving a great number of different structural components (mechanical, electronics, optical, etc.) and several control modules for sensors and actuators. Typically, many organizations and institutions do not have sufficient budget to deal with the high costs of building, testing, operating, enhancing, and maintaining these types of robots and their components. Moreover, a single expensive robot is subject to the issue known as "single point of failure". In other words, it becomes highly vulnerable and prone to errors, since even the smallest component can affect the overall performance of the whole robotic structure. Vice versa, a swarm of simple and low-cost robots can accomplish similar tasks via strong cooperation among its members. By deploying a large number of robots in different areas, it is possible to increase the exploration capability of the swarm compared to a single robot, also exploiting the high parallelism. Furthermore, the loss of an individual robot, or even some of them, does not affect, or affects only minimally, the overall performance of the swarm, so making it less susceptible to errors and accidents.

A representative example is related to one of the worst nuclear disasters in recent history, which is the Fukushima disaster. In 2011, a tsunami caused by a strong earthquake reached the coast of Fukushima prefecture and the nuclear plant located nearby. This disaster clearly highlighted the limitations of robotic technologies. In fact, the highly radioactive environment following the disaster proved to be too extreme for many robotic units deployed at the nuclear plant. Complex machines that were considered the cutting edge of technology at the time were suddenly inactivated by radiation. Sophisticated and expensive autonomous robots were disconnected or shut down, while others were trapped by fallen structures and deformed obstacles in unexpected places. In December 2016, more than 5 years after the disaster, a new powerful and sophisticated autonomous robot developed by Toshiba was deployed at the plant. The robot was equipped with sensors to measure radiation and temperature, and cameras to get the better viewing angles. Unfortunately, the robot was stuck by some blocks of molten metal found on its way, after entering unit 2 of the plant. The mission aborted after 2 hours, even though it had been planned to operate for 10 hours. After this failed attempt, Toshiba returned to the plant several times with new, improved and smaller robots operating cooperatively. Finally, they were able to complete the mission, providing very useful information about the state of the plant, such as radiation and temperature measurements, and without disturbing the surrounding environment. The robots were also able to clean out small objects in some of the indoor paths.

The main advantages of robotic swarms over a single sophisticated robot are the following:

- *Improved performance thanks to parallelization*: the members of the swarm can perform different actions in different locations at the same time, because they work following each one its individual rules. In this way the swarm gains more flexibility and efficiency in carrying out complex tasks, as individual robots (or groups of them) can solve portions of a complex task independently.

- *Enabling new tasks*: multiple robots can complete tasks that are impossible or extremely difficult for a single robot (e.g., dynamic target tracking, cooperative environmental monitoring, autonomous surveillance of large areas).

- *Scalability*: the whole swarm does not need to be reprogrammed after adding new robots. Interactions between robots occur only at local level, so their total number within the system does not rise significantly although new individuals are added.

- *Distributed sensing and action*: the sensing range and the exploration capability of the whole swarm are greater than a single complex robot. These features make the swarm much more effective in multiple tasks, such as exploration and navigation (e.g., in disaster rescue missions), or environmental monitoring (e.g., early fire detection and tracking).

- *Robustness*: the failure of a single unit does not affect the completion of the given mission due to the decentralized and self-organized nature of the swarm. If one or a few units fail or stop the task, the swarm adapts to the change in population size by implicitly reallocating the task.

- *Cost-effectiveness*: the cost of a single sophisticated robot is dramatically higher than the cost of a simple robot of a swarm. Typically, robotic swarm units are designed to be very low-cost, so they can be manufactured in large volumes. Since these robots are almost identical, their components are usually highly interchangeable and maintenance costs are also very low. Moreover, the maintenance process is simpler thanks to the less expertise required to fix simple and homogeneous robotic units.

- *Energy efficiency*: simpler and smaller robots need smaller batteries and less energy power. As a result, the lifetime of the whole swarm increases.

There also exist other areas where groups of robots are used simultaneously to complete a common mission, such as multi-robot systems, multi-agent systems, and sensor networks. However, they generally do not have the properties of robotic swarms and cannot be considered as such. Table 1.1 summarizes the main features among these multi-individual systems.

Table 1.1 - Comparison between swarm robotics and other multi-robot systems

	SWARM ROBOTICS	**MULTIROBOT SYSTEM**	**SENSOR NETWORK**	**MULTIAGENT SYSTEM**
POPULATION SIZE	Variation in great range	Small	Fixed	In a small range
CONTROL	Decentralized and autonomous	Centralized or remote	Centralized or remote	Centralized or hierarchical or network
HOMOGENEITY	Homogeneous	Usually heterogeneous	Homogeneous	Homogeneous or heterogeneous
FLEXIBILITY	High	Low	Low	Medium
SCALABILITY	High	Low	Medium	Medium
ENVIRONMENT	Unknown	Known or unknown	Known	Known
MOTION	Yes	Yes	No	Rare
TYPICAL APPLICATION	Post disaster relief Military applications Dangerous applications	Transportation Sensing Robot football	Surveillance Medical care Environmental protection	Net resources management Distributed control

Tan and Zheng (Tan & Zheng, 2013) highlight that the main differences between robotic swarms and other multi-robot systems are related to population size, type of control, homogeneity of the units, flexibility, and scalability. Regarding the population size, it can vary widely in robotic swarms, differently from multi-robot and multi-agent systems where it is kept small, while it is fixed for sensor networks. Usually, robots of multi-robot and multi-agent systems are heterogeneous, having specialized roles in the missions in which they are involved. As a result, they are poorly suited to general-purpose problem solving because they lack flexibility and scalability, while they are able to achieve high performance on specific tasks. For example, the teams of robots that participate in popular competitions such as "Robocup" cannot be considered robotic swarms, exactly because the different robots in the team have very specialized roles. Thanks to flexibility, robotic swarms can perform different missions using the same hardware and small changes in the software. They are able to adapt to new environments, changing the mission completion strategy: in order to achieve this, it is not required to reprogram the whole swarm, but is enough to perform just incremental changes aimed at improving the current strategy, typically via machine learning approaches. Finally, local sensing and communication features allow the swarm to be scalable. Indeed, the many-to-many communication represents a limit for the scalability of the system, since by adding new units the communication costs grow exponentially. Thanks to the manifest advantages that characterize the robotic swarms, in the last twenty years many relevant projects in this research area have arisen.

However, the robotic swarms have their own limitations and drawbacks too. The main ones are the following:

- *Potential collisions*: since the robots in the swarm only communicate locally and are not aware of other robots at a global level, they can sometimes obstruct each other or collide in unexpected ways.

- *Uncertainty*: swarm coordination requires each individual to know the locations of the other robots and what they are doing. However, local communication does not allow robots to have complete certainty about the status of the other members of the swarm. As a result, sometimes this can cause conflicts between robots that compete rather than cooperate.

- *Lack of specialization*: since the swarm is homogeneous, highly specialized tasks can be completed with difficulty. Vice versa, a single sophisticated and customized robot might outperform the swarm.

- *Hard to design swarm behavior*: in swarm robotics, global behavioral patterns emerge from local interactions between individuals and between individuals and the environment, thus it is challenging to determine the implicit rules that lead to the best performance in completing a task.

1.3.2 Target search problem

The problem of collective search requires a trade-off between maximizing the area to be covered and searching accurately. The study of distributed algorithms applied to the problem of collective search is currently a solution of great interest in robotics. In this context, the design aims to allow robots, without centralized coordination, to use local information to perform search and rescue operations (Countryman, et al., 2015). The problem of coordinating a team of robots for exploration is a challenging task, particularly in unstructured environments, such as hazardous and post-disaster scenarios where direct communication is limited. The ant foraging model has inspired the design of a possible exploration algorithm. The absence of governing hierarchy, the self-organizations of robots, and the indirect communication represent the crucial aspects of this kind of approach, where individual robots play the key role.

In nature, ants have evolved over a long period of time and exhibit very intelligent behaviors that are well suited to tackle complex tasks. Social insects communicate through pheromones, that allow them to perform many social functions, such as food gathering, aggregation, mating, recognition, and alarm propagation to other members of the colony. Certain swarm intelligence algorithms, such as ACO, use pheromone trails as a way of (indirect) communication between agents.

In pheromone-based coordination, robots transfer information using the environment as a medium: each robot, depending on the type of information it wants to indirectly communicate, deposits marks in the environment to send different types of signals. The shared memory provided by the aggregation of trails in the environment allows simple robots with no memory to easily coordinate, even without the need for self-awareness of other agents. These algorithms are based on memoryless agents that exhibit very simple individual behaviors and are entirely decentralized. Agents can only mark and move based on their local perceptions, and thus communicate only by marking the environment. In the robotics field, thanks to the availability of several types of sensors, the information can be encoded through a range of environmental markers, such as chemicals, heat sources, metals, and electronic tags (Kuyucu, Tanev, & Shimohara, 2012).

The recruitment problem is aimed at designing a low-cost coordination mechanism that is able to organize groups of robots at those sites where targets are detected. A single robot may not have enough resource capabilities to handle a target, so once it finds a target, it attracts other robots in order to organize a coalition that works cooperatively to process or disarm the target. Since the detection of a target can occur at any time during the exploration of the environment, the recruitment task is carried out in real time and possibly at different sites in the search environment.

For this purpose, when a fast reaction is expected and countermeasures need to be taken, direct communication may be more effective and information about found targets is shared using wireless communication. In this case, each robot is assumed to have transmitters and receivers and can send packets to other robots in its wireless range.

In this problem a key issue is how to avoid deadlock, i.e., the situation where robots wait a long time for others to proceed in the target processing. In strictly collaborative tasks, these problems are particularly relevant, since each robot has information about the environment only locally and partially, and the robots must work collectively and adaptively to disarm hazardous targets. The greedy approach is the most common one, where a detected target is instantly assigned to the robots, without taking into account future events. A more flexible strategy is possible, where the robots can react to new future events, possibly changing the decisions made. However, each robot must make individual decisions that could lead to stop taking into account from requests for help. In missions of this type, for example, while reaching a target it is possible to detect another target or receive another request, and then it is possible to change decisions in order to move in a more convenient way from the robot's point of view. So, at each time step, the robots will be able to make the best decision based on their positions and conditions, in response to the requests for help received, trying at the same time to balance the two tasks.

Generally, there are two communication mechanisms that are suitable for tackling the problem. The first communication mechanism is one-hop, where coordinating robots send packets only to the direct neighborhood (i.e., robots within the communication range) and no information forwarding can occur. Following this approach, different bio-inspired algorithms can be compared. The other mechanism involves multi-hop communication, where information can be propagated among team members. In this case, for example, an ant-based protocol can be used.

1.4 Hyper-heuristics

Bio-inspired meta-heuristic algorithms succeed in efficiently addressing many NP-hard combinatorial optimization problems and more generally constrained nonlinear optimization problems (Yang, 2020), and so they have recently become the cutting edge of current research. Each of these algorithms is based on a specific successful mechanism of a biological phenomenon found in nature, aiming to achieve the survival of the fittest individual in a dynamically changing environment. In nature, examples of collective behavior are numerous. These are primarily based on the direct or indirect exchange of information about the environment among swarm members. The overall result of collective behavior is difficult to predict, although the rules governing interactions at the local level are usually simple to describe. However, swarms in nature are able to solve complex problems that are crucial for their survival through simple collaboration.

Even more recently, known benchmark sets have been used to derive techniques that improve the results obtained from existing techniques in the field of research on solving combinatorial optimization problems such as rostering, vehicle routing, and scheduling problems. These benchmark sets are made publicly available to compare the performance of different techniques in solving this class of problems. The results of such research have shown that a specific technique may produce the best results on one or two instances of the problem, but quite often performs poorly on other instances of the problem.

The research field of hyper-heuristics arises from an attempt to provide more generalized solutions to combinatorial optimization problems, rather than deriving techniques that provide good results only for some instances of the problem for a certain reference domain, showing good performance on a set of problems. Hyper-heuristics achieve this by working in the *heuristic space* rather than in the *solution space* (Burke, et al., 2003). As such, hyper-heuristics select or generate *low-level heuristics*, which in turn are used to solve the problem at hand. In order to select or generate low-level heuristics, several techniques such as evolutionary programming, local search, and case-based reasoning are used (Pillay & Qu, 2018).

Hyper-heuristics either select low-level heuristics to construct/improve a solution or create new low-level heuristics. As such, low-level heuristics are categorized as *constructive* or *perturbative*. These heuristics are usually defined for a particular problem domain and hence are problem specific.

Constructive heuristics are usually used to create an initial solution to a problem, which in turn serves as a starting point for other optimization techniques such as simulated annealing or tabu search in solving the problem. In the domain of examination timetabling, for example, constructive heuristics are used to select the examination to schedule next based on a measure of the difficulty of scheduling it.

Perturbative heuristics are used to improve an existing initial solution created by a constructive heuristic or randomly created. Low-level perturbative heuristics have the same effect as a move operator in local search used to explore the neighborhood of a search point and therefore perform changes to the initial solution. The domain of the problem determines the type of perturbation carried out. In the case of examination timetabling, for example, perturbative heuristics can include deallocating an examination, allocating an examination, swapping examinations between timetable periods, and swapping rows in the timetable.

Since hyper-heuristics either select existing low-level heuristics or generate new low-level heuristics, and these heuristics can be constructive or perturbative, hyper-heuristics are classified as being selection constructive, selection perturbative, generation constructive or generation perturbative (Burke, et al., 2013).

Table 1.2 - Classification of Hyper-Heuristics according to two dimensions: (i) the source of feedback during learning, and (ii) the nature of heuristic search space

HYPER-HEURISTICS

Source of feedback during learning	**ON-LINE LEARNING / OFF-LINE LEARNING / NO LEARNING**			
Nature of the search space	**HEURISTIC SELECTION**		**HEURISTIC GENERATION**	
Low-level heuristics	**CONSTRUCTIVE**	**PERTURBATIVE**	**CONSTRUCTIVE**	**PERTURBATIVE**

Constructive selection hyper-heuristics select a low-level heuristic to be applied at each point of the solution construction. Techniques employed by hyper-heuristics to select the low-level construction heuristic include population-based methods, local search methods, case-based reasoning, adaptive methods, and hybrid approaches.

Perturbative selection hyper-heuristics select the low-level perturbative heuristic to be applied to each point of the solution improvement and can perform the search on a single point or on multiple points. In the first case, the hyper-heuristic includes two components, one for the selection heuristic to select a low-level perturbative heuristic, and another for the move acceptance to determine whether the move made by the selected low-level heuristic should be accepted or not. The selection heuristic and the move acceptance can use several techniques. Perturbative selection hyper-heuristics that perform multipoint search to select low-level heuristics use population-based methods to explore the heuristic space such as evolutionary algorithms. The population-based technique by its nature performs both selection heuristics and move acceptance, and thus hyper-heuristics do not contain separate components for these functions.

Generation constructive hyper-heuristics create new low-level constructive heuristics for the problem domain. The generated heuristics are used to create an initial solution, which is further optimized using other techniques. Genetic programming has mainly been used by hyper-heuristics to generate constructive heuristics. The components of low-level heuristics include existing low-level heuristics or components of these heuristics as well as problem characteristics. These components are combined using arithmetic operators and conditional operators such as if-then-else. Evolved heuristics can be disposable or reusable. Disposable heuristics are used to solve a particular instance of the problem. Reusable heuristics are generated using one or more instances of the problem and can be applied to unknown instances, i.e., instances of the problem not used in the induction and generation of the heuristic.

Generation perturbative hyper-heuristics produce new low-level perturbative heuristics through the combination of existing low-level perturbative heuristics and acceptance criteria using conditional statements, usually if-then-else statements. For example, the conditions used may include whether a solution has been found and whether a local optimum has been achieved (Nguyen, Zhang, & Johnston, 2011).

1.5 Main contributions

Robotics is becoming increasingly important in real-world applications. In fact, many expensive and high-risk tasks can be accomplished by robots autonomously. Swarm robotics represents a key research area to deal with scenarios such as environmental monitoring, surveillance, or search and rescue, where large areas must be covered. One of the challenges of swarm robotics is coordination, particularly when the environment is unknown a priori or is dynamic, and when robots cannot process all the information collected by other robots. Unfortunately, real-world scenarios often fall into these categories, and thus manual programming or centralized control is very difficult to apply and poorly effective.

In the literature, approaches based on hyper-heuristics have been very successful in solving complex problems. They aim to automatically generate algorithms from a given set of operators, and the goal is to produce a good solver rather than a good solution. Of course, hyper-heuristics have also been used in the field of robotics, for example to perform real-time path-planning for a UAV in unknown environments (Yu, Song, & Aleti, 2019). However, to the best of our knowledge, in the literature there are no methodologies for coordinating robots based on hyper-heuristics that use bio-inspired behavioral mechanisms as modular components of the low-level heuristics used to solve the problem at hand.

This thesis investigates the use of bio-inspired hyper-heuristics for coordinating swarms of robots to solve the problem of target search in unstructured environments. Specifically, the original contributions of this thesis are:

- a novel design methodology called SFE (Stigmergy, Flocking, and Evolution) based on hyper-heuristics for coordinating swarms of robots in target search. The approach combines two low-level constructive heuristics, i. e., stigmergy and flocking. Differential evolution is used to optimize the aggregation and tuning of modular heuristics on real-world scenarios;
- a modeling and optimization testbed that has been publicly released;
- a comparison between our swarm coordination methodology and some popular bio-inspired meta-heuristics made adaptive through differential evolution.

1.6 Outline of the thesis

This dissertation has five chapters. In chapter 1, an overview of the main topics considered in the thesis is outlined. In chapter 2 a theoretical background of bio-inspired metaheuristics and hyper-heuristics is presented. Specifically, the chapter begins by defining the concept of heuristics. Next, a review of the literature on bio-inspired metaheuristics is carried out. This is followed by a detailed description of some of the most important bio-inspired algorithms used in optimization problems. The second part of the chapter focuses on hyper-heuristics as a method to design more effective heuristics. Two categories of hyper-heuristics are investigated: the selection constructive hyper-heuristics and the generation constructive hyper-heuristics. Chapter 3 presents a novel hyper-heuristic-based methodology for coordinating swarms of drones in a decentralized way. First, the design of the modular components of the hyper-heuristic is detailed. Then, the simulation testbed and scenarios on which to evaluate the performance of the algorithm are presented. Chapter 4 lists some experimental applications of the previously described methodology: (i) the comparison between the proposed hyper-heuristic and some adaptive bio-inspired metaheuristics in the context of coordinating swarms of robots in target search, (ii) the tracking of dynamic targets by using swarms of drones, and (iii) the collection of plastics in the ocean through swarms of unmanned surface vehicles. Finally, Chapter 5 draws conclusions of the study and discusses possible future developments.

Chapter 2

2 From nature-inspired meta-heuristics to hyper-heuristics

Most classical or conventional optimization algorithms are deterministic. For example, in linear programming the simplex method is a deterministic algorithm. Deterministic optimization methods that use gradient information are called gradient-based algorithms. One of them is the well-known Newton-Raphson algorithm: in fact, it uses the values of the objective function and their derivatives, and works incredibly well on smooth unimodal problems. However, it no longer works well when there is some discontinuity in the objective function. In this case, it is preferable to use gradient-free or non-gradient-based algorithms that use no gradient at all, but only the values of the objective function. Examples of such types of algorithms are Nelder-Mead downhill simplex and Hooke-Jeeves pattern search.

In general, stochastic algorithms can be classified into two categories, although their difference is subtle: these are heuristics and meta-heuristics. Broadly speaking, heuristic means "to find" or "to discover by trial and error". A hard optimization problem can be solved by finding quality solutions in a reasonable amount of time, but there is no guarantee that optimal solutions will be found. Generally, these algorithms work most of the time but not all the time. This is fine when you want good solutions that are easily reachable, but it is no longer fine when you need necessarily the best solutions.

The so-called meta-heuristic algorithms represent a further development of heuristic algorithms. The prefix "meta" means "beyond" or "higher level", and usually these algorithms perform better than simple heuristics. Moreover, all meta-heuristic algorithms are characterized by a degree of trade-off between randomization and local search. There are no agreeable definitions of heuristics and meta-heuristics in the literature. Some researchers use the terms interchangeably. However, the most recent trend is to refer to meta-heuristics as all those stochastic algorithms that involve both randomization and local search. Since randomization enables the transition from local-scale search to global-scale search, almost all meta-heuristic algorithms can be considered suitable for global optimization.

Heuristics, working by trial and error, provide a method for obtaining acceptable solutions to a complex problem in a reasonable amount of time. In fact, every possible solution or combination cannot be evaluated due to the complexity of the problem of interest. The goal is to find a good feasible solution in an acceptable time. Obviously, there is no guarantee that the best solutions can be found. Furthermore, it is not even known whether the algorithm will work, and, if so, why it will work. The important thing is to have a practical and efficient algorithm that works in most cases by providing good quality solutions. Among the quality solutions that the algorithm succeeds in finding, some are expected to be near-optimal, although there is no guarantee of such optimality.

Any meta-heuristic algorithm involves two main components, which are exploration and exploitation or diversification and intensification. *Exploration* aims to generate various solutions in order to visit the search space on a global scale. *Exploitation* uses the information that a good current solution has been found in a certain area in order to focus on searching in a local region. The selection of the best solutions is combined with this behavior and ensures that the solutions will converge towards optimality. Vice versa, through randomization, exploration prevents solutions from being trapped in local optimum while simultaneously increasing the diversity of solutions. Typically, global optimality can be achieved through the right combination of these two main components.

There are many ways to classify meta-heuristic algorithms. These can be classified as population-based and trajectory-based. For example, genetic algorithms are population-based because they use a set of individuals. Similarly, the firefly algorithm (FA), particle swarm optimization (PSO), and cuckoo search all use multiple agents or particles.

Vice versa, simulated annealing uses a single solution or agent that moves within the search space or design space in a piecewise fragmented style. In this case, a better move or solution is always accepted, while a not-so-good move may be accepted with a certain probability. The moves or steps trace a trajectory in the search space, and the global optimum can be achieved by this trajectory with non-zero probability.

In the next sections we will introduce a selection of popular bio-inspired metaheuristic algorithms that have been used in this work for coordinating swarms of robots in target search missions.

Despite the success of bio-heuristics, there are relevant algorithm selection and parameterization costs associated with the specific application. Moreover, although adaptive, the logic of bio-heuristics is nevertheless constrained by models of biological species, and then, for example, it can be neither modularized nor aggregated. In order to overcome these limits, the field of hyper-heuristics represent an attempt to provide more generalized solutions to optimization problems. Hyper-heuristics achieve this by automating the combination of modular heuristics to generate more adaptable logics, so they work in the heuristic space rather than in the solution space. More specifically, a hyper-heuristic either selects or generates low-level heuristics, which are exploited to solve the problem at hand. To select or generate low-level heuristics, a hyper-heuristic can use different techniques such as case-based reasoning, local search, and evolutionary programming. In this work, an evolutionary optimization is used to optimize the aggregation and tuning of the bio-heuristics on a unique and continuous search space and, consequently, an efficient heuristics hybridization is generated for a given application domain.

2.1 Nature-inspired meta-heuristics

In the literature, many algorithms proposed for modeling swarms of robots are inspired by biological systems. In particular, solutions based on insect colonies manifest interesting properties such as local control and communication, self-organization and emergence of global behavior (Dorigo, Birattari, & Stützle, 2006) (Yang, 2009). For instance, ants release attractive pheromone trails as a medium for self-organization to mark and reinforce their most frequent paths. Coordination strategies for robots based on chemical trails have been experimented; for instance, in Fujisawa et al. (Fujisawa, Dobata, Kubota, Imamura, & Matsuno, 2008), ethanol trails have been used by robots as a medium to deposit and follow. However, chemical trails can cause problems for environmental impact, maintenance costs, control of transmission speed and range. For this reason, non-chemical media are more effective (Ducatelle, Di Caro, Pinciroli, & Gambardella, 2011). Masár et al. (Masár & Zelenka, 2012) have proposed a variant of the PSO algorithm for environment exploration. Another significant bio-inspired algorithm is called Artificial Bee Colony (ABC): it is based on the food foraging process of honey bees. ABC variants have been also used to coordinate robotic systems (Contreras-Cruz, Ayala-Ramirez, & Hernandez-Belmonte, 2015). Another well-known algorithm is the Firefly-based Team Strategy (FTS), proposed by Yang (Yang, 2009) and based on the flashing behavior of fireflies. In this work FTS, PSO and ABC will be considered as a reference for swarm coordination in target search missions.

Bio-inspired techniques, albeit provided with a certain variety of approaches, are still not organized as an operational framework: design and setting costs associated with every new type of mission and with new instances of known missions are a major drawback. The major difficulties are due to the lack of reliable guidance on how to select the algorithms and the parameters in different situations. When applied to real-world problems, the method tends to become tailored and problem-specific, characterized by expensive development and maintenance. For this purpose, a promising research direction is called *algorithm configuration*, whose goal is to automatically determine the appropriate parameters values for an algorithm. The goal can be considered as a search problem in the configuration space, in which the objective function measures the algorithm performance over a benchmark (Hutter, Hoos, & Stützle, 2007). Another approach is called *parameter control*, which performs an online tuning of the algorithm parameters at execution time (Eiben, Hinterding, & Michalewicz, 1999). However, to automate the tuning of bio-inspired methods for target search is a challenge in the field (Burke, et al., 2003). For this purpose, in this work evolutionary optimization is adopted.

2.1.1 Flocking behavior

Many species of birds found in nature adopt a type of behavior called flocking, which involves forming large groups of individuals that move together to a common destination. Other social animals also adopt similar collective behaviors, such as fish schooling and herd formation in ungulates. In these cases, local interactions among autonomous agents result in the emergence of collective-level behaviors in a distributed way. These behaviors are of interest to swarm robotics researchers, who have tried to replicate flocking in robotic swarms by studying the mechanisms underlying animal behavior. In most existing work, robots with limited sensing capabilities must keep a compact formation by measuring the distance and relative orientation of their neighbors (Turgut, Çelikkanat, Gökçe, & Şahin, 2008) (Virágh, et al., 2014) (Yasuda, Adachi, & Ohkura, 2014). Typically, the usual assumption on which flocking studies are based is that all robots have at least one neighbor that connects them to the rest of the swarm, while cases where individuals or groups of robots are outside the detection and communication range of the rest of the robots are not considered.

As just stated, a key requirement for implementing flocking behavior is the capability of a robot to measure the distance and relative orientation of neighboring robots. In real-world scenarios, since the detection and communication range is typically limited, in practice only a limited number of neighbors are detected by a given robot, and not the entire swarm population. This limitation does not prevent the implementation of flocking behavior by assuming that there are no isolated individuals or groups within the swarm. On the other hand, if the factors of scalability and processing complexity are taken into account, this limitation can be considered as an advantage.

The most distinctive feature of flocking compared to simple aggregation is the alignment of robot motion. This allows the group to collectively move toward a certain direction. The ontology of flocking behavior is showed in Figure 2.1. This is performed by simple agents, named *boids* in the literature, and consists of three fundamental features: separation, alignment, and cohesion (Reynolds, 1987).

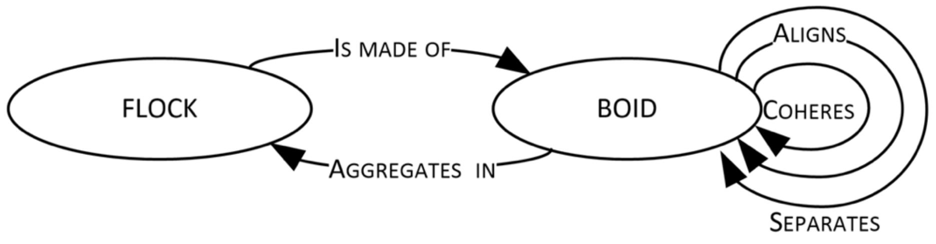


Figure 2.1 - Floking behavior ontology

Each agent moves according to the following prioritized rules:

1. *Separation*: the boid steers to avoid crowding flock mates. Separation prevents the overlapping of sensing areas by maintaining a minimum distance among the flock mates.

2. *Alignment*: the boid steers towards the average heading of the flock mates. Alignment allows to align the flock heading to the average heading of nearby agents.

3. *Cohesion*: the boid steers to move toward the center of gravity of the flock mates. Cohesion allows the agents to stay together.

For each rule, agents considered as flock mates are determined based on different reference ranges, as showed in Figure 2.2.

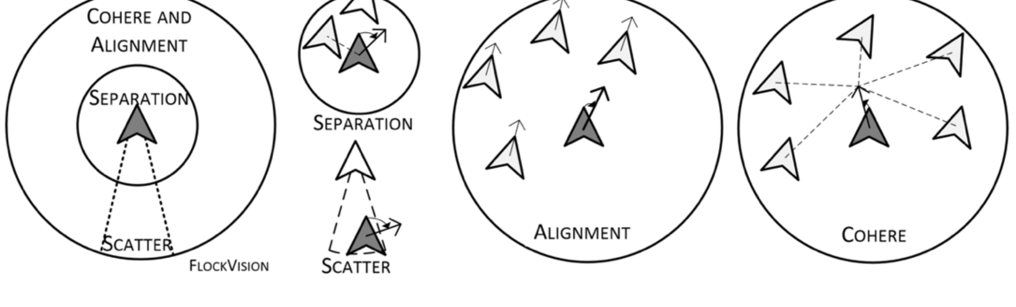


Figure 2.2 - Flocking rules: separation, alignment, and cohesion

The scatter procedure helps to maximize the sensing area, further separating nearby agents. Alfeo et al. have verified in their work (Alfeo, Cimino, De Francesco, Lega, & Vaglini, 2018) other important properties:

- the cohesion rule is not suitable for robots moving among many obstacles.
- the separation rule can be better exploited with an area different than circular, as represented in Figure 2.3.

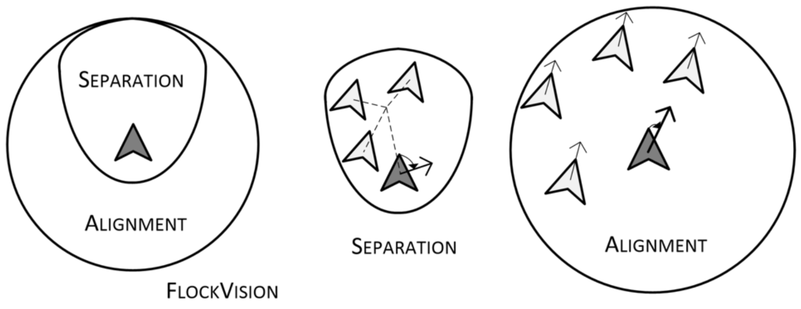


Figure 2.3 - Search area for robot flocking: separation and alignment

2.1.2 Ant Colony Optimization

The study of collective phenomena such as collective exploration can be carried out on ant colonies, which are one of the most significant examples. Exploration allows animals to obtain food, to discover new resources, to look for a new home, or to detect the presence of potential hazards, and consequently represents a very important task in nature. Ant colonies coordinate their behaviors through local interactions, operating without central control. Ants perceive only local, chemical, and tactile signals. In order to monitor their environment and detect both resources and hazards, ants in a colony must move so that if a food source appears or a certain event happens, some ants are likely to be close enough to detect it (Countryman, et al., 2015). Although individual ants are very simple, ant colonies show amazingly good results in achieving global goals. As a result, borrowing the behavior of ants and more generally social insects is becoming increasingly popular in distributed systems and robotics.

The natural behavior of ants to release pheromone trails and follow them has inspired Dorigo and Stützle (Dorigo, Birattari, & Stützle, 2006) to develop the Ant Colony Optimization (ACO) model. Ants live in colonies and the survival of the colony rather than the survival of individuals is the goal that drives their behavior. The ACO model was inspired by the foraging behavior of ants, and in particular the way they find the shortest paths between food sources and their nest. When searching for food, at the beginning ants randomly explore the area surrounding their nest. As they move, ants may release a trail of chemical pheromone on the ground and may also smell their presence. When choosing which route to take, they tend to be more likely to follow paths marked by strong concentrations of pheromone. After an ant identifies a food source and assesses its quantity and quality, it takes some of it back to the nest. Such information about quantity and quality of the food source can determine the amount of pheromone that an ant releases on the ground during its way back. Other ants will be steered by the pheromone trails to the food source. A parameterized probabilistic model, called pheromone model (Dorigo & Blum, 2005), represents the fundamental component of an ACO algorithm. During the last decade, several researchers in the field of robotics have proposed and applied many versions of Dorigo's method.

For example, taking into account the exploration problem in the domain of search and rescue operations, the mobile and autonomous robots must be able to determine the sequence of movements needed to explore the whole environment (Yang, 2020). Traditionally, the development of exploration strategies relies on approaches that involve exploring the environment incrementally by evaluating, based on a certain criterion, several candidate observation positions (which in this case are represented by neighboring cells) and selecting the next best position at each step. However, since finding as many targets as possible in as little time as possible is the main goal, the problem of building a map of the environment is not considered in this case. In search and rescue operations, time and battery limits are strong constraints and generally the mission success depends on the amount of areas explored rather than the quality of the built map. In addition to area exploration, robots should be also capable of multiple functionalities, and consequently both integration into a swarm and the ability to explore should be seamless and should not consume a large amount of the resources of the robot. Furthermore, the effectiveness of a search strategy depends on the ability to attract robots to unseen areas, to avoid situations where some areas are frequently revisited while others remain unexplored.

In general, the robots operate according to the following steps:

1. The robots sense the surrounding cells using on-board sensors.
2. The robots compute the sensed information in neighbor cells, in this case the concentration of pheromone.
3. The robots decide where to go next.
4. The robots move in their best local cell and start again from the step 1.

Carrying out search and rescue operations mainly means designing a movement strategy that allows a team of robots, each equipped only with simple sensors, to efficiently explore potentially complex environments. Exactly as it happens in biology for ant colonies, the principle of pheromone-based coordination is used, where each robot releases pheromone on the visited cells in order to inform, in an indirect way, the other robots about the already explored areas. As already pointed out, according to this approach robots do not have to communicate directly, but advise other robots not to enter the already visited regions through the release of pheromone on the boundaries. If on-board sensors detect the pheromone, the robot understands that it is entering a potentially explored area, and as a result can preemptively decide to change direction. By exploiting the physical properties that exist in a real-world environment, it is desirable to develop a simple algorithm that, through the use of pheromone, can enable complex collective behavior within a large group of simple homogeneous robots. Such an algorithm does not necessarily require that the map topology be maintained in memory. Decision making is based on probabilistic evaluations that exploit local pheromone information. Since the main goal is to build a model that performs the assigned task through self-adaptive decision mechanisms, issues related to pheromone release, sensors, or controls cannot be considered.

In general, as the robots are exploring the area, they release pheromone on the visited cells and each robot uses the pheromone distribution in its immediate neighborhood to decide where to move. Exactly as occurs in nature, pheromone trails change both in space and time. The pheromone released on a cell by a robot diffuses outwards cell-by-cell until a certain distance R_s such that $R_s \subset A \subset \mathbb{R}^2$ and the amount of the pheromone decreases as the distance from the robot increases. Mathematically, the pheromone diffusion is defined as follows. Assume that robot k at iteration t is located in a cell with coordinates $(x_k^t, y_k^t) \in A$. Then the amount of pheromone that the robot releases at cell c with coordinates (x, y) is given by:

$$\Delta\varphi_{kc}^t = \begin{cases} \Delta\varphi_0 e^{\frac{-r_{kc}}{a_1}} - \frac{\varepsilon}{a_2}, & if\ r_{kc} \leq R_s \\ 0, & otherwise \end{cases} \qquad (2.1)$$

where r_{kc} is the distance between the robot k and cell c, defined as:

$$r_{kc} = \sqrt{(x_k^t - x)^2 + (y_k^t - y)^2} \qquad (2.2)$$

As it happens in biological systems, the pheromone spreads up to a certain distance and after that it is no longer sensed by other robots. In addition, $\Delta\varphi_0$ is the quantity of pheromone released in the cell where robot k is placed and it is the maximum amount of pheromone, ε is a heuristic value (noise), and $\varepsilon \in (0, 1)$. Furthermore, a_1 and a_2 are two constants to reduce or increase the effect of the pheromone and noise. It should be noted that if multiple robots release pheromone in the environment at the same time, then the total amount of pheromone that can be sensed in a cell c depends on the contribution of many robots. Furthermore, the released pheromone concentration evaporates over time and so it is not fixed. The rate of evaporation of pheromone is given by ρ $(0 \leq \rho \leq 1)$, and the total amount of pheromone evaporated in cell c at step t is given by the following function:

$$\xi_c^t = \rho\varphi_c^t \qquad (2.3)$$

where φ_c^t is the total amount of pheromone on cell c at iteration t. Considering the evaporation of the pheromone and the diffusion according to the distance, the total amount of pheromone in cell c at iteration t is given by:

$$\varphi_c^t = \varphi_c^{(t-1)} - \xi_c^{(t-1)} + \sum_{k=1}^{N^R} \Delta\varphi_{kc}^t \qquad (2.4)$$

where N^R is the number of robots that are able to release pheromone on the cell c.

At each time step, without knowledge of the whole area, the algorithm selects the most appropriate cell for each robot from a set of neighboring cells. In fact, the algorithm does not expect the robots to have global information about the environment. The aim is to avoid any overlapping and redundant efforts. Therefore, in order to complete the mission as quickly as possible, while avoiding any waste of resources and energy, the robots must be highly distributed in the search area. Each robot k, at each time step t, is placed on a particular cell c_k^t that is surrounded by a set of accessible neighbor cells $N(c_k^t)$. Essentially, each robot senses the pheromone released into the nearby cells, and then it chooses which cell to move to at the next step.

At each time step, the probability for robot k of moving from cell c_k^t to cell $c \in N(c_k^t)$ can be calculated by:

$$p(c|c_k^t) = \frac{(\varphi_c^t)^\mu (\eta_c^t)^\lambda}{\sum_{b\in N(c_k^t)} (\varphi_b^t)^\mu (\eta_b^t)^\lambda}, \qquad \forall c \in N(c_k^t) \tag{2.5}$$

where φ_c^t is the amount of pheromone in the cell c at iteration t and η_c^t is the heuristic variable to avoid the robots being trapped in a local minimum. In addition, μ and λ are two constant parameters which balance the weight to be given to pheromone values and heuristic values, respectively. Robot k moves into the cell c that satisfies the following condition:

$$c = \arg\min[p(c|c_k^t)] \tag{2.6}$$

In this way, the robots will prefer less frequently visited areas and more likely they will direct towards unexplored regions.

The ACO-based exploration strategy applied by a swarm of robots in *Forager State* is detailed in Algorithm 2.1, which provides the pseudocode for the pheromone-based control that must be executed at each time step.

On the first iteration, **Algorithm 2.1** initializes all cells with the same pheromone trail value, set to be zero. This represents the fact that the cells have not yet been visited by any of the robots and ensures that the initial probabilities of a cell being chosen are nearly identical. Thus, at first the cell is chosen practically at random. Then the robots move from one cell to another according to the cell transition rule in Eq. (2.6). In the following iterations, the unexplored cells become more attractive to the robots. By using this approach, robots explore the area following the direction that is opposite to the pheromone gradient. Then the pheromone trails on the cells visited by the robots are updated using Eq. (2.4). When the mission is completed, i.e., all targets have been performed, **Algorithm 2.1** terminates execution for each robot.

Algorithm 2.1
ACO-based exploration strategy

```
begin
    step 1: initialization
        set $t$: $\{t$ is the time step$\}$. Define $R_s$, $\Delta\varphi_0$, $\varepsilon$, $a_1$, $a_2$, $\rho$, $\mu$, $\lambda$.
    step 2: generation coordination system
        for the whole swarm, set the initial locations in terms of $x$ and $y$
        coordinates.
    step 3: procedure
    while the stop criteria are not satisfied do
        foreach robot $k$ in Forager State do
            evaluate the current location $c_k^t$;
            evaluate neighborhood cells $N(c_k^t)$;
            compute $c$ according to Eq. (2.6);
            if $c$.hasObstacle() or $c$.isOccupied() or $c$.isInaccessible() then
                choose a random cell $c^* \in N(c_k^t)$;
                move robot $k$ towards $c^*$;
            else
                move robot $k$ towards $c$;
                release pheromone according to Eq. (2.1);
            end if
        end foreach
        foreach cell $c \in A$ do
            update pheromone according to Eq. (2.4);
        end foreach
            update $t$;
    end while
end
```

2.1.3 Firefly algorithm

The firefly algorithm (FA) was developed by Xin-She Yang in late 2007 and published in 2008 (Yang, 2009). In tropical and temperate regions FA is based on the flashing patterns and behavior of tropical fireflies. The flashing light of fireflies is an amazing show in the summer sky. There are about two thousand species of fireflies, and most of them emit short, rhythmic flashes. The pattern of the flashes is often unique to a particular species. The flashing light is produced by a process of bioluminescence. The actual functions of these signaling systems are yet discussed. However, two basic functions of such flashes are to attract mating partners (communication) and to attract potential prey (Lewis & Cratsley, 2008). Furthermore, flashing may also act as a protective warning mechanism to remind potential predators of the bitter taste of fireflies.

The rhythmic flash, the rate of flashing, and the amount of time between flashes are all part of the signaling system that joins the sexes (Lewis & Cratsley, 2008). In the same species females reply to the unique flashing pattern of the male. Whereas, in some species female fireflies may even mimic the mating flashing pattern of other species in order to attract and then eat male fireflies, who may exchange flashes as a suitable potential mate. Some tropical fireflies may even synchronize their flashes, thus forming an emerging self-organized biological behavior.

It is known that the intensity of light at a particular distance r from the light source complies with the inverse-square law. That is to say, the light intensity I decreases as the distance r increases in terms of $I \propto 1/r^2$. Moreover, the air absorbs the light, which gets progressively weaker as the distance increases. These two factors combined result in most fireflies being visible at a limited distance, usually several hundred meters at night, that is good enough for fireflies to communicate.

The flashing light can be modeled in such a way that it is associated with the objective function to be optimized, thus making it possible to formulate new optimization algorithms. It is possible to idealize some of the flashing characteristics of fireflies in order to develop firefly-inspired algorithms. To simplify the description of the standard FA, the following three idealized rules are used:

- All fireflies are unisex, so a firefly will be attracted to other fireflies regardless of their gender.
- The attractiveness is proportional to the brightness of a firefly. Thus, for any two flashing fireflies, the less bright one will move toward the brighter one. Both

attractiveness and brightness decrease as their distance increases. If there is no one brighter than a specific firefly, it will move randomly.

- The brightness of a firefly is affected or determined by the design of the objective function. For a maximization problem, the brightness may simply be proportional to the value of the objective function. Other forms of brightness can be defined similarly to the fitness function in genetic algorithms.

In the FA, there are two important issues: the variation of light intensity and the formulation of attractiveness. For simplicity, we can always assume that the attractiveness of a firefly is determined by its brightness, which in turn is associated with the encoded objective function.

In the simplest case for maximum optimization problems, the brightness I of a firefly at a particular location x can be chosen as $I(x) \propto f(x)$. However, the attractiveness β is relative, it should be seen in the eyes of the watcher or judged by the other fireflies. Thus, it will vary with the distance r_{ij} between firefly i and firefly j. In addition, light intensity decreases with the distance from its source, and light is also absorbed in the media, so the attractiveness should also vary with the degree of absorption.

In the simplest form, the light intensity $I(r)$ varies according to the inverse square law:

$$I(r) = \frac{I_s}{r^2} \qquad (2.7)$$

where I_s is the intensity at the source. For a particular medium with a fixed light absorption coefficient γ, the light intensity I varies with the distance r. In formula:

$$I = I_0 e^{-\gamma r} \qquad (2.8)$$

where I_0 is the original light intensity at zero distance $r = 0$. In order to avoid the singularity at $r = 0$ in the expression I_s/r^2, the combined effect of both the inverse-square law and absorption can be approximated as the following Gaussian form:

$$I(r) = I_0 e^{-\gamma r^2} \qquad (2.9)$$

Since the attractiveness of a firefly is proportional to the intensity of the light that is seen by the neighboring fireflies, the attractiveness β of a firefly can be defined as following:

$$\beta = \beta_0 e^{-\gamma r^2} \tag{2.10}$$

where β_0 is the attractiveness at $r = 0$. Since it is often faster to calculate $1/(1 + r^2)$ than an exponential function, the above function, if necessary, can conveniently be approximated as following:

$$\beta = \frac{\beta_0}{1+\gamma r^2} \tag{2.11}$$

It may be advantageous to use this approximation in some applications. Both (2.10) and (2.11) define a characteristic distance $\Gamma = 1/\sqrt{\gamma}$ over which the attractiveness changes significantly from β_0 to $\beta_0 e^{-1}$ for Eq. (2.10) or $\beta_0/2$ for Eq. (2.11).

The distance between any two fireflies i and j at locations x_i and x_j, respectively, is the Euclidean distance:

$$r_{ij} = \left\|x_i - x_j\right\| = \sqrt{\sum_{k=1}^{d}\left(x_{i,k} - x_{j,k}\right)^2} \tag{2.12}$$

where $x_{i,k}$ is the k-th component of the spatial coordinate x_i of i-th firefly. In the 2D case, the distance is expressed as following:

$$r_{ij} = \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2} \tag{2.13}$$

The motion of a firefly i that is attracted by another more attractive (brighter) firefly j is determined by:

$$x_i^{t+1} = x_i^t + \beta_0 e^{-\gamma r_{ij}^2}\left(x_j^t - x_i^t\right) + \alpha \varepsilon_i^t \tag{2.14}$$

where the second term is due to the attraction. The third term is randomization, with α being the randomization parameter, and ε_i is a vector of random numbers drawn from a uniform distribution or Gaussian distribution. For example, in the simplest form ε_i can be replaced by the expression $(\sigma - 1/2)$ where σ is a random number generator uniformly distributed in $[0, 1]$. For most implementation, it can be taken $\beta_0 = 1$ and $\alpha \in [0, 1]$.

The basic steps of the FA can be summarized as the pseudo code shown in **Algorithm 2.2**.

Algorithm 2.2
Firefly algorithm

```
Data: objective functions $f(x)$
Result: best or optimal solution
Initialization of parameters ($n$, $\alpha$, $\beta$, and $\gamma$);
Generate an initial population of $n$ fireflies in locations $x_i$ $(i = 1, 2, \dots, n)$;
Light intensity $I_i$ at $x_i$ is determined by $f(x_i)$;
while $(t < MaxGeneration)$ do
    for $i = 1{:}n$ $(all\ n\ fireflies)$ do
        for $j = 1{:}n$ $(all\ n\ fireflies)$ do
            if $(I_i < I_j)$ then
                move firefly $i$ towards $j$ (for maximization problems)
                according to Eq. (2.14);
            end if
            vary attractiveness with distance $r$ via $exp[-\gamma r^2]$;
            evaluate new solutions and update light intensity;
        end for
    end for
    rank the fireflies and find the current global best $g^*$;
end while
Postprocess results and visualization;
```

2.1.4 Particle Swarm Optimization

Particle Swarm Optimization (PSO) has been developed by Kennedy and Eberhart in 1995 (Kennedy & Eberhart, 1995), based on the behavior of swarms in nature, such as bird flocking and fish schooling. Since then, due to its simplicity and flexibility, it has become one of the most popular swarm intelligence-based algorithms used. PSO has been applied to almost all areas of optimization, computational intelligence, and design applications. It uses the randomness of real numbers and the global communication between swarm particles.

The PSO algorithm searches the space of an objective function by adjusting the trajectories of individual agents, called *particles*, as the piecewise paths formed by positional vectors in a quasi-stochastic fashion (Kennedy & Eberhart, 1995). The motion of a swarm particle consists of two main components: a deterministic component and a stochastic component. Each particle is attracted toward the position of the current global best g^* and its own best location x_i^* in history, while at the same time it has a tendency to move randomly.

When a particle finds a position that is better than any previously found positions, PSO algorithm updates that position as the new current best for particle i. There is a current best for all n particles at any time step t during iterations. The aim is to find the global best among all the current individual best solutions until the objective function no longer improves or after a specific number of iterations. The movement of particles is schematically represented in Figure 2.4, where x_i^* is the current best for particle i, and $g^* = min\{f(x_j^*)\}$ for $(j = 1, 2, \dots, n)$ is the current global best at time step t.

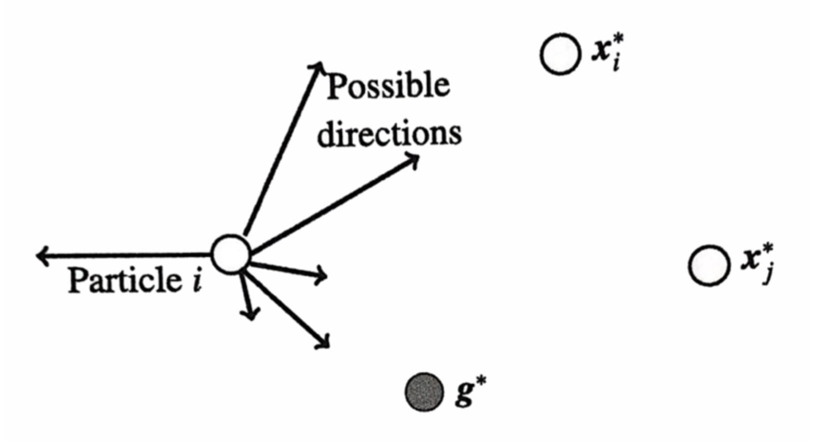


Figure 2.4 - Schematic representation of the motion of a particle in PSO

Let x_i and v_i be the position vector and the velocity vector for particle i, respectively. The new velocity vector is determined by the following formula:

$$v_i^{t+1} = v_i^t + \kappa_1 \varepsilon_{i,1}^t [g^* - x_i^t] + \kappa_2 \varepsilon_{i,2}^t [x_i^* - x_i^t] \tag{2.15}$$

where $\varepsilon_{i,1}$ and $\varepsilon_{i,2}$ are two random vectors, and each entry takes the values between 0 and 1. The parameters κ_1 and κ_2 are the learning parameters or acceleration constants, which can be typically taken as $\kappa_1 \approx \kappa_2 \approx 2$.

The initial positions of all particles should distribute quite uniformly so that it is possible to sample over most regions, that is particularly important for multimodal problems. The initial velocity of a particle can be taken as zero, that is $v_i^{t=0} = 0$. The new position can then be updated as the following:

$$x_i^{t+1} = x_i^t + v_i^{t+1} \Delta t \tag{2.16}$$

where Δt is the time increment. Since PSO is iterative with a discrete integer time counter, it can be set $\Delta t = 1$ for all implementations. Although v_i can assume any values, it is usually bounded in some range $[0, v_{max}]$.

The fundamental steps of the particle swarm optimization can be summarized as the pseudo code shown in **Algorithm 2.3**.

Algorithm 2.3
Particle Swarm Optimization

```
Data: objective functions $f(x)$
Result: best or optimal solution
Initialize locations $x_i$ and velocities $v_i$ of $n$ particles;
Find the global best $g^*$ from $min\{f(x_1), \dots, f(x_n)\}$ at $t = 0$;
while ($termination\ criterion\ is\ False$) do
    for $all\ n\ particles\ and\ all\ d\ dimensions$ do
        generate new velocity $v_i^{t+1}$ using Eq. (2.15);
        calculate new locations according to Eq. (2.16);
        evaluate the objective function at new locations $x_i^{t+1}$;
        find the current best for each particle $x_i^*$;
    end for
    find the current global best $g^*$;
    update $t = t + 1$;
end while
Output the final results $x_i^*$ and $g^*$;
```

There are many versions that extend the standard PSO algorithm, and the most obvious improvement is probably to use the inertia function $\theta(t)$ so that the velocity v_i^t is replaced by $\theta(t)v_i^t$ as following:

$$v_i^{t+1} = \theta v_i^t + \kappa_1 \varepsilon_{i,1}^t [g^* - x_i^t] + \kappa_2 \varepsilon_{i,2}^t [x_i^* - x_i^t] \qquad (2.17)$$

where θ takes in theory the values between 0 and 1 (Chatterjee & Siarry, 2006). In the simplest case, the inertia function can be taken as a constant, typically $\theta \approx 0.5 \sim 0.9$. This is equivalent to introduce a virtual mass to stabilize the motion of the particles, and thus the algorithm is expected to converge more quickly.

2.1.5 Artificial Bee Colony Optimization

Another evolutionary approach is the Artificial Bee Colony (ABC) algorithm by Karaboga et al. (Karaboga & Akay, 2009). This algorithm is inspired by the foraging behavior of honeybees when searching for a quality food source. In the ABC algorithm, there is a population of food locations, and artificial bees change these food locations over time. The algorithm uses a set of computational agents called honeybees to find the optimal solution. The honeybees in ABC can be categorized into three groups: employed bees, onlooker bees, and scout bees. The employed bees exploit food locations, while the onlooker bees wait for information from employed bees about the nectar amount of the food locations. The onlooker bees select food locations using information from the employed bees and exploit the selected food locations. Finally, the scout bees find new random food locations. Each solution, in the search space, consists of a set of optimization parameters that represent a food location. The number of employed bees is equal to the number of food sources.

In the ABC algorithm, the location of a food source represents a possible solution to the optimization problem, and the nectar quantity of a food source is equal to the quality (fitness) of the corresponding solution. The number of employed bees or onlooker bees is equal to the number of solutions in the population. At the first step, the ABC generates a randomly distributed initial population $P(C = 0)$ of SN solutions (food source locations), where SN corresponds to the size of employed bees or onlooker bees. Each solution x_i $(i = 1, \dots, SN)$ is a D-dimensional vector. Here, D is the number of optimization parameters. After initialization, the population of the locations (solutions) is subject to repeated cycles, $C = 1, 2, \dots, MCN$ (where MCN is the Maximum Cycle Number), of the search processes of the employed bees, the onlooker bees, and the scout bees.

Each cycle of the search consists of three steps:

1) Sending the employed bees onto their food sources and evaluating their nectar amounts.

2) After sharing the nectar information of food sources, the onlooker bees select the regions of food sources and evaluate the nectar quantity of food sources.

3) Determining the scout bees and then sending them randomly onto possible new food sources.

In the initialization phase, a set of food sources is randomly selected by bees and their nectar amounts are evaluated. At the first step of the cycle, these bees enter the hive and share nectar information about the sources with bees waiting on the dance area. A bee that waits on the dance area for making a decision to choose a food source is called an onlooker, while the bee that goes to the food source that she has visited just before is called an employed bee. After sharing their information with the onlookers, each employed bee goes to the area of the food source visited in the previous cycle by herself, as that food source exists in her memory, and then chooses a new food source by visual information in the neighborhood of the one in her memory and evaluates its nectar quantity. In the second step, an onlooker prefers a food source area based on nectar information shared by employed bees on the dance area. As the nectar amount of a food source increases, the probability of that food source being selected also increases. After arriving at the selected area, it chooses a new food source in the neighborhood of the one in memory based on visual information as in the case of the employed bees. The decision of the new food source is made by the bees based on the process of visually comparing the locations of the food sources. At the third step of the cycle, when the nectar of a food source is abandoned by the bees, a new food source is randomly determined by a scout bee and replaced with the abandoned one. In the model of (Karaboga & Akay, 2009), at most one scout bee exits to search for a new food source at each cycle, and the number of employed bees and onlooker bees are selected to be equal to each other. These three steps are repeated for a predefined number of cycles called the Maximum Cycle Number (MCN) or until a termination criterion is met.

An artificial onlooker bee selects a food source based on the probability value assigned to that food source, p_i, calculated by the following expression:

$$p_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_n} \quad (\ 2.18\ )$$

where fit_i is the fitness value of the solution i which is proportional to the nectar amount of the food source in the location i and SN is the number of food sources that is equal to the number of employed bees or onlooker bees.

In order to produce a candidate food position from the old one in memory, the ABC uses the following expression:

$$v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) \quad (\ 2.19\ )$$

where $k \in \{1, \dots, SN\}$ and $j \in \{1, \dots, D\}$ are randomly chosen indexes. Although k is determined randomly, it must be different from i. ϕ_{ij} is a random number between $[-1, 1]$.

It controls the locating of nearby food sources around x_{ij} and represents the visual comparison of two food locations by a bee. As can be seen from (2.19), as the difference between the parameters of x_{ij} and x_{kj} decreases, the perturbation on the position x_{ij} also decreases. Thus, as the search comes closer to the optimum solution in the search space, the step length is adaptively reduced. If the value of a parameter resulting from this operation exceeds its predefined limit, the parameter can be set to an acceptable value, for example it can be set to its limit value.

The food source whose nectar is abandoned by the bees is replaced with a new food source by the scouts. In ABC, this is simulated by randomly generating a location and replacing it with the abandoned one. In ABC, if a location cannot be improved further through a predefined number of cycles, then that food source is considered abandoned. The value of the predefined number of cycles is an important control parameter of the ABC algorithm, which is called the "abandonment limit". Assume that the abandoned source is x_i and $j \in \{1, \dots, D\}$, then the scout discovers a new food source to be replaced with x_i. This operation can be defined as in the following:

$$x_i^j = x_{min}^j + rand[0,1]\left(x_{max}^j - x_{min}^j\right) \qquad (2.20)$$

After that each candidate source location v_{ij} is generated and then evaluated by the artificial bee, its performance is compared with that of the old source. If the new food source has equal or better nectar than the old source, it is replaced with the old one in memory. Otherwise, the old one is maintained in memory. In other words, a greedy selection mechanism is employed as the selection operation between the old source location and the candidate source.

Totally, ABC algorithm employs four different selection processes:

1) a global probabilistic selection process, in which the probability value is calculated by (2.18) and used by the onlooker bees to discover promising regions.

2) a local probabilistic selection process performed in a region by the employed bees and the onlooker bees depending on the visual information such as color, shape and fragrance of the flowers (sources) to determine a food source around the source in memory as described by (2.19).

3) a local selection called greedy selection process performed by onlooker and employed bees in that if the nectar amount of the candidate source is better than that of the

current source, the bee forgets the current one and stores the candidate source generated by (2.19); otherwise, the bee keeps the current one in memory.

4) a random selection process performed by scouts as defined in (2.20).

It is clear from the above discussion that there are three control parameters in the basic ABC: (i) the number of food sources that is equal to the number of employed or onlooker bees (SN), (ii) the limit value for abandonment, and (iii) the maximum number of cycles (MCN).

In the case of bees, the recruitment rate represents a measure of how quickly the bee colony locates and exploits a newly found food source. The artificial recruitment could similarly represent a measure of how quickly feasible or *good solutions* to difficult optimization problems can be discovered. The bee colony survival and evolution depend on the rapid discovery and efficient exploitation of the best food sources. Similarly, the successful solution of difficult engineering problems is related to the relatively fast discovery of *good solutions* especially for problems that need to be solved in real time. In a robust search process, the exploration and exploitation processes must be performed together. In the ABC algorithm, while the onlookers and employed bees perform the exploitation process in the search space, the scouts control the exploration process. The detailed pseudocode of ABC is shown in **Algorithm 2.4**.

Algorithm 2.4
Artificial Bee Colony

```
Data: objective functions $f(x)$
Result: best or optimal solution
Initialize the population of solutions $x_i$, $i = 1, \dots, SN$;
Evaluata the population;
$cycle = 1$;
while ($cycle \neq MCN$) do
    calculate new solutions $v_i$ for the employed bees by using Eq. (2.19);
    evaluate the objective function at new solutions $v_i$ for the employed bees;
    apply the greedy selection process for the employed bees;
    calculate the probability values $p_i$ for the solutions $x_i$ by using Eq. (2.18);
    calculate new solutions $v_i$ for the onlookers from the solutions $x_i$
    selected depending on $p_i$;
    evaluate the objective function at new solutions $v_i$ for the onlooker bees;
    apply the greedy selection process for the onlooker bees;
    determine the abandoned solution for the scout (if it exists);
    replace the abandoned solution with a new randomly calculated solution
    $x_i$ by using Eq. (2.20);
    memorize the best solution achieved so far;
    $cycle = cycle + 1$;
end while
Output the best solution $x_i^*$;
```

2.1.6 Differential Evolution

Differential evolution, or DE, was first developed by R. Storn and K. Price in their nominal papers in 1996 and 1997 (Storn & Price, 1997). DE is a vector-based meta-heuristic algorithm that shows some similarity to pattern search and genetic algorithms because of its use of crossover and mutation. DE is a stochastic search algorithm with self-organization and does not use derivative information. Thus, it is a population-based method with no derivatives. Also, DE uses real numbers as solution strings, so no encoding and decoding is required.

As in genetic algorithms, the design parameters in a d-dimensional search space are represented as vectors, and various genetic operators are applied on their string bits. However, in contrast to genetic algorithms, differential evolution performs operations on each component (or each dimension of the solution). Almost everything is done in terms of vectors. For example, in genetic algorithms, mutation is performed at one site or multiple sites on a chromosome, whereas in differential evolution, a difference vector of two randomly chosen population vectors is used to perturb an existing vector. Such a vector mutation can be seen as a more implementation-efficient approach. This type of perturbation is performed on every population vector and thus can be expected to be more efficient. Similarly, crossover is also a vector-based swapping of chromosomes or vector segments. In addition to using mutation and crossover as differential operators, DE has explicit updating equations. This also makes it easy to implement and design new variants.

For a d-dimensional optimization problem with d parameters, a population of n solution vectors x_i, where $i = 1, 2, \dots, n$, is initially generated. For each solution x_i at any generation t, it can be used the following formal notation:

$$x_i^t = \left(x_{1,i}^t, x_{2,i}^t, \dots, x_{d,i}^t\right) \qquad (\ 2.21\ )$$

that consists of d-components in the d-dimensional space. This vector can be considered as the chromosome or genome.

DE involves three main steps:

1. Mutation.
2. Crossover.
3. Selection.

Mutation is performed by the mutation scheme. For each vector x_i at any time or generation t, three distinct vectors x_p, x_q, and x_r at t are first randomly chosen, and then it is generated a so-called donor vector by the following mutation scheme:

$$v_i^{t+1} = x_p^t + F\left(x_q^t - x_r^t\right) \tag{2.22}$$

where $F \in [0, 2]$ is a parameter, frequently denoted as the *differential weight*. This requires that the minimum number of the population size is $n \geq 4$. In practice, a scheme with $F \in [0, 1]$ is more efficient and stable. In fact, almost all the studies in the literature use $F \in [0, 1]$. In Figure 2.5 it is shown that the perturbation $\delta = F\left(x_q - x_r\right)$ to the vector x_p is used to calculate a donor vector v_i.

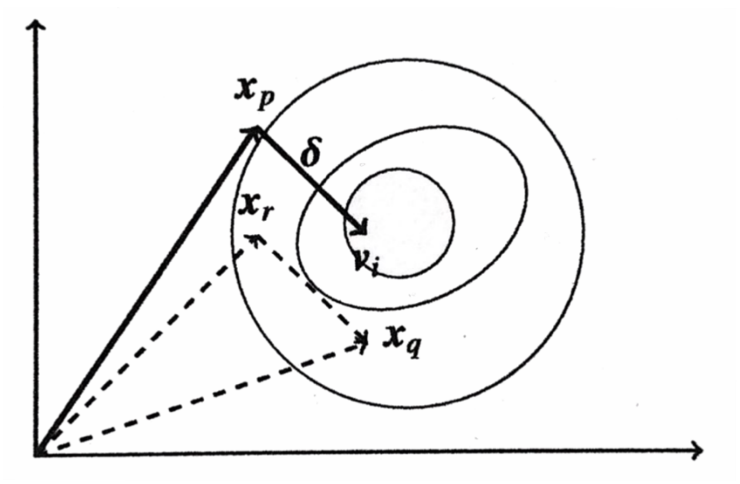


Figure 2.5 - Schematic representation of donor (mutation) vector

The crossover is controlled by a crossover parameter $C_r \in [0, 1]$, that represent the rate or probability for crossover. The crossover can be performed in two ways: binomial and exponential. The binomial scheme carries out crossover on each of the d components or variables. By generating a uniformly distributed random number $r_{j,i} \in [0, 1]$, the j-th component of v_i is processed in the following way:

$$u_{j,i}^{t+1} = \begin{cases} v_{j,i}^{t+1} & if\ r_{j,i} \leq C_r \\ x_{j,i}^{t} & otherwise \end{cases} \tag{2.23}$$

where $j = 1, 2, \ldots, d$. In this way, it can be randomly decided whether to swap each component with the donor vector component or not.

In order to ensure that $v_i^{t+1} \neq x_i^t$, that may increase the exploratory or evolutionary efficiency, the Eq. (2.23) can be replaced by the following:

$$u_{j,i}^{t+1} = \begin{cases} v_{j,i}^{t+1} & if\ r_{j,i} \leq C_r\ or\ j = J_r \\ x_{j,i}^t & if\ r_{j,i} > C_r\ and\ j \neq J_r \end{cases} \tag{2.24}$$

where $J_r \in \{1, 2, \dots, d\}$ is a random index generated by permutation.

In the exponential scheme, a segment of the donor vector is selected, and this segment starts with a random integer k and have a random length L, that can include more than one component. Mathematically, this means choosing $k \in [0, d-1]$ and $L \in [1, d]$ randomly, and the new vector is calculated by the following:

$$u_{j,i}^{t+1} = \begin{cases} v_{j,i}^{t+1} & for\ j = k, \dots, k+L-1 \\ x_{j,i}^t & otherwise \end{cases} \tag{2.25}$$

where $k + L - 1 \in [1, d]$. Since the binomial is easier to implement, it is used the binomial crossover in most of the implementations.

The selection is basically the same as that used in genetic algorithms. It involves selecting the best fit and, for a minimization problem, the minimum objective value. Therefore, the new solution vector is given by the following:

$$x_i^{t+1} = \begin{cases} u_i^{t+1} & if\ f(u_i^{t+1}) \leq f(x_i^t) \\ x_i^t & otherwise \end{cases} \tag{2.26}$$

All of the three DE components can be found in the pseudocode shown in **Algorithm 2.5**.

Algorithm 2.5
Differential Evolution

```
Data: objective functions $f(x)$
Result: best or optimal solution
Initialize the population $x$ with randomly generated solutions;
Set the weight $F \in [0, 2]$ and crossover probability $C_r \in [0, 1]$;
while ($stopping\ criterion\ is\ False$) do
    for $i = 1{:}\, n$ ($all\ x_i\ solution\ vectors$) do
        randomly choose three distinct vectors $x_p$, $x_q$ and $x_r$;
        generate a new vector $v_i$ by mutation scheme (2.22);
        generate a random index $J_r \in \{1, 2, \dots, d\}$ by permutation;
        for $j = 1{:}\, d$ ($all\ v_{j,i}\ components\ of\ v_i$) do
            generate a randomly distributed number $r_{j,i} \in [0, 1]$;
            update by Eq. (2.24);
        end for
        select and update the solution by Eq. (2.26);
    end for
end while
Postprocess and output the best solution;
```

The overall efficiency of the search is controlled by two parameters: the differential weight F and the crossover probability C_r. Most studies have focused on the choice of F, C_r and n as well as the variations of Eq. (2.22). In fact, many different ways of formulating Eq. (2.22) can be used for generating the mutation vectors. This results in various schemes with the naming convention DE/x/y/z, where x is the mutation scheme (rand or best), y is the number of difference vectors, and z is the crossover scheme (binomial or exponential). So, DE/rand/1/bin means the basic DE scheme using random mutation and one difference vector with a binomial crossover scheme.

The basic DE/rand/1/bin scheme is given in Eq. (2.22), that is:

$$v_i^{t+1} = x_p^t + F\left(x_q^t - x_r^t\right) \tag{2.27}$$

If the vector x_p^t is replaced by the current best x_{best}^t found so far, the previous scheme is changed into the so-called DE/best/1/bin scheme as the following:

$$v_i^{t+1} = x_{best}^t + F\left(x_q^t - x_r^t\right) \tag{2.28}$$

There is no reason why one cannot use more than three distinct vectors. For example, if four different vectors are used plus the current best, the DE/best/2/bin scheme is considered:

$$v_i^{t+1} = x_{best}^t + F\left(x_{k_1}^t + x_{k_2}^t + x_{k_3}^t + x_{k_4}^t\right) \qquad (\ 2.29\ )$$

Furthermore, if five different vectors are used, the scheme becomes DE/rand/2/bin:

$$v_i^{t+1} = x_{k_1}^t + F_1\left(x_{k_2}^t - x_{k_3}^t\right) + F_2\left(x_{k_4}^t - x_{k_5}^t\right) \qquad (\ 2.30\ )$$

where F_1 and F_2 are differential weights in $[0, 1]$. Obviously, for simplicity it can also be taken $F_1 = F_2 = F$.

Following a similar strategy, it is possible to design various schemes. For example, these variants can be written in a generalized form as follows:

$$v_i^{t+1} = x_{k_1}^t + \sum_{s=1}^{m} F_s \cdot \left(x_{k_{2,s}}^t - x_{k_{3,s}}^t\right) \qquad (\ 2.31\ )$$

where $m = 1, 2, 3, \ldots$ and F_s $(s = 1, \ldots, m)$ are the scale factors. The number of vectors involved into these schemes is equal to $2m + 1$.

On the other hand, there is also another type of variants that uses an additional influence parameter $\lambda \in (0, 1)$. For example, DE/rand-to-best/1/* scheme can be written as the following:

$$v_i^{t+1} = \lambda x_{best}^t + (1 - \lambda)x_{k_1}^t + F\left(x_{k_2}^t - x_{k_3}^t\right) \qquad (\ 2.32\ )$$

which introduces an extra parameter λ. Again, this type of variants can be generalized as follows:

$$v_i^{t+1} = \lambda x_{best}^t + (1 - \lambda)x_{k_1}^t + \sum_{s=1}^{m} F_s\left(x_{k_{2,s}}^t - x_{k_{3,s}}^t\right) \qquad (\ 2.33\ )$$

In the literature, more than 10 different schemes have been formulated (Yang, 2020). There are also good variants of DE that include the self-adapting of the control parameters, and others for multi-objective optimization.

Differential Evolution (DE) algorithm is an evolutionary algorithm for optimization in continuous spaces. It can tackle non-linear and complex optimization problems, requiring just the objective function values. Nevertheless, the performance of the DE depends on the mutation control parameters, especially when the problem is complex (Brest, Greiner, Boskovic, Mernik, & Zumer, 2006). To balance the convergence (fitness evaluations) and the reliability (optimum's globality), ranges of parameters values have been studied. The most popular variant of DE is called "DE/rand/1/bin", where, as pointed out in the previous section, "DE" stands for Differential Evolution, "rand" means that the individuals selected to compute the mutation values are randomly chosen, "1" is the number of pairs of individuals chosen for mutation, and "bin" denotes the binomial crossover. Another variant is based on the best selection strategy: it is called "DE/best/1/bin", because the perturbing individual is generated from the best population member. It is known that "DE/rand/1/bin" is slow but robust compared to the strategies based on the best member. Among the most sensitive parameters, the Crossover Rate (*CR*) is a probability of mixing between mutant (donor) and target vectors of the current population (Zaharie, 2009). Low/high *CR* values are good for uni/multi-modal problems. Good convergence can be achieved with large *CR* values. Recommended *CR* values are in $[.2, .9]$. The differential weight $F \in [0, 2]$ controls the mutant vector: large *F* values allow escaping from local optima; low values cause premature convergence; $F \leq 1$ determines a fast and reliable search process. As a result, *F* is usually set in $[0.4, 0.9]$. The population size (*NP*) is another important parameter. In the literature, there is a lack of sufficient justifications and a lot of conflicting motivations about the manual parameter tuning of DE. To solve the issue, in our research a grid search technique is used.

2.2 Hyper-heuristics to design heuristics

Although adaptive, the logic of bio-heuristics is nevertheless constrained by models of biological species and can be neither modularized nor aggregated. A novel design approach based on hyper-heuristics (HH) can be used to overcome these limits. A hyper-heuristic is a search method that automates the combination of modular heuristics to generate more adaptable logics. In (Burke, et al., 2013) a unified classification and definition of HH able to capture the research work in the field has been presented. The authors define HH as a search method or a learning mechanism to select or generate heuristics solving search problems. Specifically, in a *learning* HH a feedback is given from the search process. In *online* learning HH the learning occurs while solving the problem, whereas in *offline* learning HH knowledge is gathered from training instances and modelled as rules or programs. Considering the type of search space, the *heuristic selection* chooses or selects predefined heuristics, whereas the *heuristic generation* generates new heuristics from modular components. Both search paradigms can be further divided into *constructive*, when iteratively extending partial candidate solutions with missing components and *perturbative*, when adjusting full candidate solutions by modifying their components. A comprehensive classification of hyper-heuristic approaches is represented in Table 1.2.

In the literature, hybrid approaches are also used. In particular, Garrido *et al.* (Garrido & Riff, 2010) have solved the dynamic vehicle routing problem via an evolutionary HH. Their framework is based on a combination of both constructive and perturbative HH and is evaluated on a large and complex set of problems. Results are competitive with respect to well-known methods of the literature. The HH approach aims to provide a general method for many application domains, rather than a better solution to a specific problem. Indeed, the search space of HH is a space of new heuristics, rather than a space of solutions. The difference is that a new heuristic can be potentially reused for solving many problem instances. In the literature, a well-known method for generating heuristics is genetic programming (Poli & Koza, 2014). It is an evolutionary computation technique evolving a population of computer programs. Genetic programming can be considered as an HH if the evolved programs are heuristics or heuristic's components. For example, in (Geiger, Uzsoy, & Aytug, 2006) Geiger *et al.* have illustrated the main motivations to automatically generate heuristics in production scheduling. The research in the field has shown also that successful components can be derived by the available human-created heuristics (Fukunaga, 2008). Another research field, related to perturbative heuristics, is called *adaptive memetic algorithms*. It is an evolutionary algorithm characterized by local searchers called memes, adaptively selected/generated during the search (Ong, Lim, Zhu, & Wong, 2006).

2.2.1 Selection constructive hyper-heuristics

In an optimization problem, selection constructive hyper-heuristics select a low-level heuristic at each point in the construction of a solution. Low-level constructive heuristics aim to construct a complete solution, or an initial solution for optimization. The solution of a problem starts from an initial state, going through a certain number of intermediate states until it reaches the final state or solution state.

In order to move from one state of the problem to the next, selection constructive hyper-heuristics select the low-level constructive heuristics to be applied. The domain of the problem determines the low-level heuristics. A formal definition of constructive selection hyper-heuristics is provided in **Definition 2.1**.

Definition 2.1: given a problem p and a set of low-level construction heuristics $L = \{L_0, L_1, \dots, L_n\}$ for the problem domain, a selection constructive hyper-heuristic constructs a solution s for p by using a technique T to select a low-level heuristic from L and by applying this low-level heuristic to change from one problem state $s^{(t)}$ to the next state $s^{(t+1)}$, beginning at the initial state and stopping at the solution state s.

Hyper-heuristics generally exploit a high-level technique such as meta-heuristics or case-based reasoning to select low-level heuristics. For solving a combinatorial optimization problem, usually the algorithm that is used by a selection constructive hyper-heuristic is outlined in **Algorithm 2.6**.

Algorithm 2.6
Selection constructive hyper-heuristic

```
procedure $SelectionConstructiveHyperHeuristic(p, L)$
    initialize solution $s$ to be empty;
    while $solution\ s\ is\ not\ completely\ constructed$
        use technique $T$ to select a low-level constructive heuristic $L_i$ from $L$;
        apply $L_i$ to extend the solution $s$;
    end while
    return solution $s$;
end procedure
```

The categories of techniques employed by selection constructive hyper-heuristics to select low-level heuristics include case-based reasoning, local search methods, population-based methods, and hybridization and adaptive methods.

Population-based search methods explore multiple points simultaneously, in contrast to local search methods that move from one point in the search space to the next. The population of solutions represents different points in the search space. Evolutionary algorithms have mainly been used in the literature to explore the heuristic space. In these cases, genetic operators are applied to combinations of heuristics and thus perform exploration and exploitation in the heuristic space. The performance of an EA depends, in turn, on the configurations of the hyper-parameters: for example, mutation and crossover probabilities, population size, and number of generations. Parameter control methods have been proposed, such as deterministic, adaptive, and self-adaptive methods (Smith, 2008).

Each element in the population, i.e., the chromosome, is a combination of heuristics. The combination includes low-level constructive heuristics, and each heuristic in the combination represents a heuristic selected by the selection constructive hyper heuristic. The selection as such is performed by the genetic algorithm through the process of fitness evaluation, selection, and recombination. Each chromosome is applied to solve one or more instances of the problem, and the fitness is the objective value in the case of a single instance of the problem or a function of the objective values of different instances of the problem. When the source of feedback is the evaluation of a problem instance, the goal is to evolve a combination of heuristics specific to the problem at hand. In this case, the combination of heuristics is disposable, i.e., the combination of heuristics or the evolved rule is used to solve an instance of the problem. In contrast, when the source of feedback is the evaluation of more than one instance, the goal is to evolve a reusable combination of heuristics. In this case, the problem instances are divided into training and testing sets. The training set is used to evolve the combination of heuristics, while the testing set is a set of unseen problems on which the evolved combination of heuristics is tested.

The representation used by the population-based approach affects the performance of the hyper-heuristic. The simplest representation of a chromosome is a single combination of low-level heuristics of a specific type. A chromosome may also include more than one type of low-level heuristics. The set of low-level heuristics to be used to compose the combinations of heuristics must be chosen carefully. A large set with less useful constructive heuristics may lead to a search space that is too large to explore an optimal combination of heuristics in a limited runtime. In addition to low-level constructive heuristics, the heuristic space may alternatively consist of condition-action rules, where the condition represents the states of the problem and the action the corresponding heuristic to be applied. In the literature, it has also been shown that different low-level constructive heuristics are needed at different points in the construction of a solution, i.e., a different heuristic is needed for each state of the problem from the initial state to the solution state. Adaptive methods have been shown to be effective in tailoring hybridizations or different types of constructive heuristics at different stages of the construction of solution (Pillay & Qu, 2018).

2.2.2 Generation constructive hyper-heuristics

In solving optimization problems, a low-level constructive heuristic is used to create an initial solution, which is a starting point for solving the problem using optimization techniques. These heuristics are usually problem dependent. In fact, research has shown that different low-level constructive heuristics are effective for different classes of problems, and for some problem domains it is more effective to generate heuristics suitable for each instance of the problem (Lu, Xin, Zhang, & Chen, 2020) (Kahar & Kendall, 2010) (Felipe, Ortuño, Righini, & Tirado, 2014) (Paquay, Limbourg, & Schyns, 2018). Deriving constructive heuristics is a time-consuming process due to the relevant heuristic selection and parameterization costs associated with each new problem type and new instances of known problems. Thus, deriving low-level constructive heuristics becomes expensive to do manually (Drake, Hyde, Ibrahim, & Ozcan, 2014). Constructive hyper-heuristics generation aims to automate this process by generating low-level constructive heuristics by using a given set of problem attributes. Automating this process reduces the human hours involved in deriving low-level heuristics and can result in the generation of new constructive heuristics that humans would not think of. This allows the constructive heuristic to be tailored to a particular instance of the problem or to be engineered for different classes of problems. Thus, the heuristics generated can be disposable, i.e., created for a specific problem instance, or reusable, i.e., used to solve similar problems never seen before (Burke, et al., 2013).

Two criteria should be used to evaluate the performance of the generation constructive hyper-heuristics, i.e., the time needed to generate the heuristics and the performance of the generated heuristics compared to existing manually derived heuristics. The time required by the generation constructive hyper-heuristics should be less than the time needed to manually derive these heuristics (Burke, Hyde, Kendall, & Woodward, 2010). Moreover, the performance of the generated low-level heuristics cannot be expected to be comparable to the state of the art for the specific problem domain (Drake, Hyde, Ibrahim, & Ozcan, 2014). Similar to manually derived heuristics, the purpose of these heuristics is to provide a starting point for optimization techniques. Thus, automatically generated heuristics should perform at least as well as manually derived heuristics. However, research in this field to date suggests that heuristics created by the generation constructive hyper-heuristics have been shown to outperform existing heuristics (Pillay & Qu, 2018). Another important issue concerns the interpretability of the generated constructive heuristics, i.e., whether it is needed for the generated heuristic to be readable to understand what it is doing, or the generation constructive hyper-heuristics should work as a black box.

A formal definition of generation constructive hyper-heuristics is outlined in **Definition 2.2**.

Definition 2.2: given a problem instance i or a set of problem instances $I = \{I_0, I_1, \dots, I_m\}$ and a set of problem attributes $A = \{A_0, A_1, \dots, A_n\}$ for a problem domain, a generation constructive hyper-heuristic generate a new low-level constructive heuristic lch, using the attributes in A, to produce an initial solution for either i or the problems in I and similar problems.

In solving combinatorial optimization problems, the low-level derived heuristic is fundamentally a priority function that is used to order events or entities to be chosen to create a solution. As such, the low-level derived heuristic is an arithmetic function or rule composed of attributes and operators. Genetic programming (Poli & Koza, 2014) and its variants have been primarily used to generate these low-level heuristics. Hyper-heuristics reach generalization by using the same technique to derive heuristics for different domains and instances of the problem, with the only difference being the set of attribute values A

used, which depends on the problem. However, the derived low-level heuristic may or not be generalized, i.e., it may be reusable or disposable.

The generated low-level heuristic includes problem attributes and operators. Thus, the methods for creating low-level heuristics combine or configure the attributes and the operators in some way. It is important that an appropriate set of attributes is chosen and that all features of the problem domain are represented. However, including too many attributes will result in a larger heuristic space, which can lead to high processing times or not finding suitable heuristics. According to (Branke, Nguyen, Pickardt, & Zhang, 2016), the attributes should be in their most basic form, and it should be left to hyper-heuristics to create aggregate features by combining them. The attributes for a problem domain also include the components of existing constructive low-level heuristics. In fact, the basic components that constitute existing constructive low-level heuristics may be more representative of the problem domain than the heuristic as a whole. Thus, existing low-level heuristics are decomposed into basic components, and these are used as attributes. For example, in the methodology based on hyper-heuristic described in this work, two bio-inspired meta-heuristics have been decomposed with respect to the pheromone model used in the stigmergic communication mechanism and the implementation rules for flocking behavior.

Genetic programming has been employed by generation constructive hyper-heuristics to generate new low-level constructive heuristics. Genetic programming is an evolutionary algorithm that explores a program space rather than a solution space (Poli & Koza, 2014). Programs can represent arithmetic functions or algorithms that, when executed, will produce a solution to the problem at hand. Each program is represented as an expression tree. For a combinatorial optimization problem, the algorithm generally employed by a generation constructive hyper-heuristic to solve the problem at hand is outlined in **Algorithm 2.7**.

The algorithm starts with an initial population of programs, each of them is an expression tree representing a new constructive heuristic. A fitness function is applied to evaluate each program in the population, i.e., how good the program is at solving the problem at hand. In the case of the evolution of constructive heuristics, the fitness of each expression tree is calculated by the resulting solution created using the program tree. A selection method chooses parents based on their fitness to create the offspring of the next generations. Tournament selection is generally used for genetic programming (Poli & Koza, 2014). Genetic operators that include selection, mutation, and crossover, are usually applied to parents to create next generation offspring.

Algorithm 2.7
Generation constructive hyper-heuristic using genetic programming

```
procedure $GenerationConstructiveHyperHeuristic(I, A)$
    create an initial population of programs;
    while $termination\ criteria\ are\ not\ met$
        evaluate each program in the population;
        select parents;
        apply genetic operators to the parents to create offspring of
        the new generation;
    end while
    return $lch$ and solution $s$;
end procedure
```

Chapter 3

3 A hyperheuristic-based methodology for robotic swarms coordination

Despite the success of bio-inspired techniques (bio-heuristics), there are relevant algorithm selection and parameterization costs related to every new type of mission and to new instances of known missions. In this work an evolutionary optimization is described to automate the tuning of the bio-inspired coordination for target search. Experimental results on real-world scenarios reveal a significant improvement of the mission performance after optimization.

Although adaptive, the logic of bio-heuristics is nevertheless constrained by models of biological species, and then, for example, it can be neither modularized nor aggregated. In order to overcome these limits, a novel design approach based on *hyper-heuristics* (HH) is adopted. This is a search methodology that automates the combination of modular heuristics to generate more adaptable logics: fundamental behavioral components for many biological swarms are aggregated and tuned in a unique and continuous search space. Two fundamental swarm behavioral components are considered: *stigmergy* and *flocking*. Stigmergy is used to release an attractive – or repulsive – stimulus when detecting the presence – or absence – of a target during exploration. Multiple stimuli can overlap, creating a stigmergic trail which, in turn, evaporates over time. As a result, stigmergy creates a kind of context-aware memory of the swarm (Cimino, Lazzeri, & Vaglini, Improving the Analysis of Context-Aware Information via Marker-Based Stigmergy and Differential Evolution, 2015). Flocking is used to model a robust and flexible swarm formation. It is based on the rules of cohesion, separation, and alignment (Alfeo, et al., Swarm coordination of mini-UAVs for target search using imperfect sensors, 2018). Depending on the type of mission and on the environment layout, flocking of different sizes and flexibility can be adaptively modelled.

The *Differential Evolution* (DE) algorithm optimizes the aggregation and tuning of the heuristics on a unique search space and, consequently, an efficient heuristics hybridization is generated for a given application domain. DE is a population-based metaheuristic optimization algorithm, based on computational mechanisms of biological evolution, such as reproduction, mutation, recombination, and selection of solutions. DE can tackle non-linear and complex optimization problems, requiring just the objective function values. A modeling and optimization testbed has been developed and publicly released (Monaco, 2021). Experimental results on real-world scenarios show that the proposed approach, called SFE because it is based on Stigmergy, Flocking, and Evolution, significantly outperforms the adaptive bio-heuristics.

3.1 Design

We consider a novel algorithmic design based on hyper-heuristics. In this approach, the logic is not constrained by models of biological species. It consists of an optimization method of fundamental functional components, whose aggregation and tuning are represented on a unique and continuous search space. Specifically, we consider two fundamental swarm behavioral components as bio-inspired heuristics, namely stigmergy and flocking (Alfeo, Cimino, & Vaglini, 2019). The differential evolution algorithm (DE) is adopted to optimize the aggregation and tuning of the heuristics on a problem of target search. The quality measure of a target search is the time needed for completing the mission, i.e., for discovering a given percentage of target (Alfeo, Cimino, De Francesco, Lega, & Vaglini, 2018). Consequently, the fitness of the DE is defined as the mission duration.

More formally, we consider a simulated scenario Ω, composed by:

i. simulation instants of time $t \in \mathbb{N}^+$;

ii. a set of robots R, each robot k having a dynamic position (x_k^t, y_k^t);

iii. a set of targets T, each target z having a fixed position (x_z, y_z).

Hence, the set of found targets $F(t) \subseteq T$, at a given instant of time t, is the set of targets $\{z\} \in T$ for which it exists a time $t' \leq t$ and a related set of robots $\{k_{i,z}\}$, with $i = 1, \dots, N_{min}^R$ and N_{min}^R the number of robots needed to process each target, such that the robots' Euclidean distances from the target position is lower than the detection range δ, i.e.:

$$F(t) = \left\{ z \mid \exists\, k_{i,z}, \exists\, t' \leq t : d\left[\left(x_{k_{i,z}}^{t'}, y_{k_{i,z}}^{t'} \right), (x_z, y_z) \right] \leq \delta \right\} \qquad (3.1)$$

The fitness of the simulated scenario Ω is then defined as the minimum instant of time for which $F(t)$ has cardinality greater than or equal to $\vartheta \cdot |T|$, i.e.:

$$fitness(\Omega) = min_{t \in \mathbb{N}^+}\{t : |F(t)| \geq \vartheta \cdot |T|\} \tag{3.2}$$

where ϑ is a percentage threshold close to 1 (usually set to 0.95), used to reduce the simulation duration without sensibly affecting the overall accuracy. In order to better explain the use of the threshold percentage ϑ, we can consider the following Figure 3.1 that shows the number of targeted cells found (%) against time (sec.).

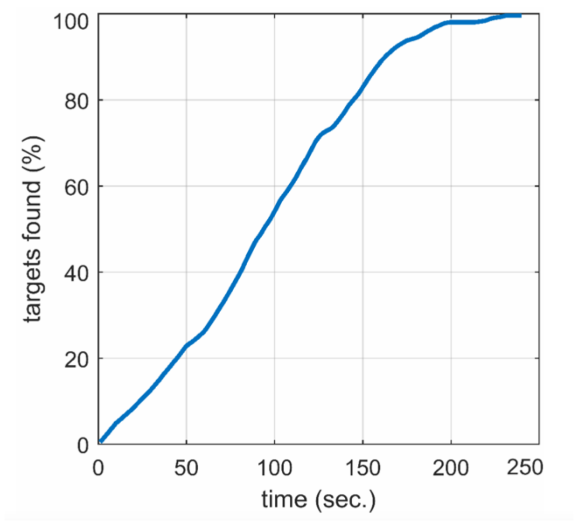


Figure 3.1 - Percentage of targets found against time

The plot indicates a constant trend of targets found per second, up to about 95%. Specifically, the scenario could include up to 5% of targets located in areas that are difficult to access and whose detection may result in a significant deterioration in the performance measure. Since this is commonly a point of trend variation, to shorten the simulation duration the target threshold value ϑ is set to 95%.

3.1.1 Flocking-based exploration

Flocking is used to model a robust and flexible swarm formation. It is based on the rules of cohesion, separation and alignment, as illustrated in Figure 3.2.

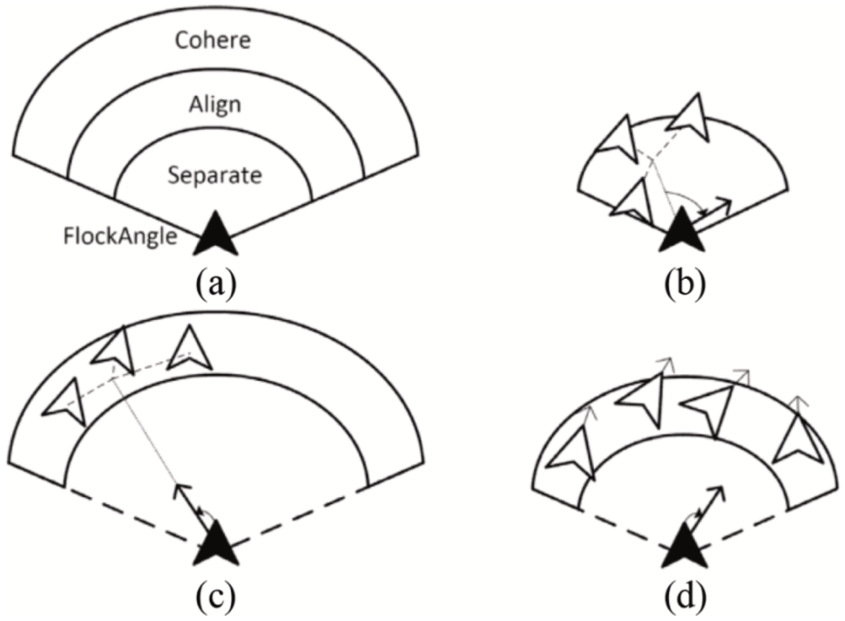


Figure 3.2 - Model of flocking behavior: (a) activation regions, (b) separation, (c) cohesion, (d) alignment

The different rules are activated in separate regions, as in Figure 3.2(a). The separation rule, showed in Figure 3.2(b), maintains a certain distance among flock mates for a better scan of the area. The cohesion rule, showed in Figure 3.2(c), directs the robot to the flock center, to avoid dispersion. Finally, the alignment rule, showed in Figure 3.2(d), keeps the heading of each robot aligned to the average heading of its flock mates. Depending on the type of mission, flocking of different sizes can be modelled.

3.1.2 Stigmergy-based coordination

Stigmergy is used to release an attractive (or repulsive) stimulus (pheromone) while (not) detecting targets. In the adopted computational model, a digital pheromone mark is released by the robot in the environment. Figure 3.3 illustrates the model of the pheromone mark: it is a truncated cone with unit height, radius top and down.

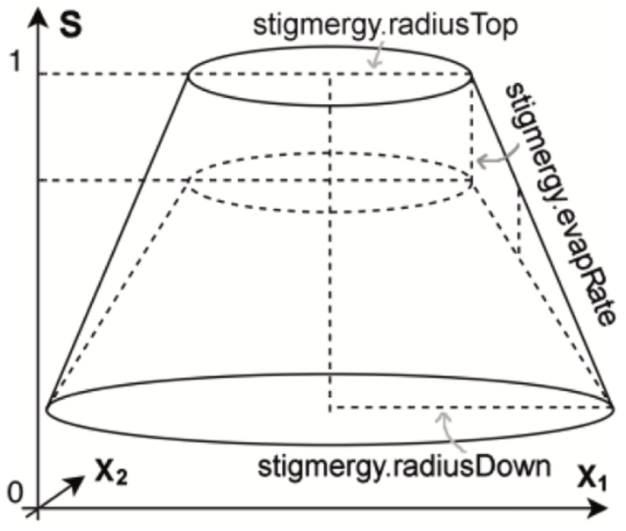


Figure 3.3 - Model of a pheromone mark

Multiple pheromone marks can overlap, creating a pheromone trail. Pheromone trails evaporate over time. Since the pheromone trail is maintained in a digital environment, it is instantly diffused, to immediately propagate information to nearby robots. More formally, let us consider the target z detected by the robot k at time t, with position $(x_z^t, y_z^t) \in W$, and $W \subset \mathbb{R}^2$ that is the exploration area (in the computerized model it is actually a discretized area $\mathbb{N}^2$). The pheromone quantity $\Delta\varphi_{k,c}^t$ released on the cell c located in (x_c, y_c) is given by:

$$\Delta\varphi_{k,c}^t = \begin{cases} 1 & if\ d_{zc} \leq r_{top} \\ \frac{d_{zc}-r_{down}}{r_{top}-r_{down}} & if\ r_{top} < d_{zc} < r_{down} \\ 0 & otherwise \end{cases} \qquad (3.3)$$

where r_{top} and r_{down} are $stigmergy.radiusTop$ and $stigmergy.radiusDown$, respectively, and d_{zc} is the Euclidean distance between the target z and the cell c. The pheromone trail intensity in the cell c at time t is given by:

$$\varphi_c^t = max\left\{0, min\left\{\varphi_{max}, \varphi_c^{t-1} - e_{rate} \cdot \varphi_c^{t-r} + \sum_{k=1}^{N_R} \Delta\varphi_{k,c}^t\right\}\right\} \qquad (3.4)$$

where r is the time elapsed since the last pheromone release and φ_c^{t-r} is the trail intensity on the cell c at the time $t - r$ when the last pheromone mark has been released, e_{rate} is the evaporation rate, i.e. the given amount of intensity evaporated per unit time, and N_R is the number of robots that are able to release pheromone on the cell c. The model with a linear evaporation and a streamlined shape allows a good control of the aggregated trail in the parameter space. The perceived pheromone intensity is based on olfactory receptors, which can decrease in sensibility over time to prevent overstimulation (olfactory habituation).

3.1.3 Stigmergy-Flocking-Evolution (SFE) algorithm

An efficient heuristics hybridization is generated for a given application domain, in which Differential Evolution minimizes the mission discovery time. The resulting algorithm is called SFE, which stands for Stigmergy, Flocking, Evolution. Since SFE can adapt his behavior to the problem, it can be used for both environment exploration and targets resolution. More specifically, a repulsive pheromone is used as an anti-stimulus during the exploration, while an attractive pheromone is used as a stimulus for collecting sufficient target information. During exploration, the robot adopts an if-then-else approach. In fact, it turns primarily to the maximum attractive pheromone, if detected, else follows the flocking rules, if flock mates are detected; otherwise, it turns to the minimum repulsive pheromone.

More formally, the DE logic is summarized by the pseudocode presented in **Algorithm 3.1**. Moreover, **Algorithm 3.2** and **Algorithm 3.3** define the mutation and the crossover operators, respectively.

In a simulated scenario (or mission), the swarm S_i explores an environment where robots, obstacles and targets are statically specified. Let K be the number of aggregated parameters. In DE, S_i is a solution represented by a real K-dimensional vector called genotype p_i. The search time returned by the simulated mission is used as a fitness of the solution, f_i. DE starts with a population $P^{(0)}$ made by N candidate solutions, $p_i^{(0)}$, randomly generated under user-specified parametric constraints. At each iteration t, and for each genotype p_i^t of the current population $P^{(t)}$, a mutant vector m is created by applying the mutation of randomly selected members. Then, a trial vector p_i^* is created by crossover of m and p_i^t. In the binomial crossover algorithm (**Algorithm 3.3**), K represents the number of aggregated parameters to be optimized. Then, the population is modified by selecting the best fitting vector between the fitness of the trial vector (f_i^*) and the fitness of the initial genotype ($f_i^{(t)}$). When the termination criterion is true, i.e., number of iterations performed or adequate fitness reached, the vector characterizing the swarm with the best fitness (i.e. the shortest search time) in the current population is considered as the optimal swarm parameterization. The DE algorithm has at least two hyper-parameters: the scaling factor $F \in [0, 2]$ from which results the mutant vector, and the crossover probability CR. The smaller CR the higher the probability of producing a vector that is more similar to the target vector rather than to the mutant vector.

Algorithm 3.1
Differential Evolution algorithm

```
**function** $differentialEvolution(Robots, Obstacles, Targets)$
$t = 0;$
$P^{(0)} = initializePopulation();$
**for each** $genotype\ p_i^{(0)}$ **in** $P^{(0)}$ **do**
    $S_i^{(0)} = genotypeToSwarm(p_i^{(0)});$
    $f_i^{(0)} = simulateMission(S_i^{(0)}, Robots, Obstacles, Targets);$
**do**
    **for each** $genotype\ p_i^{(t)}$ **in** $P^{(t)}$ **do**
        $m = generateMutant(P^{(t)}, p_i^{(t)});$
        $p_i^* = binomialCrossover(p_i^{(t)}, m);$
        $S_i^* = genotypeToSwarm(p_i^*);$
        $f_i^* = simulateMission(S_i^*, Robots, Obstacles, Targets);$
    **for each** $genotype\ p_i^{(t)}$ **in** $P^{(t)}$ **do**
        **if** $(f_i^* < f_i^{(t)})$ **then**
            $p_i^{(t+1)} = p_i^*;$
            $f_i^{(t+1)} = f_i^*;$
        **else**
            $p_i^{(t+1)} = p_i^{(t)};$
            $f_i^{(t+1)} = f_i^{(t)};$
    $f_{min}^{(t+1)} = min\{f_1^{(t+1)}, \dots, f_N^{(t+1)}\};$
    $t = t + 1;$
**while** $(terminationCriterion(f_{min}^{(t)}, t) = false);$
**return** $genotypeToSwarm(p_{min}^{(t)});$
```

Algorithm 3.2
Mutation for DE/rand/1/bin

```
**function** $generateMutant(P^{(t)}, p_i^{(t)})$
$p' = randomExtraction\left(P^{(t)} \setminus \{p_i^{(t)}\}\right);$
$p'' = randomExtraction\left(P^{(t)} \setminus \{p_i^{(t)}, p'\}\right);$
$p''' = randomExtraction\left(P^{(t)} \setminus \{p_i^{(t)}, p', p''\}\right);$
**return** $p' + F \cdot (p'' - p''');$
```

Algorithm 3.3
Binomial crossover

```
function $binomialCrossover(p_i^{(t)}, m)$
$k = randomInteger(1, K)$;
for each $j$-th gene $p_{j,i}^{(t)}$ in $p_i^{(t)}$ do
    if $(randomReal(0,1) < CR)$ $or$ $(j = k)$ then
        $w_j = m_j$;
    else
        $w_j = p_{j,i}^{(t)}$;
return $w$;
```

Table 3.1 shows the parameter space of the SFE algorithm, i.e. the variable parameters to be tuned to a target search mission, together with the corresponding units of measurement in a real world scenario.

Table 3.1 - Parameter space of the SFE algorithm

Parameters	Unit of measurement
stigmergy.radiusTop	meters
stigmergy.radiusDown	meters
stigmergy.evapRate	percentage
stigmergy.olfactoryHabituation	seconds
stigmergy.repulsiveRadius	meters
stigmergy.repulsiveEvapRate	percentage
flocking.angle	degrees
flocking.wiggleVar	degrees
flocking.radiusSeparate	meters
flocking.maxSeparateTurn	degrees
flocking.radiusAlign	meters
flocking.maxAlignTurn	degrees
flocking.radiusCohere	meters
flocking.maxCohereTurn	degrees

It is worth to be noted that when the parameters *stigmergy.radiusTop, stigmergy.radiusDown*, and *stigmergy.repulsiveRadius* are very small, then attractive and repulsive stigmergy are very low too. Similarly, if *flocking.angle* in Figure 3.2(a) is very small, then flocking is very low since no flock mate is visible. Thus, such parameters can lower/raise the contribution of each component in the overall workflow, in a continuous optimization space.

3.2 Simulation testbed

The methodology described in the section 3.1 has been implemented within a modeling and optimization testbed that has been publicly released (Monaco, 2021). The computational models related to attractive pheromone, repulsive pheromone, stigmergy-based behavior, and flocking behavior have been implemented by using NetLogo (Wilensky & Rand), that is the most popular toolkit for agent-based modeling in the socio-ecological modeling community. NetLogo is a java-based graphical environment for the programming of multi-agent systems, which allows to interact thousands of independent, heterogeneous and parallel agents and to reproduce in real time the dynamic aspect of the simulated phenomena. These characteristics allow, on one hand, the exploration of the agents' behaviors at the level of local interactions and, on the other hand, the analysis of the effects of such interactions at a global level. NetLogo represents the most popular simulation platform for swarm intelligence-based systems. The NetLogo programming language is designed to be easy to learn. The single statements of the language are represented by English words, and the statement sequences appear very similar to simple sentences. Furthermore, NetLogo offers a series of tools very useful for representing, during a simulation, the intrinsic characteristics of a model in execution, as for example monitor, graphs, fields to insert textual values, and so on. The strength that makes this language powerful is definitely the Java core. Basically, NetLogo's tools and the language used to describe the models constitute an intermediate layer between the user and the machine, which NetLogo automatically translates into fast and powerful Java programs.

In addition to environment and swarm algorithms, the testbed considers the robots sensing, actuation, and collision avoidance, by modeling drone size, battery duration, sensing radius, sensing angle, collision vision, collision angle, angular speed, acceleration, and cruise speed. Scenarios of different complexity have been considered.

In Figure 3.4 it is shown an ongoing scenario of target search. We consider a swarm of mobile robots, or drones, deployed in an exploration area, in order to search and process the targets cooperatively. We assume that the environment is unstructured, i.e., obstacles or targets number and locations are unknown. In Figure 3.4 obstacles are represented in grey color. Drones are depicted as green arrowheads, and undetected/detected targets as red/yellow points. Finally, an attractive/repulsive pheromone is represented as blue/pink continuous intensity.

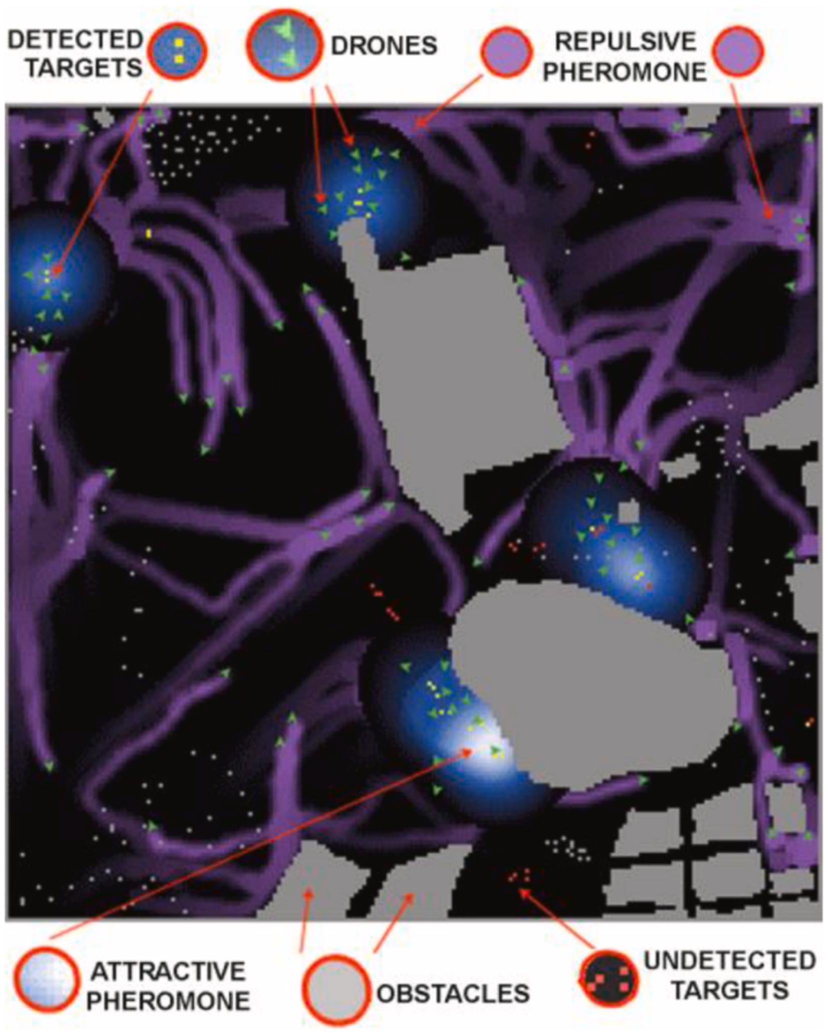


Figure 3.4 - Environment: drones, targets, attractive and repulsive pheromones, obstacles

The swarm is also divided into flocks: flexible, dynamic, and autonomous groups communicating between themselves (flock mates) and self-organizing, splitting around obstacles, rejoining, and avoiding collisions with each other. Moreover, an attractive pheromone released by flock mates creates a short-medium term potential to compact the flock where multiple targets are detected. In contrast, a repulsive pheromone helps the drones to avoid multiple exploration of the same area whereas new targets are not detected. Finally, olfactory habituation is another bio-inspired form of memory: when exposed to the maximum intensity of attractive pheromone, the sensing saturates and becomes unable to sense for a while, to leave the saturated area more efficiently.

The parametric optimization of the aggregated low-level heuristics has been performed using a Python language implementation of the Differential Evolution algorithm, available in the open-source SciPy library. In order to provide the parameter values to the NetLogo model and obtain the swarm performance, that, as stated earlier, represents the fitness value, we have used NL4Py (Gunaratne & Garibay, 2021), a controller software for Python, developed with the goals of usability, fast parallel execution, and access to model parameters. NL4Py uses a client-server architecture via Py4J, a Python-Java bridging software. Moreover, it parallelizes the execution of NetLogo workspaces, instead of leaving it to the user's python application.

The Figure 3.5 shows the overall software architecture of the released simulation testbed. The optimizer subsystem is fully developed in python language and includes the starting point of the hyper-heuristic methodology, identified by the file *de_with_nl4py.py*. The component uses the implementation of the differential evolution algorithm, that at each generation evaluates the population in parallel, exploiting all available CPUs. Each running objective function communicates with the NL4Py software through a singleton class *WorkspaceManager*, created according to the façade design pattern.

NL4Py consists of two main components, a client written in Python and a server written in Java. The client code communicates to the *NetLogoControllerServer* through a soket enabled by the Py4J library. The client-server architecture allows NetLogo headless workspaces (that are essentially NetLogo models running without the GUI enabled) to be run in parallel as Java threads on the *NetLogoControllerServer*. This allows users to not have to manage the connection to the JVM, thread/process creation, and garbage collection of multiple headless workspaces from their Python application code. NetLogo provides, through its controlling API, headless workspaces that can be controlled via Java or Scala application and are implicitly thread safe. NL4Py pushes concurrency to the JVM through the *NetLogoControllerServer*. The NL4Py Python client supplies thread safe *NetLogoHeadlessWorkspace* objects to the Python application developer, created according to the factory design pattern. Each running objective function created by the optimizer component is mapped to a *NetLogoHeadlessWorkspace* object. In turn, each *NetLogoHeadlessWorkspace* object is mapped to a *HeadlessWorkspaceController* object on the *NetLogoControllerServer*, which has the responsibility to start and stop the NetLogo model, to send commands to the model, to query parameters, and to fetch results from the model.

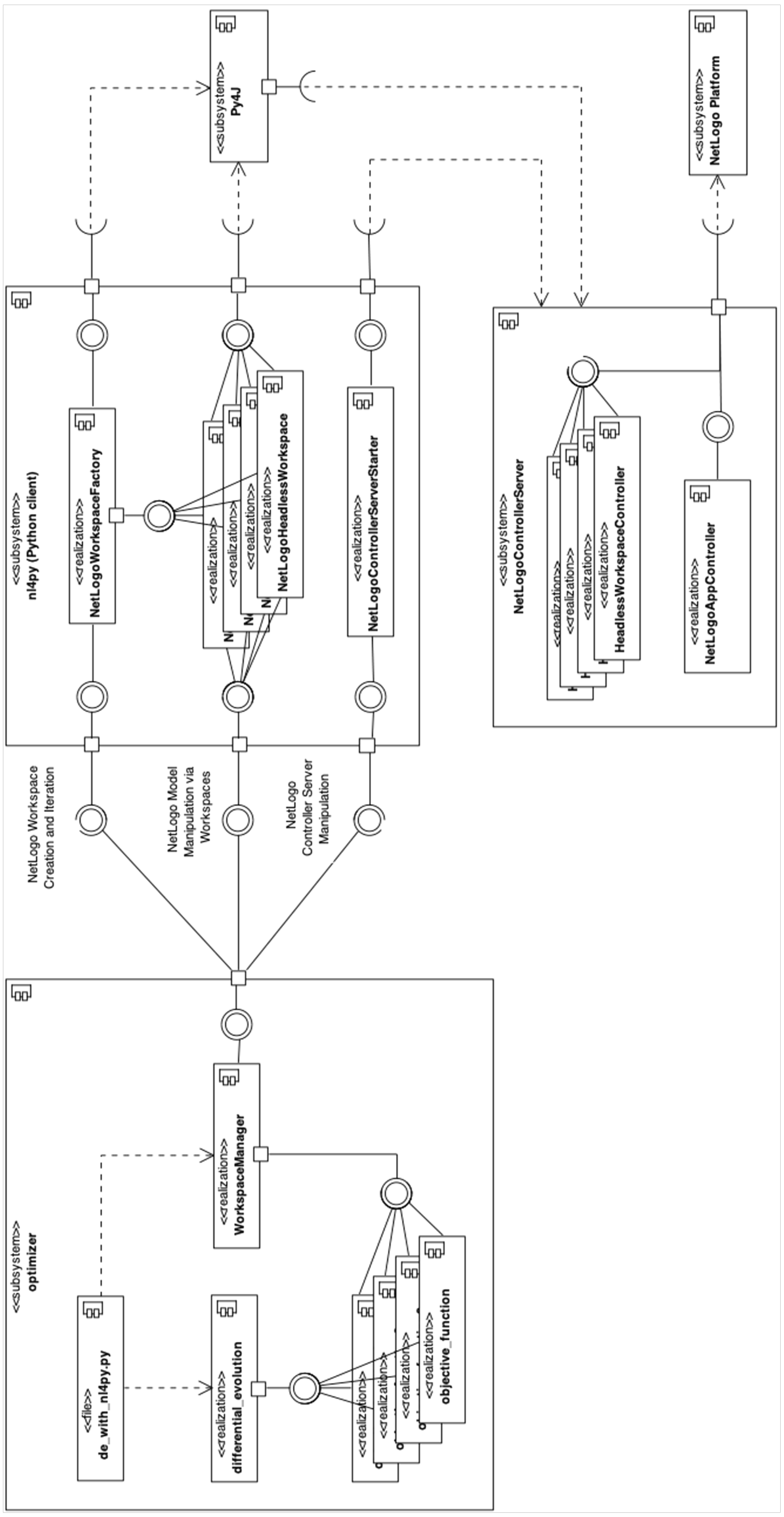
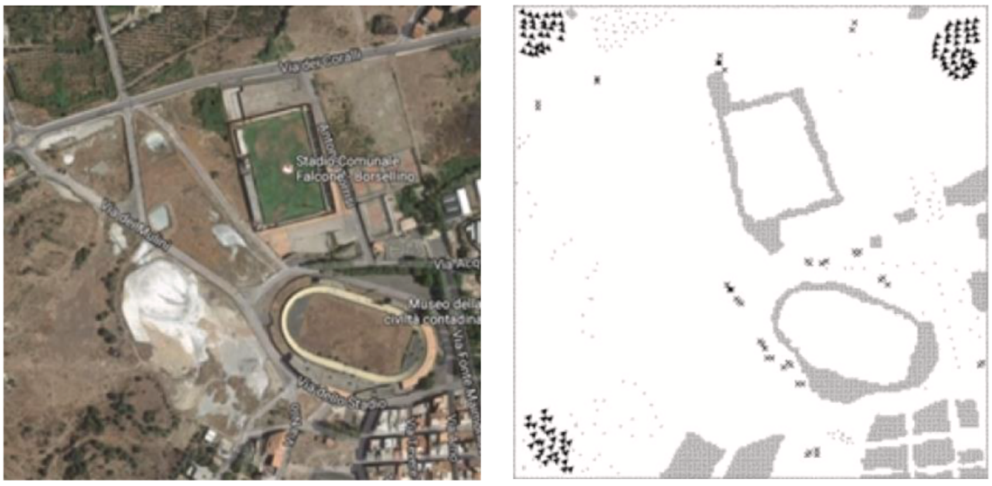


Figure 3.5 - UML component diagram of the simulation testbed SFE

3.3 Real-world scenarios

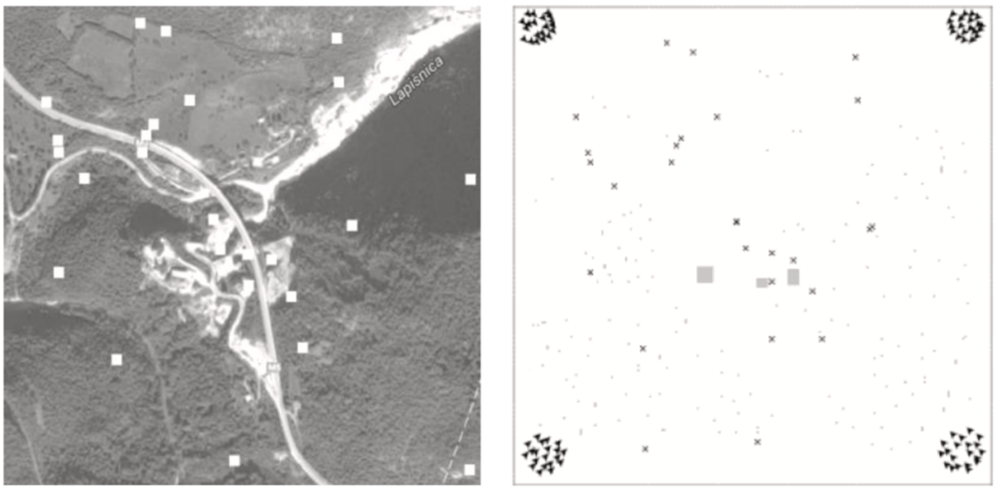
Real world scenarios of different complexity have been considered for the exploration simulator. The *Illegal Dump* scenario represents a real abusive trash map of 80,000 m^2 near the town of Paternò (Italy), and is composed by 11 groups of targets with an average number of 4 targets per group, 19 buildings of different sizes, 140 trees (www.trashout.me). Figure 3.6(a) and Figure 3.6(b) show the aerial photo and the corresponding vectorial model, respectively.


Via dei Coralli
Stadio Comunale Falcone Borsellino
Via dei Mulini
Museo della civiltà contadina
Via dello Stadio

(a) (b)

Figure 3.6 - Illegal dump scenario: (a) aerial photo (Google Maps ©), (b) vectorial model

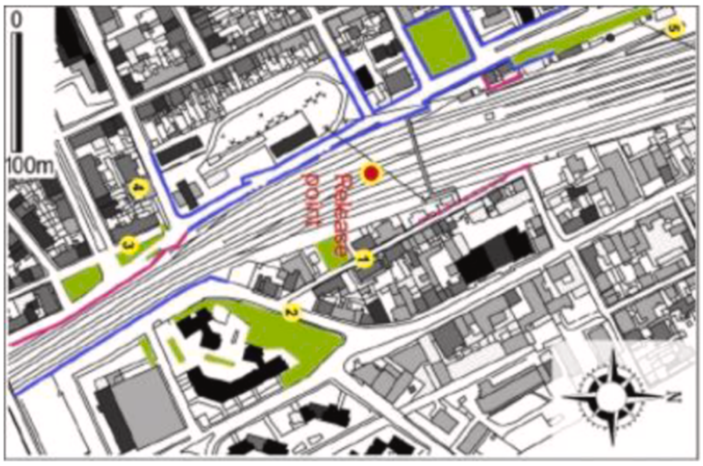
The *Rural Mine* scenario is a real-world example of areas with landmine objects in Bosnia-Herzegovina, described in public available data (www.seedemining.org). It is made up of 28 buildings, 59 trees and 40 targets. Figure 3.7(a) and Figure 3.7(b) show the aerial photo and the corresponding vectorial model, respectively.



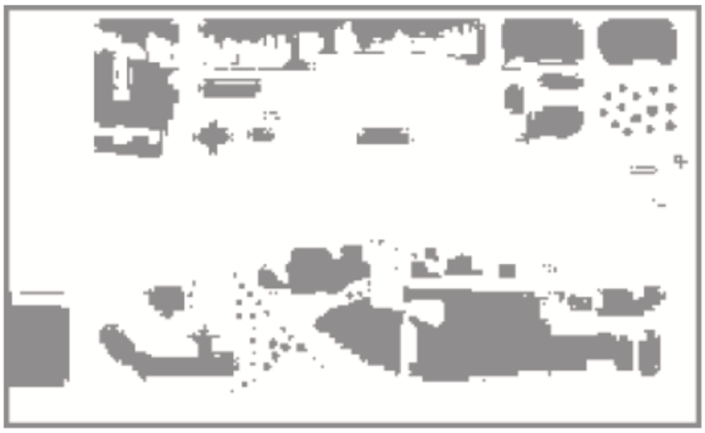
(a) (b)

Figure 3.7 - Rural mine scenario: (a) aerial photo (Google Maps ©), (b) vectorial model

The *LPG Leak* scenario is based on an accident caused by an LPG railcar rupture, which occurred in 2009 in the urban area of Viareggio, Italy (ref). Figure 3.8(a) and Figure 3.8(b) show the aerial map and the corresponding vectorial model, respectively.



(a)



(b)

Figure 3.8 - LPG leak scenario: (a) aerial map (Pontiggia, et al., 2011), (b) vectorial model

Chapter 4

4 Applications

In this chapter some experimental applications of the methodology described in the former chapter are detailed. The experimental results of these applications are published in an international journal paper and in peer reviewed international conferences papers. We start with the problem of target search via a swarm of robots and the comparison between some popular bio-inspired swarm algorithms, that have been made adaptive, and our SFE hyper-heuristic. It follows the problem of coordinating multiple Unmanned Aerial Vehicles (UAV) for distributed targets tracking, in different technological and environmental settings. Finally, we focus on the results of a simulated analysis concerning the mitigation of plastic pollution in oceans via a swarm of Unmanned Surface Vehicles (USV).

4.1 Comparison between SFE and adaptive bio-inspired meta-heuristics

In the target search problem via swarms of robots, in complex or open environments, the robots cannot exploit static information on layout and target's locations. Therefore, their coordination is fundamental for an efficient target discovery. To coordinate the swarm, the following popular bio-inspired swarm algorithms have been considered and made adaptive: Ant Colony Optimization, namely ACO (inspired by ants) for exploration; Firefly Team Strategy, namely FTS (inspired by fireflies), Particle Swarm Optimization, namely PSO (inspired by birds), and Artificial Bee Colony, namely ABC (inspired by honeybees) for recruitment. Parameter tuning for adaptation is performed via the Differential Evolution (DE) optimization. The DE is able to find the best algorithmic parameters of a bio-inspired algorithm for improving the mission performance. In order to overcome the design constraints of bio-inspired approaches, an approach based on hyper-heuristics is also considered. This approach is called SFE because it is based on Stigmergy, Flocking and Evolution. Experimental results on real-world scenarios, carried out and released as a public testbed, show that the SFE significantly outperforms the adaptive bio-heuristics, in both exploration and recruitment. The SFE is also faster in terms of optimization duration, although it requires more memory.

4.1.1 Problem statement

In a target search mission, a robot can assume two major roles: explorer and coordinator (Palmieri, Yang, De Rango, & Marano, 2017). The purpose of exploration is to discover new targets, whereas the purpose of coordination is to recruit the necessary number of follower robots to process the discovered target. Figure 4.1 shows a UML activity diagram with the overall workflow carried out by each robot involved in a target search mission.

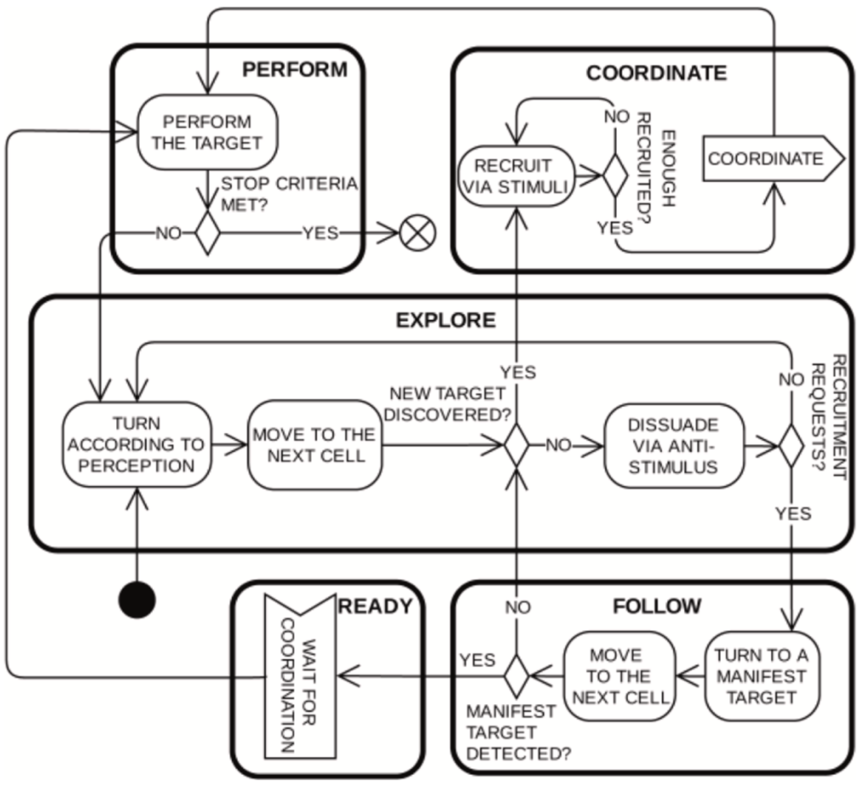


Figure 4.1 - Target search mission: UML activity diagram with the overall workflow

The process begins at the black start circle and ends at the white circle with a cross inside. The major activities are represented by bold round-cornered rectangles, and are connected by the following two core flows:

1) $explore \rightarrow coordinate \rightarrow perform$
2) $explore \rightleftarrows follow \rightarrow ready \rightarrow perform$

Each activity is defined as a workflow of generic tasks. The implementation of a task can vary depending on the bio-inspired approach (e.g., ACO for exploration; FTS, PSO, and ABC, for recruitment (Palmieri, Yang, De Rango, & Marano, 2017). It follows the description of each activity:

- *Explore*: the robot explores the area for discovering targets. First, it is oriented by its perception of a medium depending on the biological model; then, it moves to the next cell. If a new target is discovered, it *coordinates*; otherwise it dissuades from following its recent path by releasing some anti-stimulus. Finally, if no recruitment request arrives, the robot continues to *explore*, otherwise it *follows*.

- *Follow*: the robot is recruited by a coordinator robot. It selects one of the manifest targets, then turns to it, and moves to the next cell. When the manifest target is detected, the drone waits for coordination (*ready*).

- *Coordinate*: the robot becomes a coordinator when it detects a target, and after it starts to recruit the needed robots. The recruitment is based on stimuli depending on the bio-inspired approach.

- *Ready*: once reached the target, a recruited robot waits until the coordinator delivers the authorization to *perform* the target.

- *Perform*: the target is processed by all recruited robots. Then, a stop criterion is checked, e.g. a maximum time or a maximum percentage of targets found.

Bio-inspired techniques require relevant algorithm selection and parameterization costs associated with every new type of mission and with new instances of known missions. We adopt DE for the parametric adaptation of the bio-inspired exploration and recruitment algorithms on target search. According to our approach, the DE finds the optimum in the parametric search space of the bio-inspired algorithm which, in turn, solves a target discovery problem in a bidimensional space, via exploration and recruitment of robots (Cimino, Lega, Monaco, & Vaglini, 2019). Thus, for each bio-inspired algorithm, a corresponding adaptive variant is adopted.

4.1.2 Experimental setup

Considering the ACO for exploration, the FTS, PSO and ABC for recruitment, the names of the corresponding variants of the algorithms are: ACO-E, FTS-E, PSO-E and ABC-E, where the term "E" stands for Evolution. An algorithm variant solving both exploration and recruitment tasks includes two acronyms and is parameterized in a search space that is the union of the two search spaces. Since the recruitment problem varies significantly in complexity depending on the number of robots needed to process a target, namely N_{min}^{R}, a suffix "RR*" is added to highlight the different complexity. For example, "RR3" means an algorithm for recruitment with $N_{min}^{R} = 3$, whereas no suffix means an algorithm without recruitment, in other words an exploration algorithm. When both exploration and recruitment problems are considered, the name of the algorithmic solution includes both acronyms. For example, ACO-ABC-RR3-E is an algorithm that involves ACO for exploration, ABC for recruitment, both adapted through the DE, and in which at least three robots are required to process a single target. Table 4.1, Table 4.2, Table 4.3, and Table 4.4 show the parametric spaces of ACO-E, used only for exploration task, ACO-FTS-RR3-E, ACO-PSO-RR3-E and ACO-ABC-RR3-E, used for both exploration and recruitment tasks, respectively. In Table 4.2 the value L represents the maximum length between dimensions of the simulated scenario. In Table 4.3, the PSO algorithm considers only the global best particle, that, in this case, is represented by the location of the coordinator robot.

Table 4.1 - Parameter space of the ACO-E algorithm

Parameters	**Interval**
R_s (pheromone range)	$[0, 8]$
$\Delta\varphi_0$	$[0, 4]$
ε	$Uniform\ [0, 1]$
a_1	$[0, 2]$
a_2	$[0, 2]$
ρ	$[0, 1]$
η	$[0, 2]$
μ	$[0, 2]$
λ	$[0, 2]$

Table 4.2 - Parameter space of the ACO-FTS-RR3-E algorithm

Parameters	Interval
R_T (perception range)	$[1, 19]$
R_s (pheromone range)	$[0, 8]$
$\Delta\varphi_0$	$[0, 4]$
ε	$Uniform\ [0, 1]$
a_1	$[0, 2]$
a_2	$[0, 2]$
ρ	$[0, 1]$
η	$[0, 2]$
μ	$[0, 2]$
λ	$[0, 2]$
β_0	$[0, 1]$
γ	$1/L\ (L = max\{m, n\})$
α	$[0, 0.4]$
σ	$Uniform[0, 1]$

Table 4.3 - Parameter space of the ACO-PSO-RR3-E algorithm

Parameters	Interval
R_T (perception range)	$[1, 19]$
R_s (pheromone range)	$[0, 8]$
$\Delta\varphi_0$	$[0, 4]$
ε_{ACO}	$Uniform\ [0, 1]$
a_1	$[0, 2]$
a_2	$[0, 2]$
ρ	$[0, 1]$
η	$[0, 2]$
μ	$[0, 2]$
λ	$[0, 2]$
θ	$[0.4, 1]$
κ	$[0, 4]$
ε_{PSO}	$Uniform\ [0, 1]$

Table 4.4 - Parameter space of the ACO-ABC-RR3-E algorithm

Parameters	Interval
R_T (perception range)	$[1, 19]$
R_S (pheromone range)	$[0, 8]$
$\Delta\varphi_0$	$[0, 4]$
ε	$Uniform\ [0, 1]$
a_1	$[0, 2]$
a_2	$[0, 2]$
ρ	$[0, 1]$
η	$[0, 2]$
μ	$[0, 2]$
λ	$[0, 2]$
ϕ	$Uniform\ [-1, 1]$

During the recruitment task, if a robot is subject to many recruitment requests, it moves to the closest target in case that the coordination algorithm is FTS or PSO. Instead, in case of ABC algorithm, the k-th robot selects the z-th target with a probability inspired by Eq. (2.18), as in the following:

$$p_{kz} = \frac{1/d_{kz}}{\sum_{r=1}^{FR_k} 1/d_{kr}} \qquad (4.1)$$

where FR_k are the set of help requests (i.e., found targets) received by the k-th robot, $FR_k \subset F \subset T$, and d_{kz} is the Euclidean distance. Moreover, the ABC algorithm takes into account only the Eq. (2.19) to calculate the new position of a robot recruited by a coordinator.

Table 4.5 show the parametric space of SFE algorithm, used for both exploration and recruitment.

Table 4.5 - Parameter space of the SFE and SFE-RR3 algorithms

Parameters	**Interval**
stigmergy.radiusTop	$[1, 13]$
stigmergy.radiusDown	$[13, 19]$
stigmergy.evapRate	$[0.01, 1]$
stigmergy.olfactoryHabituation	$[1, 10]$
stigmergy.repulsiveRadius	$[0, 8]$
stigmergy.repulsiveEvapRate	$[0.01, 0.5]$
flocking.angle	$[15, 45]$
flocking.wiggleVar	$[5, 15]$
flocking.radiusSeparate	$[6, 16]$
flocking.maxSeparateTurn	$[30, 45]$
flocking.radiusAlign	$[16, 22]$
flocking.maxAlignTurn	$[30, 45]$
flocking.radiusCohere	$[18, 26]$
flocking.maxCohereTurn	$[15, 30]$

It is worth noting that when the parameters *radiusTop*, *radiusDown*, and *repulsiveRadius* are very small, then attractive, and repulsive stigmergy are very low too. Similarly, if the flocking *angle* is very small, then flocking is very low since no flock mate is visible. Thus, such parameters can lower/raise the contribution of each component in the overall workflow, in a continuous optimization space.

The most sensitive hyper-parameters of Differential Evolution are the differential weight (*F*), the crossover rate (*CR*), and the population size (*NP*). We use a multiplier of the problem dimension for setting the total population size: the population has $4D$ individuals. Based on the literature, as discussed in the section 2.1.6.1, the range of values to consider are

CR in $[0.1, 0.9]$ with steps of 0.1, F in $[0.4, 0.9]$ with steps of 0.1. Two mutation strategies have been experimented.

Figure 4.2 and Figure 4.3 show the grid search on the *Illegal Dump* scenario, with the ACO-RR1-E algorithm, for the DE/rand/1/bin ("r" for short) and the DE/best/1/bin ("b" for short), respectively. We use the grid search because it is the traditional way of performing hyperparameter optimization and the number of combinations of hyperparameters is not very high.

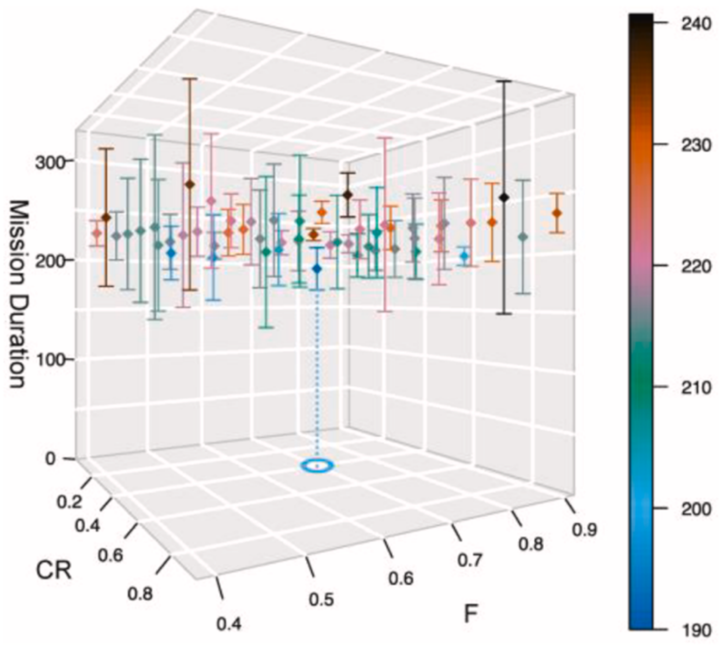


Figure 4.2 - DE/best/1/bin hyperparameters grid search, with the ACO-E algorithm and the Illegal Dump scenario

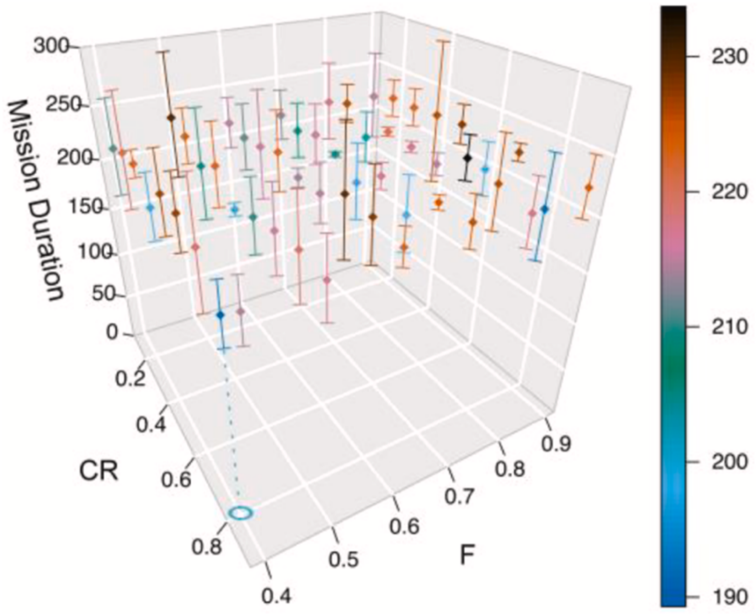


Figure 4.3 - DE/rand/1/bin hyperparameters grid search, with the ACO-E algorithm and the Illegal Dump scenario

In these cases, the minimum duration of 189.3 ± 26.55 (r) and 190.0 ± 20.46 (b) is achieved for (CR, F) equals to $(0.8, 0.4)$ (r) and $(0.7, 0.4)$ (b), respectively. In both figures, the optimal values are highlighted with a small circle in the (CR, F) plane.

Similarly, Figure 4.4 and Figure 4.5 show the grid search process with the SFE-RR1 algorithm, for DE/rand/1/bin (r) and DE/best/1/bin (b), respectively.

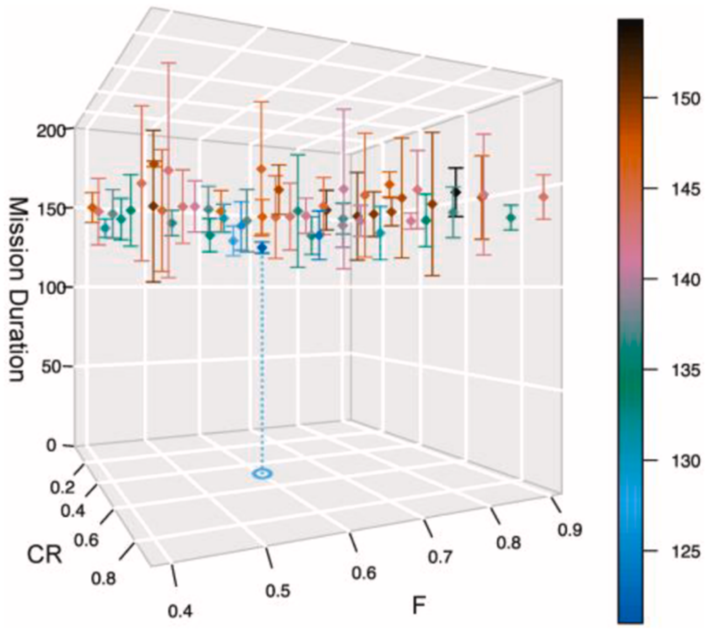


Figure 4.4 - DE/best/1/bin hyperparameters grid search, with the SFE algorithm and the Illegal Dump scenario

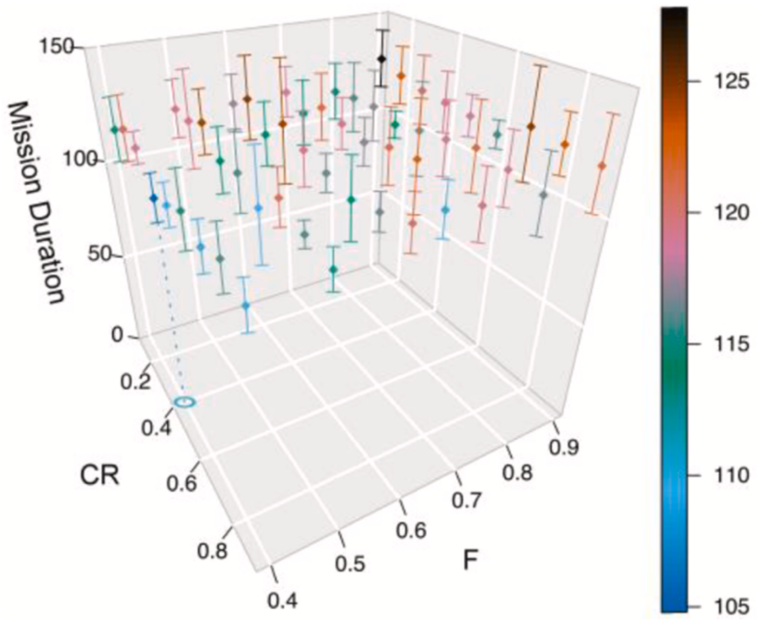


Figure 4.5 - DE/rand/1/bin hyperparameters grid search, with the SFE algorithm and the Illegal Dump scenario

Here, the minimum duration of 104.8 ± 10.45 (r) and 121.0 ± 2.99 (b) is achieved for (CR, F) equals to $(0.4, 0.4)$ (r) and $(0.6, 0.5)$ (b), respectively. As a result, the DE/rand/1/bin strategy achieves better performance than DE/best/1/bin with the SFE-RR1 algorithm and achieves performance similar to DE/best/1/bin with the ACO-RR1-E algorithm. Overall, the effectiveness of DE/rand/1/bin can be considered better.

4.1.3 Management of the stochastic behavior

An important aspect to consider is the control of the uncertainty potentially resulting from the initial swarm position and from the random-evaluated parameters. For this purpose, the initial swarm position is fixed: the swarms are initially located at the corners of the environment and oriented towards the center of it. However, there are two sources of non-determinism that can further influence the algorithmic performance.

The first source occurs at the application level of the target search, because all swarm algorithms inherently include random-valued parameters: $wiggle$ (SFE), ε_{ACO} (ACO), σ (FTS), ε_{PSO} (PSO), and ϕ (ABC). To manage this uncertainty, we adopt confidence intervals as a way to measure performance beyond statistical fluctuations. Furthermore, in contrast to the other swarm algorithms, the SFE allows to adapt the range of the *wiggle* via the DE optimization, for achieving the best cost-uncertainty ratio.

The second source of non-determinism occurs at the optimization level provided by the DE. Specifically, in the **Algorithm 3.1** the *intializePopulation* function is managed via the lower/upper bounds per parameter, and by a Latin Hypercube sampling to maximize the coverage of the available parameter space. The *generateMutant* also involves multiple random extractions, except for the DE/best/1/bin, to select the best individual as a base vector $p_i^{(t)}$. Finally, the *binomialCrossover* includes some random extractions, managed by the parameter *CR*. To control the last two variabilities, two mutation strategies and various *CR* values have been compared in the hyperparameters search process. Finally, to further reduce the overall uncertainty, each fitness evaluation is measured as an average of 10 trials, and the best result provided by the DE is calculated as an average of 3 independent trials made by 40 generations.

4.1.4 Experimental results

The study is based on the three scenarios presented in the section 3.2. By using the DE/rand/1/bin and the optimal values of hyperparameters determined by the grid search, a comparative analysis of the different algorithms has been carried out. For each scenario and for each strategy, the DE optimization has been carried out 10 times, determining via a graphical normality test that the resulting mission duration is well modelled by a normal distribution. Finally, the 95% confidence intervals have been calculated. Table 4.6 and Table 4.7 show the mission duration before and after the DE.

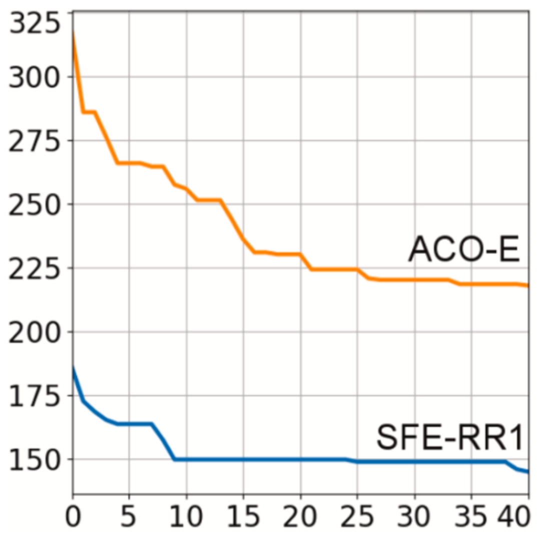
Table 4.6 - Swarm Exploration: mission duration before and after Differential Evolution

Scenario	**Algorithm**	**Mission duration before DE**	**Mission duration after DE**
Dump	SFE-RR1	185.97 ± 13.50	144.87 ± 09.62
"	ACO-RR1-E	317.10 ± 18.10	217.87 ± 09.56
Rural Mine	SFE-RR1	226.67 ± 51.03	159.53 ± 20.37
"	ACO-RR1-E	256.80 ± 14.08	205.00 ± 07.61
LPG Leak	SFE-RR1	191.57 ± 17.13	134.87 ± 05.09
"	ACO-RR1-E	215.43 ± 25.35	168.80 ± 04.04

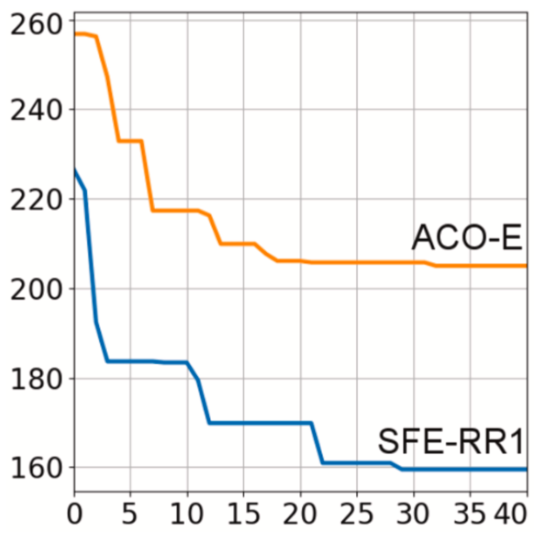
Table 4.7 - Swarm Exploration + Recruitment: mission duration, before and after Differential Evolution

Scenario	**Algorithm**	**Mission duration before DE**	**Mission duration after DE**
Dump	SFE-RR3	251.87 ± 27.31	186.20 ± 04.02
"	ACO-FTS-RR3-E	331.23 ± 20.68	261.47 ± 09.10
"	ACO-PSO-RR3-E	396.00 ± 09.99	269.23 ± 03.55
"	ACO-ABC-RR3-E	575.87 ± 131.24	409.33 ± 19.93
Rural Mine	SFE-RR3	267.70 ± 24.51	193.90 ± 24.71
"	ACO-FTS-RR3-E	338.00 ± 61.86	236.67 ± 01.73
"	ACO-PSO-RR3-E	316.70 ± 30.45	262.00 ± 03.85
"	ACO-ABC-RR3-E	409.10 ± 26.56	318.00 ± 22.83
LPG Leak	SFE-RR3	220.10 ± 05.05	168.77 ± 07.44
"	ACO-FTS-RR3-E	459.77 ± 09.10	286.20 ± 21.12
"	ACO-PSO-RR3-E	482.43 ± 28.61	302.23 ± 14.45
"	ACO-ABC-RR3-E	832.43 ± 15.15	577.13 ± 19.20

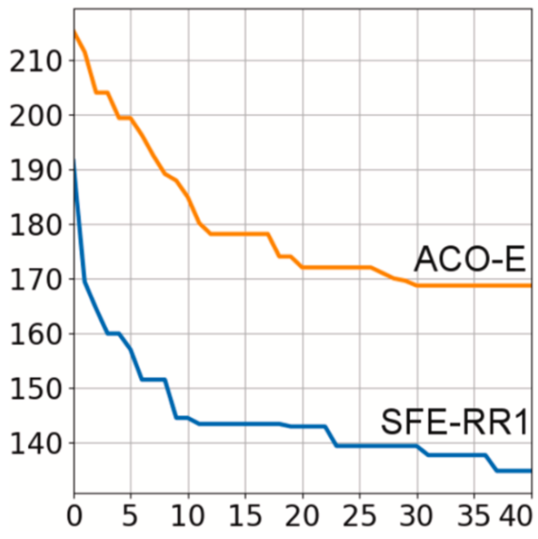
In both tables, it is apparent that the swarm exploration and recruitment carried out by the proposed SFE algorithm outperform the other strategies. Furthermore, it is clear that the DE optimization sensibly improves all the algorithms by providing adaptation to the specific scenario. Figure 4.6 shows the average best fitness against number of generations of the optimization process.



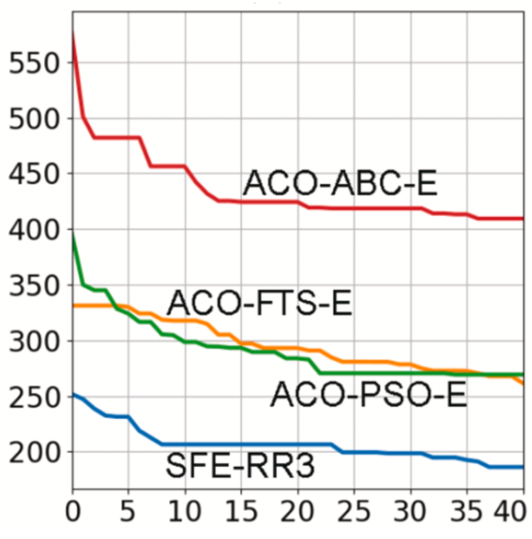
(a)



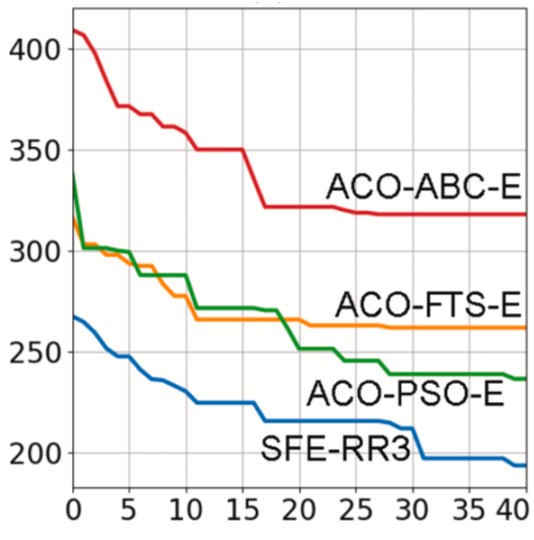
(b)



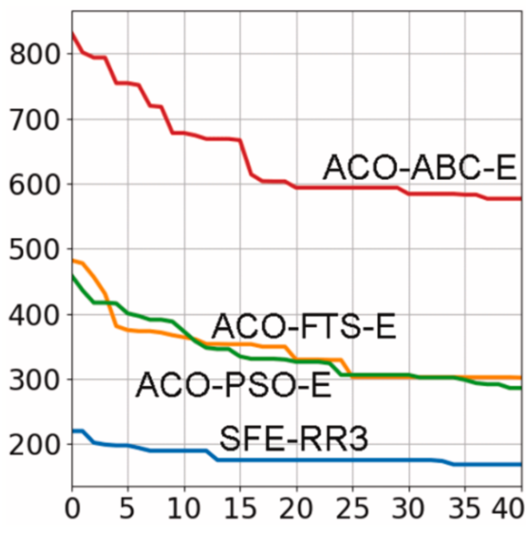
(c)



(d)



(e)



(f)

Figure 4.6 - Mission duration optimization: average best fitness against number of generations. Exploration on (a) Dump, (b) Rural Mine, (c) LPG Leak, Exploration + Recruitment on (d) Dump, (e) Rural Mine, (f) LPG Leak

Moreover, to better show the improvements made by the optimization of the target discovery process, Figure 4.7 shows the average percentage of target found against time by the SFE on Illegal Dump scenario, before and after the DE, over 10 trials.

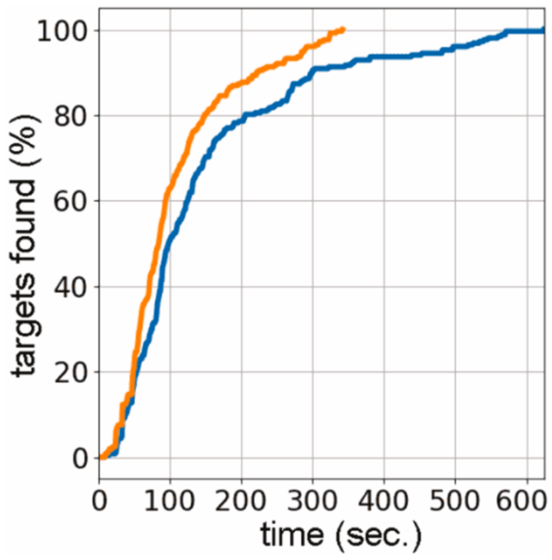


Figure 4.7 - Illegal Dump scenario: average percentage of targets found against time achieved by SFE algorithm, before (blue) and after (orange) DE

Finally, to show the computational efficiency, we consider the duration of DE for the different algorithms and for each scenario. The runtime of the DE depends linearly on the population size and on the number of generations. We fix the generations to 40 for all the algorithms. We have also to consider that implementation is engineered for parallel computing. The hardware and software platforms used are CPU Intel® Xeon® Gold 6140M at 2.2-2,3 GHz, Linux OS and Python for optimization process, and Java/NetLogo for coding algorithms and mission simulation. The optimization time of a mission depends on the scenario complexity and on the quality of the coordination mechanism, which are difficult to express. The optimization time can be empirically measured via the average DE runtime per scenario. Table 4.8 shows the average DE optimization time, over 3 runs, for 40 generations.

The computational model of the SFE is the most efficient for both exploration and recruitment. In contrast, when considering the complexity in memory, a different situation appears. Table 4.9 shows the memory usage for each algorithm, for the Illegal Dump scenario. It is apparent from Table 4.9 that the SFE is much more expensive in terms of memory.

Table 4.8 - Average DE opimization duration for 40 generations

Scenario	**Algorithm**	**Avg DE time**	**Population size**
Dump	SFE-RR1	1h 04' 01"	56
"	ACO-RR1-E	9h 49' 53"	32
"	SFE-RR3	1h 56' 26"	56
"	ACO-FTS-RR3-E	15h 29' 39"	44
"	ACO-PSO-RR3-E	1d 7h 29' 41"	44
"	ACO-ABC-RR3-E	19h 41' 13"	36
Rural Mine	SFE-RR1	1h 26' 29"	56
"	ACO-RR1-E	19h 34' 52"	32
"	SFE-RR3	2h 07' 22"	56
"	ACO-FTS-RR3-E	14h 26' 24"	44
"	ACO-PSO-RR3-E	1d 19h 39' 24"	44
"	ACO-ABC-RR3-E	14h 42' 13"	36
LPG Leak	SFE-RR1	1h 04' 16"	56
"	ACO-RR1-E	1d 1h 49' 55"	32
"	SFE-RR3	1h 59' 12"	56
"	ACO-FTS-RR3-E	22h 39' 00"	44
"	ACO-PSO-RR3-E	15h 16' 18"	44
"	ACO-ABC-RR3-E	1d 3h 13' 45"	36

Table 4.9 - Memory usage at the end of the 1st DE generation

Algorithm	**RAM (GB)**	**Population size**	**RAM (GB) per individual**
SFE-RR1	317	56	5.66
ACO-RR1-E	127	32	3.97
SFE-RR3	321	56	5.73
ACO-FTS-RR3-E	201	44	4.57
ACO-PSO-RR3-E	175	44	3.98
ACO-ABC-RR3-E	165	36	4.58

4.2 Targets tracking via UAV swarms

In the simulation testbed a target represents the basic element to model different types of objects, substances or chemical agents that should be detected, within the battery life, by the specific sensors that the drones are supposed to be equipped with. The scenarios described so far are characterized by the presence of static targets, that is, where the number and position of individual targets remain unchanged throughout the simulation time. However, this class of problems does not cover the range of possible real-world applications that find a natural solution in the use of Unmanned Aerial Vehicles (UAVs or drones) swarms. Specifically, certain dynamic phenomena such as the evolution of a fire or the expansion of a toxic cloud into the atmosphere need to be analyzed as a whole, possibly in a three-dimensional way, and not from a single perspective. As a consequence, in scenarios of this type, the need to use a swarm of drones has an undiscussed value especially of a technical nature, otherwise not achievable by other means. In fact, the observation of a phenomenon that can be characterized only if contextually and in parallel we have data from all observable points of view, is something that probably can be done only with swarms of drones and not with single drones.

The possibility of being able to evaluate the coordination performance of drones also in the characterization of dynamic phenomena requires the need to extend the simulation logic. In this case, essentially, the number and position of targets may change during the simulation. In the case of scenarios with static targets, the performance of the coordination algorithm is measured by considering the time it takes to detect most (typically 95%) of the targets to be found (best effort). However, considering a scenario with moving targets, this metric is unsuitable mainly because, being the characterization of a dynamic phenomenon, its speed of evolution could prevent drones from ensuring such a high detection rate (real time). So, it is needed to introduce a different suitable metric. The use of drone swarms within scenarios characterized by dynamic phenomena is particularly suitable for two types of missions:

- *discovery* of the dynamic phenomenon: the swarm of drones has the task of patrolling the search environment and promptly detect a possible anomaly in an early stage of its evolution.

- *tracking* of the dynamic phenomenon: the swarm of drones has the task of characterizing the phenomenon during its partial or complete evolution.

The nature of these tasks is deeply different from the detection of a static object. In fact, the dynamic phenomenon to be detected or tracked is likely to occur when the swarm of drones is already deployed in flock to patrol an area. Unlike the case of scenarios with static targets, dynamic scenarios must include the possibility of deploying drones on the environment before the phenomenon begins to evolve. Therefore, the simulation testbed has been suitably adapted to this purpose. In order to make a dynamic phenomenon suitable for simulation, the evolution of the targets can be realized by means of a succession of frames obtained through the sampling of the real phenomenon at certain time instants. By adopting this solution, in addition to the simulated time, even the dynamics of the targets is discretized. This simplification does not affect the correctness of the modeling in the hypothesis in which the cruise speed at which the drones fly, and their analysis rate are an order of magnitude higher than the sampling rate of the dynamic phenomenon to be observed. For example, considering an adequate drone model that flies at a speed of 50 Km/h and knowing that the examined phenomenon is evolving, but has a speed of 5 Km/h, for a mission of about 20 minutes we can imagine to consider only 4 or 5 frames of the evolution of the phenomenon, in which the phenomenon itself can be considered almost stationary.

In the simulation testbed, the target dynamics is supplied as a sequence of frames whose transition is ruled by a preset time frequency. This both avoid the effort of coding the equations underlying the dynamics of targets and allows to use real available frames to recreate a new scenario. We consider three dynamic scenarios:

- *Fire Tracking* comes by a propagation model developed by the Northwestern University (Wilensky, 1997).
- *H_2S Leak* is based on a sour gas accident occurred in December 2003, in Chongqing City, in a Gas Field located in the northeastern of Sichuan, China (Qingchun & Laibin, 2011).
- *LPG Leak* is based on an accident occurred in June 2009 in Viareggio, Italy, and involving an LPG railcar rupture in a congested urban area (Pontiggia, et al., 2011).

Table 4.10 summarizes the main features of each scenario.

Table 4.10 - Characteristics of dynamic scenarios

Scenario	**Area size (m × m)**	**Targets animation**	**N. of frames**
Fire Tracking	1400 × 1400	20 min.	5
H_2S Leak	4816 × 4400	48 min.	4
LPG Leak	500 × 300	4 min.	4

Formally, given a simulated scenario Ω, made of:

i. simulation instants of time $t \in \mathbb{N}^+$;

ii. a set of drones $\{D\}$, each drone having a dynamic position $(x_t, y_t)_D$;

iii. a set of targets $\tau \in T$, that can change every frame transition period P, i.e., $(x, y)_{\tau(\varphi)}$, $\varphi = 0, P, 2P, \dots, tP, \dots, \phi$, where $\phi = nP$ is the predefined final instant of the simulation, and n is the number of frames of the simulation.

The fitness of the dynamic simulated scenario Ω is then defined as the average percentage of targets discovered in all frames, as in the following:

$$fitness(\Omega) = \frac{P}{\phi} \sum_{\varphi=P}^{\phi} \frac{|T_F(\varphi)|}{|T(\varphi)|} \qquad (4.2)$$

where $T(\varphi)$ is the number of targets in the frame that ends at time φ, and $T_F(\varphi)$ is the corresponding number of found targets.

As a pilot example, Figure 4.8 and Figure 4.9 show two frames of the Fire Tracking scenario. Here, drones are represented as lilac arrowheads. The fire front to be tracked is represented by colored targeted cells. Thus, a single-colored targeted cell represents a small portion of the fire front. A targeted cell can be discovered/tracked, i.e., the yellow cell, or undiscovered/untracked, i.e., the red cell. The pheromone clouds are depicted in the figures as clusters of gray cells, where the level of gray represents the intensity of the pheromone. These clearly show that the swarm is tracking the fire evolutions.

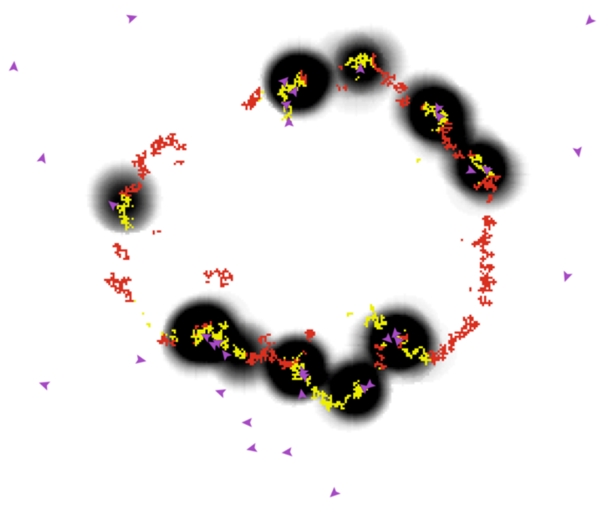


Figure 4.8 - Fire Tracking: simulation frame at tick 1013

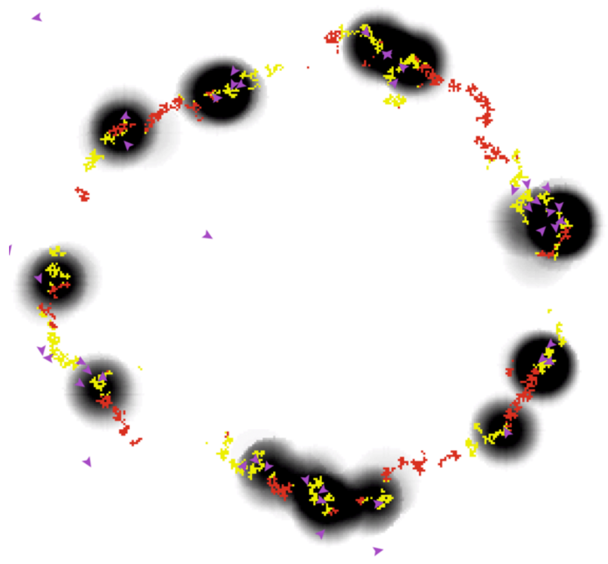


Figure 4.9 - Fire Tracking: simulation frame at tick 1277

Table 4.11 shows the performance of 80 UAVs swarm, adapted for each scenario, in terms of the 95% confidence interval over 10 repeated trials.

Table 4.11 - 80 UAVs swarm performance over dynamic scenarios

Scenario	Performance
Fire Tracking	99.88 ± 0.06 %
H_2S Leak	98.78 ± 0.17 %
LPG Leak	93.88 ± 0.28 %

Experimental results show the effectiveness of the swarm in tracking the dynamic phenomenon. The number of UAVs has been determined by setting incremental values and assessing the impact on performance. For example, Table 4.12 shows the performance of 20, 40, 60, 80 UAVs for Fire Tracking, in terms of 95% confidence interval over 10 repeated trials.

Table 4.12 - Fire Tracking: swarm performance for a different number of UAVs

N. of UAVs	Performance
20	60.64 ± 2.06 %
40	90.36 ± 0.54 %
60	98.43 ± 0.25 %
80	99.88 ± 0.06 %

4.3 Oceans cleanup management via USV swarms

Plastic pollution is a major source of marine debris. Many plastics, including polypropylene, polyethylene, nylon, polystyrene, polycarbonate, and polyvinyl chloride (PVC) are very durable; some are predicted to persist in the marine environment for many years. The wind and ocean current can lead to the accumulation over time of buoyant plastic in specific geographical areas so inducing serious pollution problems, also related to the degradation of plastic materials and the formation of sea-slicks and biofilms. Europe, after China, is the second largest producer of plastic. The major plastic-consuming countries in the European Union are Germany and Italy (Villarrubia-Gómez, Cornell, & Fabres, 2018). Observation and mitigation represent a fundamental step to marine plastics reduction. Among the most common mitigation techniques we mention those based on a removing, cleaning-up and biotechnology strategies.

This application focuses on the perspective use of the Albatross Unmanned Surface Vehicle (USV) prototype which was designed and presented in 2019 at the NASA Space Apps Challenge. In the literature, swarms of robots are increasingly proposed as a viable solution to mitigate the problem of plastic pollution in oceans. The cooperation of a USV swarm can sensibly increase the performances of cleaning dirty oceanic zones. The USV is assumed to be equipped with on-board sensors that allow it to identify the plastic debris (Kylili, Kyriakides, Artusi, & Hadjistassou, 2019).

In general, the cooperation of USVs can be coordinated either in a centralized or a decentralized way. The centralized coordination asks for a human operator who analyses and collects information about dirty zones and updates the environment map of USVs. As a result, the swarm navigates to a new assigned dirty zone and cleans it. The main characteristic of a USV coordination strategy is its capability to be autonomous, robust, resilient, and adaptive. Centralized logic solutions are not effective for this purpose, due to the high level of complexity, design, and management effort. In contrast, decentralized logic approaches can provide a USV swarm with a certain degree of autonomy (Meng, et al., 2014).

In this application, two swarm intelligence algorithms are compared, i.e., Ant Colony Optimization (ACO) (Palmieri, Yang, De Rango, & Marano, 2017) with Evolution (ACO-E), and Stigmergy Flocking Evolution (SFE). As highlighted in the previous chapter, SFE includes different biological cooperation models, inspired by chemical pheromone, olfactory, and visual perception. Both algorithms are parametrically adaptive with respect to the layout, thanks to the use of Evolutionary Optimization. Simulation results show that the SFE algorithm sensibly overcomes the ACO in terms of amount of collected debris per month.

A key point of the USV swarm coordination is the capability to provide a dynamic update of the environment map, according to the sea current that moves the plastic debris. To this purpose, here the model of the Copernicus Marine Service is used (Liubartseva, Coppini, Lecci, & Clementi, 2018). The model provides a stream of frames with the spatial distribution of floating plastics, based on the pattern of ocean currents. The model has been created from Earth Observation data in a Numerical Weather Prediction (NWP).

The exploration problem is modelled by discretizing the environment into a lattice of cells. Each cell has an area of 0.25 Km^2. The temporal unit (tick) of the simulation environment is set to 5 minutes. The duration of the mission is statically specified and corresponds to one month of floating plastic movement. The target dynamics is reproduced by using a sequence of frames with daily transition. The USV position and direction is dynamic and set according to exploration and coordination rules, which can be parametrically adapted by Differential Evolution algorithm.

Figure 4.10 summarizes the main steps of the procedure used to model the daily spatial distribution of plastics within the study area.

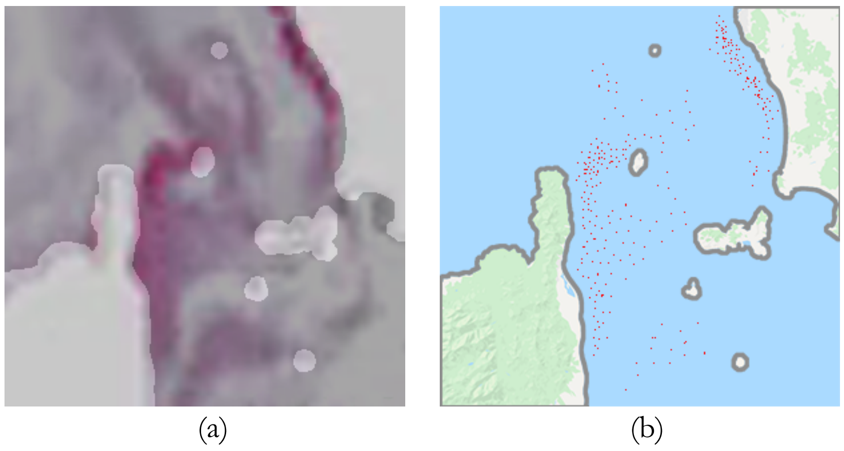

(a) (b)

Figure 4.10 - Procedure to determine the daily distribution of plastics

The starting point of the procedure is given by a frame providing spatial density of plastics over sea. The Figure 4.10(a) represents the spatial density of plastic provided by the Copernicus Marine Service. This frame is used to estimate the 2D probability density function to find plastics at a location (latitude, longitude) over the sea. A Montecarlo technique is then used to sample the location of plastics over sea, to generate the vectorial map with the target to collect, represented as red points in Figure 4.10(b). The overall collected plastic by the swarm is returned by the simulated mission and is used as a fitness value to measure the effectiveness of each algorithm.

The study area covers the portion of the Tyrrhenian Sea between northeastern Corsica and Tuscany. Specifically, it is a 150.5 × 150.5 Km^2 area, with an overall navigable surface of 16235 Km^2. This area is often affected by the formation of non-permanent floating plastic islands, due to the characteristic sea currents (Fossi, et al., 2017). A realistic scenario is simulated by using a video animation of the sea plastic pollution made available by the Copernicus Marine Service (Liubartseva, Coppini, Lecci, & Clementi, 2018). For this study, the period from 01/07/2016 till 30/07/2016 has been selected. Figure 4.11 shows the pheromone clouds and the USV swarm tracking the floating plastic movements.

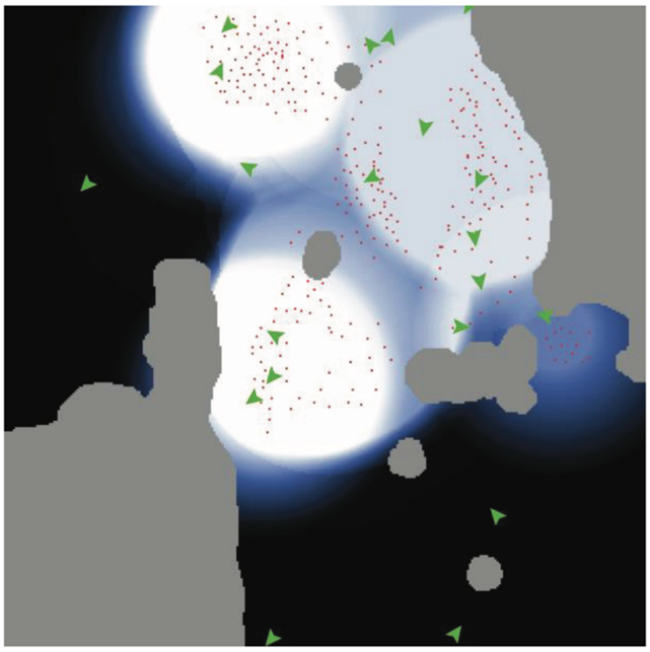


Figure 4.11 - Simulation of plastic collection

In our study, we have set the simulator with the physical and technological parameters of the USV prototype designed in the ALBATROSS project (ALBATROSS, Trash Cleanup, 2019). The main characteristics of this drone are summarized in Table 4.13.

Table 4.13 - Techincal specification of the ALBATROSS trimaran

USV Parameter	Real value
cruising speed	6 Km/h
maximum payload	6000 Kg
net capacity	33.3 Kg
size	25 × 13 m

The performances of the swarm coordination algorithm have been assessed by considering both ACO and SFE algorithms. Figure 4.12 shows the performance of 20 USVs swarm, obtained with the two coordination strategies in the same simulation configurations. For each strategy, the DE optimization is carried out 5 times, to calculate the 95% confidence intervals.

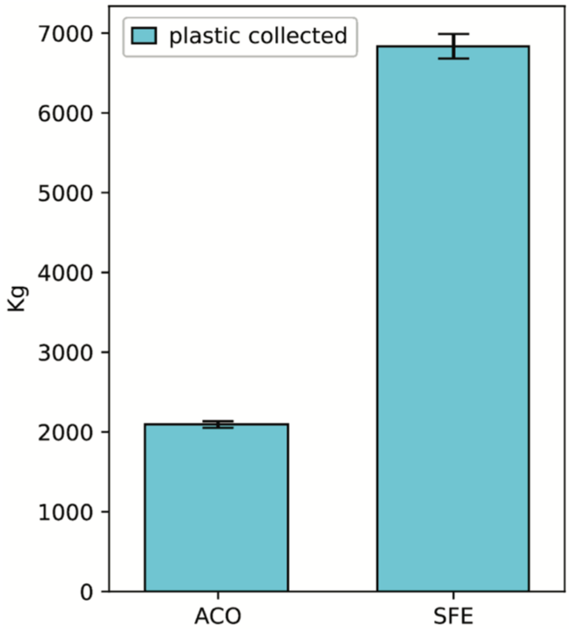


Figure 4.12 - Amount of plastic collected by the USV swarm

The results show that the SFE algorithm clearly outperforms the ACO strategy.

Chapter 5

5 Conclusion

This chapter draws the conclusion of this Ph.D. thesis. First, we discuss the proposed approach to design coordination algorithms for swarms of robots in target search. Then, a final section is devoted to possible future directions of this research.

5.1 Discussion

In this thesis, we present a novel methodology for designing heuristics for decentralized coordination of robot swarms. The approach is based on the use of a hyper-heuristic that aggregates and tunes the modular components of heuristics of lower level. The experimental results obtained from the simulations over heterogeneous scenarios are very promising. This confirms the effectiveness of hyper-heuristics in providing more generalized solutions to optimization problem, by working well over a set of problems, rather than producing good results for just a few problem instances.

Swarm robotics is the discipline that studies how to manage and coordinate large groups (swarms) of mostly simple autonomous robots. In practice, modeling robot behavior gets inspiration from swarm intelligence, where the desired collective behavior emerges from simple rules and local interactions. This coordination approach has been shown to have many advantages compared with other multi-robot systems. One of the problems in which swarm robotics is most used is target search in unstructured environments. Target search aims to discover elements of various complexity in a physical environment, by minimizing the overall discovery time. A target search mission is usually organized into environmental exploration, i.e., to search targets, and targets resolution, i.e., to collect sufficient target information.

In complex and open environments, the robots cannot exploit static information on layout and targets locations, therefore swarm robotics is well suited for an efficient targets discovery. In the literature, different heuristics inspired by biological systems have been proposed to guide robots to complete missions. We have considered and made adaptive some of these heuristics: Ant Colony Optimization (ACO), inspired by ants, for exploration tasks, whereas Firefly algorithm (FTS, Firefly Team Strategy), inspired by fireflies, Particle Swarm Optimization (PSO), inspired by birds flocking, and Artificial Bee Colony (ABC), inspired by honeybees, for recruitment tasks. We have adapted algorithm parameters via Differential Evolution optimization. Differential evolution has proven to be very effective in finding the best algorithmic parameters of a bio-inspired heuristic in order to improve the mission performance. In practice, it is not easy to select the most suitable heuristic for a specific mission. Moreover, although adaptive, the logic of bio-heuristics is constrained by models of biological species and requires relevant parametrization costs related to every new type of mission and to new instances of known missions.

In order to overcome the design constraints of bio-inspired approaches, we propose a novel methodology based on hyper-heuristics. Basically, a hyper-heuristic is a search method or a learning mechanism to select or generate heuristics that solve search problems. We have considered two fundamental behavioral components: stigmergy and flocking. Stigmergy is used to release an attractive, or repulsive, pheromone while detecting the presence, or absence, of a target during exploration. Flocking uses simple rules to model a robust and flexible swarm formation. We have parametrized the pheromone model and the flocking rules to obtain modular components. Then, in our design approach, for a given application domain the Differential Evolution optimizes the aggregation and tuning of the basic behavioral components in a unique and continuous search space. The proposed hyper-heuristic is called SFE because it is based on Stigmergy, Flocking, and Evolution.

Experimental results on real-world scenarios, carried out and released as a public simulation testbed, have shown that searching in the heuristic space allows to obtain more efficient coordination logics, in both exploration and recruitment. Indeed, the SFE significantly outperforms the other adaptive bio-heuristics in all considered scenarios. The SFE is also faster in terms of optimization duration, although it requires more memory. The technique has also proven effective when considering the problem of tracking dynamic targets. Experimental results obtained by using swarms of Unmanned Aerial Vehicles on real-world scenarios involving early fires and early toxic and dangerous gas dispersion are very promising. Moreover, we have considered another important and current topic, that is the problem of mitigation of plastic pollution in oceans. In this context, a realistic scenario has been simulated considering a dataset provided by the Copernicus Marine Service. We have configured the simulation testbed by using the technical specification of an Unmanned Surface Vehicle prototype designed in the ALBATROSS project. Comparative results with the ACO algorithm have clearly shown that the SFE algorithm is more suitable in coordinating swarms of USVs to collect plastic.

5.2 Future work

In the problem of coordinating swarms of robots searching for targets in unstructured environments, the approach based on hyper-heuristics has proved to be successful compared to the use of simple adaptive bio-heuristics. However, parametric optimization of modular heuristics considers the accuracy of the solution as the only goal, in terms of minimizing target search time in static scenarios or maximizing detected targets in dynamic scenarios. The limit of this approach lies in the difficulty to understand the solution with respect to the specific application context. In missions where it is useful to employ robot swarms, it would be desirable to consider both accuracy and transparency requirements in order to acquire a knowledge base that is accessible to human users. In this regard, one possible research direction could be to optimize the transparency of the system. In other words, a first goal could be to understand, through a proper design of the experimental plan, the relationship between some relevant features of the application scenarios (e.g., target density, target distribution, obstacle size, obstacle distribution, etc.) and the parameters related to stigmergy and flocking. Later, the rules extracted from the experimental results could be used to address the optimization of new types of missions or new instances of known missions. As a result, by intelligently and comprehensibly constraining the search space, it should be possible to reduce the variance of the solutions and obtain more statistically significant results.

Appendix A

Publications

International Journal Paper

- Mario G.C.A. Cimino, Domenico Minici, Manilo Monaco, Stefano Petrocchi, Gigliola Vaglini, "A hyper-heuristic methodology for coordinating swarms of robots in target search", Computers & Electrical Engineering, Volume 95, 2021, 107420.

Peer Reviewed International Conferences Papers

- Manilo Monaco, Mario G.C.A. Cimino, Gigliola Vaglini, Francesco Fusai, Giovanni Nico, "Managing the Oceans Cleanup via Sea Current Analysis and Bio-Inspired Coordination of USV Swarms", In: Proceedings of the *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*, pp. 8344-8347, Brussels, Belgium, 11-16 July 2021.

- Cimino Mario G.C.A., Lega Massimiliano, Monaco Manilo, Vaglini Gigliola, "Adaptive Exploration of a UAVs Swarm for Distributed Targets Detection and Tracking", In: Proceedings of *the 8th International Conference on Pattern Recognition Applications and Methods, (ICPRAM 2019)*, pp. 837-844, Prague, Czech Republic, 19-21 February 2019.

- Manilo Monaco, Giovanni Nico, Pier Francesco Biagi, Anita Ermini, Aleksandra Nina, Mario G.C.A. Cimino, Gigliola Vaglini, "Using VLF Time Series from the INFREP Network for the Study of Pre-Seismic Radio Anomalies", In: Proceedings of the *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*, pp. 8624-8627, Brussels, Belgium, 11-16 July 2021.

- Cimino Mario G.C.A., Dalla Bona Federico, Foglia Pierfrancesco, Monaco Manilo, Prete Cosimo A., Vaglini Gigliola, "Stock Price Forecasting Over Adaptive Timescale Using Supervised Learning and Receptive Fields", In: Groza A., Prasath R. (eds) *Mining Intelligence and Knowledge Exploration, (MIKE 2018)*, Cluj-Napoca, Romania, 20-22 December 2018, Lecture Notes in Computer Science, vol 11308. Springer, Cham.

Bibliography

Alfeo, A. L., Cimino, M. G., & Vaglini, G. (2019). Enhancing biologically inspired swarm behavior: Metaheuristics to foster the optimization of UAVs coordination in target search. *Computers & Operations Research, 110*, 34-47.

Alfeo, A. L., Cimino, M. G., De Francesco, N., Lazzeri, A., Lega, M., & Vaglini, G. (2018). Swarm coordination of mini-UAVs for target search using imperfect sensors. *Intelligent Decision Technologies, vol. 12*(no. 2), pp. 149-162.

Alfeo, A. L., Cimino, M. G., De Francesco, N., Lega, M., & Vaglini, G. (2018). Design and simulation of the emergent behavior of small drones swarming for distributed target localization. *Journal of Computational Science, 29*, 19-33.

Arvin, F., Murray, J. C., Shi, L., Zhang, C., & Yue, S. (2014). Development of an autonomous micro robot for swarm robotics. *IEEE International Conference on Mechatronics and Automation* (pp. 635-640). IEEE.

Beni, G., & Wang, J. (1989). Swarm intelligence in cellular robotic systems. *Robots and Biological Systems: Towards a new Bionics? Proceedings of the NATO Advanced Workshop on Robots and Biologica Systems* (pp. 703-712). Springer.

Bonabeau, E., Theraulaz, G., Deneubourg, J.-L., Aron, S., & Camazine, S. (1997). Self-organization in social insects. *Trends in Ecology & Evolution, vol. 12*(5), 188-193.

Branke, J., Nguyen, S., Pickardt, C. W., & Zhang, M. (2016). Automated Design of Production Scheduling Heuristics: A Review. *IEEE Transactions on Evolutionary Computation, 20*(1), 110-124.

Brest, J., Greiner, S., Boskovic, B., Mernik, M., & Zumer, V. (2006). Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. *IEEE Transactions on Evolutionary Computation, 10*(6), 646-657.

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., & Qu, R. (2013). Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society, 64*, 1695-1724.

Burke, E. K., Hyde, M., Kendall, G., & Woodward, J. (2010). A Genetic Programming Hyper-Heuristic Approach for Evolving 2-D Strip Packing Heuristics. *IEEE Transactions on Evolutionary Computation, 14*(6), 942-958.

Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., & Schulenburg, S. (2003). Hyper-Heuristics: An Emerging Direction in Modern Search Technology. *Glover F., Kochenberger G.A. (eds) Handbook of Metaheuristics. International Series in Operations Research & Management Science, 57*, 457-474.

Chatterjee, A., & Siarry, P. (2006). Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization. *Computers & Operations Research, 33*(3), 859-871.

Cimino, M. G., Lazzeri, A., & Vaglini, G. (2015). Improving the Analysis of Context-Aware Information via Marker-Based Stigmergy and Differential Evolution. *International Conference on Artificial Intelligence and Soft Computing. vol. 9120*, pp. 341-352. Springer.

Cimino, M. G., Lega, M., Monaco, M., & Vaglini, G. (2019). Adaptive exploration of a UAVs swarm for distributed targets detection and tracking. *The 8th International Conference on Pattern Recognition Applications and Methods (ICPRAM)* (pp. 837-844). Prague: SciTePress.

Contreras-Cruz, M. A., Ayala-Ramirez, V., & Hernandez-Belmonte, U. H. (2015). Mobile robot path planning using artificial bee colony and evolutionary programming. *Applied Soft Computing, 30*, 319-328.

Countryman, S. M., Stumpe, M. C., Crow, S. P., Adler, F. R., Greene, M. J., Vonshak, M., & Gordon, D. M. (2015). Collective search by ants in microgravity. *Frontiers in Ecology and Evolution, 3*, 25.

Dorigo, M., & Blum, C. (2005). Ant colony optimization theory: a survey. *Theoretical Computer Science, 344*(2-3), 243-278.

Dorigo, M., Birattari, M., & Stützle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine. vol.1, no. 4*, pp. 28-39. IEEE.

Dorigo, M., Bonabeau, E., & Theraulaz, G. (2000). Ant algorithms and stigmergy. *Future Generation Computer Systems, 16*(8), 851-871.

Drake, J. H., Hyde, M., Ibrahim, K., & Ozcan, E. (2014). A genetic programming hyper-heuristic for the multidimensional knapsack problem. *Kybernetes, 43*(9/10), 1500-1511.

Ducatelle, F., Di Caro, G. A., Pinciroli, C., & Gambardella, L. M. (2011). Self-organized cooperation between robotic swarms. *Swarm Intelligence, 5*(2), 73-96.

Eiben, A., Hinterding, R., & Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation, 3*(2), 124-141.

Felipe, Á., Ortuño, M. T., Righini, G., & Tirado, G. (2014). A heuristic approach for the green vehicle routing problem with multiple technologies and partial recharges. *Transportation Research Part E: Logistics and Transportation Review, 71*, 111-128.

Fossi, M. C., Romeo, T., Baini, M., Panti, C., Marsili, L., Campani, T., . . . Lapucci, C. (2017). Plastic Debris Occurrence, Convergence Areas and Fin Whales Feeding Ground in the Mediterranean Marine Protected Area Pelagos Sanctuary: A Modeling Approach. *Frontiers in Marine Science, 4*(167).

Fujisawa, R., Dobata, S., Kubota, D., Imamura, I., & Matsuno, F. (2008). Dependency by Concentration of Pheromone Trail for Multiple Robots. *International Conference on Ant Colony Optimization and Swarm Intelligence. vol. 5217*, pp. 283-290. Springer.

Fukunaga, A. S. (2008). Automated Discovery of Local Search Heuristics for Satisfiability Testing. *Evolutionary Computation, 16*(1), 31-61.

Garrido, P., & Riff, M. C. (2010). DVRP: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic. *Journal of Heuristics, 16*(6), 795-834.

Gasa, K., Vinci, A. E., Puccinelli, E., Modafferi, L. M., & Sophie. (2019). *ALBATROSS, Trash Cleanup*. Retrieved from NASA Space Apps Challenge at University of Pisa: https://2019.spaceappschallenge.org/challenges/earths-oceans/trash-cleanup/teams/albatross/project

Geiger, C. D., Uzsoy, R., & Aytug, H. (2006). Rapid Modeling and Discovery of Priority Dispatching Rules: An Autonomous Learning Approach. *Journal of Scheduling, 9*(1), 7-34.

Gunaratne, C., & Garibay, I. (2021). NL4Py: Agent-based modeling in Python with parallelizable NetLogo workspaces. *SoftwareX, 16*, 100801.

Howden, D. J. (2013). Fire Tracking with Collective Intelligence using Dynamic Priority Maps. *IEEE Congress on Evolutionary Computation* (pp. 2610-2617). Cancun, Mexico: IEEE.

Hutter, F., Hoos, H. H., & Stützle, T. (2007). Automatic Algorithm Configuration based on Local Search. *The 22nd AAAI Conference on Artificial Intelligence* (pp. 1152-1157). AAAI.

Kahar, M., & Kendall, G. (2010). The examination timetabling problem at Universiti Malaysia Pahang: Comparison of a constructive heuristic with an existing software solution. *European Journal of Operational Research, 207*(2), 557-565.

Karaboga, D., & Akay, B. (2009). A comparative study of Artificial Bee Colony algorithm. *Applied Mathematics and Computation, vol. 214*(1), 108-132.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *ICNN'95 - International Conference on Neural Networks. vol. 4*, pp. 1942-1948. IEEE.

Kuyucu, T., Tanev, I., & Shimohara, K. (2012). Evolutionary Optimization of Pheromone-Based Stigmergic Communication. *European Conference on the Applications of Evolutionary Computation, 7248*, 63-72.

Kylili, K., Kyriakides, I., Artusi, A., & Hadjistassou, C. (2019). Identifying floating plastic marine debris using a deep learning approach. *Environmental Science and Pollution Research, 26*, 17091–17099.

Lewis, S. M., & Cratsley, C. K. (2008). Flash Signal Evolution, Mate Choice, and Predation in Fireflies. *Annual Review of Entomology, 53*(2), 293-321.

Liubartseva, S., Coppini, G., Lecci, R., & Clementi, E. (2018). Tracking plastics in the Mediterranean: 2D Lagrangian model. *Marine Pollution Bulletin, 129*(1), 151-162.

Lu, S., Xin, B., Zhang, H., & Chen, J. (2020). Agent-based Self-organized Constructive Heuristics for Travelling Salesman Problem. *2020 59th IEEE Conference on Decision and Control (CDC)* (pp. 1164-1169). Jeju, Korea (South): IEEE.

Masár, M., & Zelenka, J. (2012). Modification of PSO algorithm for the purpose of space exploration. *IEEE The 16th International Conference on Intelligent Engineering Systems (INES)* (pp. 51-54). Lisbon, Portugal: IEEE.

Meng, W., He, Z., Su, R., Shehabinia, A. R., Lin, L., Teo, R., & Xie, L. (2014). Decentralized Control of Multi-UAVs for Target Search, Tasking and Tracking. *IFAC Proceedings Volumes, 47*(3), 10048-10053.

Mohan, B. C., & Baskaran, R. (2012). A survey: Ant Colony Optimization based recent research and implementation on several engineering domain. *Expert Systems with Applications, 39*(4), 4618-4627.

Monaco, M. (2021). *GitHub*. Retrieved from Stigmergy Flocking Evolution repository: https://github.com/mlpi-unipi/sfe

Nguyen, S., Zhang, M., & Johnston, M. (2011). A genetic programming based hyper-heuristic approach for combinatorial optimisation. *The 13th annual Conference on Genetic and Evolutionary Computation GECCO'11* (pp. 1299-1306). ACM Digital Library.

Ong, Y.-S., Lim, M.-H., Zhu, N., & Wong, K.-W. (2006). Classification of adaptive memetic algorithms: a comparative study. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 36*(1), 141-152.

Palmieri, N., Yang, X.-S., De Rango, F., & Marano, S. (2017). Comparison of bio-inspired algorithms applied to the coordination of mobile robots considering the energy consumption. *Neural Computing and Applications, 31*(1), 263-286.

Paquay, C., Limbourg, S., & Schyns, M. (2018). A tailored two-phase constructive heuristic for the three-dimensional Multiple Bin Size Bin Packing Problem with transportation constraints. *European Journal of Operational Research, 267*(1), 52-64.

Pillay, N., & Qu, R. (2018). *Hyper-Heuristics: Theory and Applications*. Springer.

Poli, R., & Koza, J. (2014). Genetic Programming. In *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques* (pp. 143-185). Springer.

Pontiggia, M., Landucci, G., Busini, V., Derudi, M., Alba, M., Scaioni, M., . . . Rota, R. (2011). CFD model simulation of LPG dispersion in urban areas. *Atmospheric Environment, 45*(24), 3913-3923.

Qingchun, M., & Laibin, Z. (2011). CFD simulation study on gas dispersion for risk assessment: A case study of sour gas well blowout. *Safety Science, 49*(8-9), 1289-1295.

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. *The 14th annual conference on Computer graphics and interactive techniques SIGGRAPH '87* (pp. 25-34). ACM Digital Library.

Sauter, J. A., Matthews, R., Parunak, H. V., & Brueckner, S. A. (2007). Effectiveness of digital pheromones controlling swarming vehicles in military scenarios. *Journal of Aerospace Computing, Information, and Communication, 4*(5), 753-769.

Senanayake, M., Senthooran, I., Barca, J. C., Chung, H., Kamruzzaman, J., & Murshed, M. (2016). Search and tracking algorithms for swarms of robots: A survey. *Robotics and Autonomous Systems, vol. 75*(Part B), 422-434.

Smith, J. E. (2008). Self-Adaptation in Evolutionary Algorithms for Combinatorial Optimisation. In *Adaptive and Multilevel Metaheuristics. Studies in Computational Intelligence.* (Vol. 136, pp. 31-57). Springer.

Storn, R., & Price, K. (1997). Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization, 11*, 341-359.

Tan, Y., & Zheng, Z.-y. (2013). Research Advance in Swarm Robotics. *Defence Technology, 9*(1), 18-39.

Turgut, A. E., Çelikkanat, H., Gökçe, F., & Şahin, E. (2008). Self-organized flocking in mobile robot swarms. *Swarm Intelligence, 2*(2-4), 97-120.

Villarrubia-Gómez, P., Cornell, S. E., & Fabres, J. (2018). Marine plastic pollution as a planetary boundary threat – The drifting piece in the sustainability puzzle. *Marine Policy, 96*, 213-220.

Virágh, C., Vásárhelyi, G., Tarcai, N., Szörényi, T., Somorjai, G., Nepusz, T., & Vicsek, T. (2014). Flocking algorithm for autonomous flying robots. *Bioinspiration & Biomimetics, 9*(2), 049501.

Wilensky, U. (1997). *NetLogo Fire Model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.* Retrieved from http://ccl.northwestern.edu/netlogo/models/Fire

Wilensky, U., & Rand, W. (2015). *An introduction to agent-based modeling: modeling natural, social, and engineered complex systems with NetLogo.* Cambridge: MIT Press.

Yang, X.-S. (2009). Firefly Algorithms for Multimodal Optimization. *The 5th International Symposium on Stochastic Algorithms: Foundations and Applications. vol. 5792*, pp. 169-178. Springer.

Yang, X.-S. (2016). *Nature-Inspired Computation in Engineering.* Springer International Publishing.

Yang, X.-S. (2020). *Nature-Inspired Computation and Swarm Intelligence: Algorithms, Theory and Applications.* London: Academic Press. An imprint of Elsevier.

Yang, X.-S. (2020). *Nature-Inspired Optimization Algorithms.* Academic Press, Elsevier.

Yang, X.-S., & Deb, S. (2014). Cuckoo search: recent advances and applications. *Neural Computing and Applications, vol. 24*, pp. 169-174.

Yasuda, T., Adachi, A., & Ohkura, K. (2014). Self-organized flocking of a mobile robot swarm by topological distance-based interactions. *2014 IEEE/SICE International Symposium on System Integration* (pp. 106-111). Tokyo: IEEE.

Yu, S., Song, A., & Aleti, A. (2019). A study on online hyper-heuristic learning for swarm robots. *2019 IEEE Congress on Evolutionary Computation (CEC)* (p. 2721-2728). IEEE.

Zaharie, D. (2009). Influence of crossover on the behavior of Differential Evolution Algorithms. *Applied Soft Computing, 9*(3), 1126-1138.