2-1-2017

Decentralized algorithm of dynamic task allocation for a swarm of homogeneous robots

Maha AlShawi

The American University in Cairo

School of Sciences and Engineering

Decentralized Algorithm of Dynamic Task Allocation for a Swarm of Homogeneous Robots

By

Maha Hasan AlShawi

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science in Robotics, Control and Smart Systems (RCSS)

Under supervision of:

Dr. Mohamed Shalan

Associate Professor, Computer Science and Engineering (CSCE)

July, 2017

DECLARATION

I declare that this thesis is my work and that all used references are cited and properly indicated.

Maha Hasan AlShawi

ACKNOWLEDGMENTS

First, I thank God for providing me with the power to be able to finish my thesis and manage to cross all the obstacles that I have been through during my study time.

Then, I would like to thank my supervisor Dr. Mohamed Shalan for his dedication to get many ideas implemented, through the digestion and redirection of the crazy ideas I came up with. I do appreciate a lot his support when I first proposed the hybrid DPSO/SA approach. I have my first experience in providing a complete research and publication under his supervision. Thanks for his continuous technical support and comments, during various design phases.

My deepest gratitude goes to Prof. Amal Esawi; the associate dean of graduate studies at AUC who made my master life easier by just being there. Due to her loyalty, sincerity, her generous support, encouragement and guidance, I have continued this work till the end. Thanks for her inspiration.

I would like to extend thanks to many colleagues who generously supported me a lot in the past few years. My appreciation goes to Rana, Heba, Amr Morsy, Abdullah, Mahmoud Elfar, Mohamed Shalaby, Amr Elhenawy, Ahmed Samir, Ahmed Zahran, and Louis. I do really wish them all the best in their career and life.

Also, I have to mention all those close friends, who have been always there when I needed them most, or those who interacted with me almost on daily basis, I mean: Nada, Wessam, Alaa, Omnia, and Sahar.

Eman Helal; my mother has been a steady, silent soldier in my background lousy life. Without her dedication and perfection, I would never be able to do those many things I have done during the past few years. I have also to show gratitude to my father, sister Nada and my brothers who always believe in me. My extended family, Fatma, Sara, Ahmed, Abdullah, Areej, Morooj, and Basma, I hope they excuse me for being away when they needed me most.

Second, beyond the call of duty, I have to appreciate my luck to gain two grants, research grant and conference grant throughout my study time, which are supported by Yousef Jameel Science and Technology Research Center.

Thanks to Dr. Florin and Dr. Yousra, for providing me with a greater insight on the big picture, results, and analysis. I have to acknowledge that I have learned from them both a lot in my proposal defense.

ABSTRACT

The current trends in the robotics field have led to the development of large-scale swarm robot systems, which are deployed for complex missions. The robots in these systems must communicate and interact with each other and with their environment for complex task processing.

A major problem for this trend is the poor task planning mechanism, which includes both task decomposition and task allocation. Task allocation means to distribute and schedule a set of tasks to be accomplished by a group of robots to minimize the cost while satisfying operational constraints. Task allocation mechanism must be run by each robot, which integrates the swarm whenever it senses a change in the environment to make sure the robot is assigned to the most appropriate task, if not, the robot should reassign itself to its nearest task.

The main contribution in this thesis is to maximize the overall efficiency of the system by minimizing the total time needed to accomplish the dynamic task allocation problem. The near-optimal allocation schemes are found using a novel hybrid decentralized algorithm for a dynamic task allocation in a swarm of homogeneous robots, where the number of the tasks is more than the robots present in the system. This hybrid approach is based on both the Simulated Annealing (SA) optimization technique combined with the Discrete Particle Swarm Optimization (DPSO) technique.

Also, another major contribution in this thesis is the formulation of the dynamic task allocation equations for the homogeneous swarm robotics using integer linear programming and the cost function and constraints are introduced for the given problem. Then, the DPSO and SA algorithms are developed to accomplish the task in a minimal time. Simulation is implemented using only two test cases via MATLAB. Simulation results show that PSO exhibits a smaller and more stable convergence characteristics and SA technique owns a better quality solution.

Then, after developing the hybrid algorithm, which combines SA with PSO, simulation instances are extended to include fifteen more test cases with different swarm dimensions to ensure the robustness and scalability of the proposed algorithm over the traditional PSO and SA optimization techniques.

Based on the simulation results, the hybrid DPSO/SA approach proves to have a higher efficiency in both small and large swarm sizes than the other traditional algorithms such as Particle Swarm Optimization technique and Simulated Annealing technique. The simulation results also demonstrate that the proposed approach can dislodge a state from a local minimum and guide it to the global minimum. Thus, the contributions of the proposed hybrid DPSO/SA algorithm involve possessing both the pros of high quality solution in SA and the fast convergence time capability in PSO.

Also, a parameters' selection process for the hybrid algorithm is proposed as a further contribution in an attempt to enhance the algorithm efficiency because the heuristic optimization techniques are very sensitive to any parameter changes.

In addition, Verification is performed to ensure the effectiveness of the proposed algorithm by comparing it with results of an exact solver in terms of computational time, number of iterations and quality of solution. The exact solver that is used in this research is the Hungarian algorithm.

This comparison shows that the proposed algorithm gives a superior performance in almost all swarm sizes with both stable and small execution time. However, it also shows that the proposed hybrid algorithm's cost values which is the distance travelled by the robots to perform the tasks are larger than the cost values of the Hungarian algorithm but the execution time of the hybrid algorithm is much better.

Finally, one last contribution in this thesis is that the proposed algorithm is implemented and extensively tested in a real experiment using a swarm of 4 robots. The robots that are used in the real experiment called Elisa-III robots.

TABLE OF CONTENTS

LIST Of FIGURES

LIST Of TABLES

ACRONYMS

TA	Task Allocation
DTA	Dynamic task allocation
MTDTA	Minimal Time Dynamic Task Allocation
PSO	Particle Swarm Optimization
DPSO	Discrete Particle Swarm Optimization
SA	Simulated Annealing
DPSO/SA	Hybrid Discrete Particle Swarm Optimization/Simulated Annealing
DPSO	Discrete Particle Swarm Optimization
TS	Taboo Search
GA	Genetic Algorithm
AC	Ant Colony
ABC	Artificial Bee Colony
UAV	Unmanned aerial vehicle
CRASAR	Center for Robot-Assisted Search and Rescue
ASyMTRe	Automated Synthesis of Multi-Robot Task Solutions through Software Reconfiguration
SR	Single-robot tasks
MR	Multi-robot tasks
NP	Non-deterministic polynomial-time
LED	Light Emitting Diode
IR	Infrared

DC	Direct current
RF	Radio Frequency
SPI	Serial Peripheral Interface
ISM	Industrial, Scientific and Medical band basis
NI	necessary information
MATLAB	MATrix LABoratory

NOMENCLATURE

n	Number of robots
m	Number of tasks
A	Allocation
T	Temperature
T0	Initial temperature
t_k	current temperature value
t_{k-1}	previous temperature value
Tend	End temperature value
w	Inertia weight
wmax	Max inertia weight
wmin	Min inertia weight
I	Current iteration
Imax	Max iteration
Imin	Min iteration
pbest	Personal best solution
Gbest	Global best solution
lbest	Local best solution
V	Velocity
X	Particle position
c1	Acceleration constant 1
c2	Acceleration constant 2
maxit	Maximum number of iterations
cnt	counter
rg	Refresh Gap

rand	Random function
μ	Mean
σ	Represents the standard deviation
C	Distance between robot and task
Q	Number of allocations
λ	Cooling rate parameter
ΔX_{max}	The difference between the fitness value of the worst solution and the initial solution
p1	Acceptance rate of the worst new solution
Cold	SA current state
Cnew	SA new state
E(s)	SA new fitness value
E(s')	SA previous fitness value
D	Difference between the SA's current fitness value and previous fitness value
E	Explore state
B	Back state
R	Retrieve state
Ni	Number of prey of i type
T	Total number of robots
λ	The rate by which objects were found in the environment by a single robot
K	The probability for a single robot to find the nest
ϕi	The incoming rate per second of prey of type i
εi	The probability constant over time for a prey of type i to disappear
μi	The inverse of the average time required to retrieve a prey of type i
ρ	The probability to give up an ongoing retrieval
β	The probability for a robot to return to nest

γ	The probability for a robot to leave the nest and look for prey
πi	The probability to take a prey of type i upon encounter

Chapter 1

Introduction

In this chapter, a brief introduction about robotics and swarm robotics in general is presented. Then, the foraging problem is addressed. After that, a mapping of the experimental scenarios into the real application of autonomous human rescuing after a semi-disaster happened is provided. Finally, the thesis outline is provided.

1.1 Robotics

The history of robotics [1] embeds computers, electronics, communications and mechanics technology. Thus, it can be considered as an interdisciplinary trend of these fields. Robotics nowadays represents one of the human's outstanding contributions and one of the major attempts for human to improve electronics and artificial life.

Although robotics can be considered as the main achievement of the 20th century, their corner stone starts long away in history. From all the beginning, superstitions were diffused through people about connecting the exceptional power of human with the artificial intelligence life. Since 270 before century Egyptians and Greeks attempts to perform easy tasks using mechanical machines. Nowadays, children's toys and other essential machines used in our daily life are automated using electronic devices.

Researchers are now more attracted in the building of artificial machines that own some intelligence to work autonomously, as the speed of improvements in the robotics and computer-engineering fields is extremely fast. Nowadays, robotics almost engage in a huge number of applications, they are involved in everything in our life such as exploration, military applications, police work, space field and medication industry.

To be fair, it is worth to mention that robotics field is not new as many people assume, however the origins of the robotics field [1] were started from 1250. In 1250, the first humanoid robot was implemented. Also, during the period from 1250 to 1950, the robots development field was transferred to be used more in entertainment field instead of serious domains.

A set of main stops [1] in robotics history in the 20th century can be addressed as follows:

1. In 1942, three rules were formulated and accepted for robots to be legally manufactured:

    1.1. Robots cannot be harmful to human being even during their operation. Also, Human is permitted to damage them if a harmful situation happened for him/her.

    1.2. If human gives a robot a command, which might contradict with the robotics first low, the robot should follow it.

    1.3. A robot should protect its survival in case this defense does not contradict with the robotics' first and second laws.

2. In 1956, the first company for robots' development was build.

3. In 1959, MIT confirmed the computer assisted manufacturing.

4. In 1961, the first automated robot was released online in a General Motors automobile plant.

5. The most outstanding year was 1963, as the first automatic robotic arm was developed mainly for the handicapped disable people.

These inventions [1], which are listed above was the first inventions in the field but not the only. Many inventions are developed too especially after the robotic arm. Through years, there were many surprising grants for human beings in the robotics field.

Branches that are involved in the robotics' establishment include:

Unlike other domains in engineering, robotics is a new multi-disciplinary field. The major domains, which are involved in the establishment of the robotics field include:

1. **Mechanical Engineering**: provides robotics field with the design, structure and machinery.

2. **Electrical Engineering**: Provides robotics field with the main control function and the artificial intelligence actions.

3. **Computer Engineering**: Provides robotics field with the implementation of robots' movements and sensing capabilities.

Classification of Robots:

Robots [1] are classified according to the design of the robots and the number of applications they can be involved in. Thus, robots can be categorized into three main classes:

- **Simple design Robotics -** This class includes automated machines of mostly simple design. The main target for such machines is just to help human beings, such as Washing Machine.

- **Middle design Robotics** – The main development in this class over the previous one is that the robots can be programmed but only once. They also own a more sophisticated circuit with sensors included and they mainly can execute multiple tasks, such as the newly sophisticated multi-function washing machine.

- **Complex design Robotics -** In this class, robots can be programmed multiple times. They also include very complex digital circuits, such as PCs and laptops.

Robotics types and applications:

One of the major trends in technologies for the past one hundred years is robotics field. Also, people became more aware and knowledgeable about technology and robotics due to the world-wide movies' industry and the global media. Nowadays, the number of tasks that can be executed by robots in different fields is progressively increasing. Some of these applications [1] can be outlined as follows:-

1. **Space robots** – tele-operated Robotic arms are used to construct a space station or to launch a satellite. These robots could be controlled remotely by human. Another example could be the ATHLETE (All-Terrain Hex-Legged Extra-Terrestrial Explorer); which is a

six-limbed robotic lunar rover test-bed. ATHLETE is a test bed for various systems that could be used for lunar or Martian exploration. As can be seen from figure 1.1, each of the ATHLETE's six limbs has six degrees of freedom. For general traveling purposes, the ATHLETE rolls on its six wheels but if it encounters more rugged and extreme terrain, it has the ability to lock each wheel into place and walk using its limbs.

Figure 1.1: ATHLETE robot [2].

2. **Domestic or household robots** – Robotics and electronic systems could be utilized currently in lots of applications in home to provide more prosperity to people's life. For example, they could be used in home security system, automatic system for doors and windows, and turning on and off electrical devices like A/C and lights. Examples that are more sophisticated are robotic vacuum cleaners (see figure 1.2), robotic pool cleaners, and robotic sweepers.

Figure 1.2: irobot [3].

3. **Exploration robots** – Robots can be deployed in dangerous semi-destroyed site after a disaster as an earthquake happened which could be harmful for human beings to get inside in an attempt to rescue victims, see figure 1.3. Another example could be the observation of the atmosphere within a volcano site.

Figure 1.3: Rescue robots [4].

4. **Military Robots** – As shown in figure 1.4 and 1.5, Robots are used nowadays in the modern armed force especially the flying robot drones, see figure 1.5. A common example on using robots in army could be using the airplane and automobiles in transmitting bombs, petroleum, or bullets (see figure 1.4). Other more sophisticated example could be using robots to clear minefields.

Figure 1.4: Military automobile robots [5]

Figure 1.5: An MQ-9 reaper during a training mission (flying robot drone) [6]

5. **Farm robots** – collecting crops could be done nowadays using autonomous robots, see figure 1.6. In addition, they could help workers to nourish and milk their cattle remotely.

Figure 1.6: Farm robots [4].

6. **The Car Industry (Industrial robots)** – Lots of cars' assembling and manufacturing process could be performed smoothly using robotic arms, see figure 1.7. Some of other tasks that could be performed easily using robots are painting, welding, sorting, bending, cutting, and lifting. They could also be used in the food industry to carry out jobs such as cutting and trimming of food like fish, beef, chicken or lamb.

Figure 1.7: Car assembled by robots [4].

7. **Medical robots** – Robots could be used in medical field to help nurses to lift patients while keeping their backbones in a straight position so that they could reduce the pain. In addition, a power-facilitated suit was invented in Japan, which could make it easier for the nurses to lift the patients.

8. **Entertainment robots** – these robots are very interactive and can be used for children's education. For example, there is a robot developed by SONY that could react to your voice commands, they also could move around freely, and even hold your luggage, see figure 1.8.

Figure 1.8: Humanoid robot [4].

9. **Hobby and competition robots** – These robots are designed mainly by students and can be used in competitions between students, such as the line follower competition.

10. **Underwater Robots -** Finally, underwater robots can be used to discover and gather essential information to be used by military, see figure 1.9.

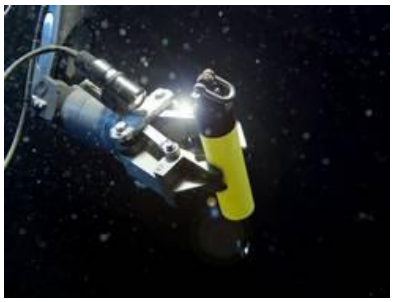


Figure 1.9: Underwater robots [4].

To sum up, it is better to use robots in recurring jobs, so that human could save their time and efforts for innovative, imaginative, and multifaceted tasks.

1.2 Swarm Robotics

The current trends in the robotics field have led to the development of swarm robotics systems. Swarm robotics is a new approach to the coordination of multi-robot systems, which consist of large numbers of mostly simple physical robots. It is supposed that a desired collective behavior emerges from the interaction between the robots and the interaction of robots with the environment. This approach emerged in the field of artificial swarm intelligence as well as the biological study of insects, ants and other fields in the nature, where a swarm behavior occurs. Most swarm intelligence researches are inspired from how the nature swarms, such as social insects, fishes or mammals, interact with each other in the swarm in real life.

The research on the swarm robotics is to study the design of large amount of relatively simple robots, their physical body and their controlling behaviors. The individuals in the swarm are normally simple, small and low cost to take the advantage of a large population. A key component of the system is the communication between the agents in the group, which is normally local, and guarantees the system to be scalable and robust.

A plain set of rules at individual level can produce a large set of complex behaviors at the swarm level. The rules of controlling the individuals are abstracted from the cooperative behavior in the nature swarm. The swarm is distributed and de-centralized, and the system shows high efficiency, parallelism, scalability and robustness.

The potential applications of swarm robotics include the tasks that demand the miniaturization, like distributed sensing tasks in micro machinery or the human body. In addition, the swarm robotics can be suited to the tasks that demand the cheap designs, such as mining task or agricultural foraging task. The swarm robotics can be also involved in the tasks that require large space and time cost, and are dangerous to the human being or the robots themselves, such as post-disaster relief, target searching, military applications, etc.

As an emerging research area, the swarm intelligence has attracted many researchers' attention since the concept was proposed in 1980s [7]. It now becomes an interdisciplinary frontier and the focus of many disciplines including artificial intelligence, economics, sociology, biology, etc. It was observed a long time ago that some species survive in the cruel nature taking the advantage of the power of swarms, rather than the wisdom of individuals. The individuals in such swarm are not highly intelligent, yet they complete the complex tasks through cooperation and division of labor and show high intelligence as a whole swarm, which is highly self-organized and self-adaptive.

Swarm intelligence is a soft bionic of the nature swarms, i.e. it simulates the social structures and interactions of the swarm rather than the structure of an individual in traditional artificial intelligence. The individuals can be regarded as agents with simple and single abilities. Some of them have the ability to evolve themselves when dealing with certain problems to make better compatibility. A swarm intelligence system [7] usually consists of a group of simple individuals autonomously controlled by a plain set of rules and local interactions. These individuals are not necessarily unwise, but are relatively simple compared to the global intelligence achieved through the system. Some intelligent behaviors never observed in a single individual would soon emerge when several individuals begin cooperating or competing.

The swarm can complete the tasks that a complex individual can do while having high robustness, flexibility and with low cost. Swarm intelligence takes the full advantage of the swarm without the need of centralized control and global model, and provides a great solution for large-scale sophisticated problems, such as using autonomous swarm of robots in mining or rescuing applications.

Most swarm intelligence researches [8] are inspired from how the nature swarms, such as social insects, fishes or mammals, interact with each other in the swarm in real life. These swarms range in size from a few individuals living in the small natural areas to highly organized colonies that may occupy the large territories and consist of more than millions of individuals. The group behaviors emerging in the swarms show great flexibility and robustness, such as path planning, nest constructing, task allocation and many other complex collective behaviors in various nature swarms.

The individuals in the nature swarm show very poor abilities, yet the complex group behaviors can emerge in the whole swarm, such as migrating of bird crowds and fish schools, and foraging of ant and bee colonies. It is tough for an individual to complete the task itself, even a human being without certain experiences finds it difficultly, but a swarm of animals can handle it easily. Researchers observed the intelligent group behaviors emerging from a group of individuals with poor abilities through local communication and information transmission.

1.2.1 Swarm robotics properties

Swarm robotics has been used a lot in recent years due to their unique advantages and properties. These properties could be listed as follows:

Robustness:

- *If a set of robots leave to recharge, to be maintained, or for any other reasons, the system will not be affected that much and it will continue towards finishing the whole tasks.*

Scalability:

- *Each particle in the swarm is separated from the other particles, and communicates with all of them and with the environment. This communication is not a global communication as each robot communicates only with its neighborhood. This characteristic enables the swarm to adapt to any change like, adding more robots or removing some robots without any damage in the whole process.*

Energy efficiency:

- *In swarm robotics, a large number of robots which are small in size and simple in design are deployed. Therefore, only small-size batteries are needed, that's why the cost of energy is much lower than that of a single but complicated-design robot.*

Economist:

- *As we said previously, the particles in the swarm robotics is small and simple in design with very limited capabilities, which make them very cheap. Thus, they could be produced easily in huge amounts with low cost.*

Parallelism:

- *There could be a set of targets "set of tasks", which should be implemented by different robots concurrently. This could significantly save effort and time.*

1.3 Autonomous Rescue Swarm Robotics Application

Nowadays swarm robotics is being utilized in different application domains of both real applications and academic research. For example, underwater and automatic space exploration can be considered as promising applications in this field. However, in this section, we are going to map our experimental scenarios into the search and rescue application within semi-destroyed site after a disaster like an earthquake that is dangerous for human to get inside in an attempt to search and rescue victims.

The first 24 hours after a natural or man-made disaster [9] are the most critical for the survival of victims. Unfortunately, this is also the time, when the fewest resources are available to rescuers. This research describes the potential for using a swarm of autonomous mobile robots to help the first responders to a disaster site focus their search for victims on those areas with the highest probability of finding survivors. Specifically, we are trying to present a scenario for deploying autonomous rescue robots' swarm at a semi-disaster site.

In this thesis, we explore the problem of robots allocation using a real world search and rescue group of robots, which is divided into two teams, searching team and rescue team. In our scenario, dedicated ‘searching’ robots must find victims, and the 'rescue' team transports them to a designated rescue zone. Each rescue robot tries to transport its allocated victim. It is also worth to mention that the system has no prior knowledge either about the number of rescue robots or of the number of the victims. Thus, the objective of such an application is to show that the system could:

I. Inspect victims located in the semi-disaster environment.
II. Efficiently assign robots to victims.
III. Finally, carry out victims successfully to the designated rescue zone.

Survival studies of earthquake victims [9] showed that the survival rate drops from 81 % for those victims rescued during the first 24 hours to 37% in the next 24 hours. Unfortunately, for a large-scale disaster, the resources available to rescuers, both personnel and equipment, are likely to be limited as it takes time for rescue teams to be mobilized and deployed to the disaster site. For this reason, a tool that could be deployed by the first responders to a disaster that would highlight locations of potential victims and hazards without the need for scarce human operators would appear to be a valuable asset to the on-scene incident commander. Therefore, an autonomous rescue robot swarm would be most useful during the critical first hours after a disaster occurs.

Stormont [9] suggested that the swarm robots should be inexpensive enough to enable wide spread prepositioning. The first responders to a disaster should activate and deploy the rescue

swarm robots. The swarm could search through as much of the site as the robots can reach and highlight areas containing victims that the rescue robots could focus on first and highlight potential hazards to be avoided.

Also, Stormont [9] mentioned that the robots would need to be small enough that they would not pose a threat to human in the scene (either by becoming obstacles or by causing a structural collapse), inexpensive enough that they would not need to be recovered, and operate only for a limited time before deactivating (which wouldn't be a problem for battery-operated robots).

However, stormont also [9] stated that small robots could also present challenges of their own such as in locomotion, sensing capability, and processing power. By maintaining redundant communications paths, the robots in the swarm can spread out into the disaster area to form an ad-hoc sensor network to detect victims and hazards, and then rescue those victims.

Grady and Rahmani [10, 11] declared that robustness and flexibility are two main characteristics that should be found in the robots to be used in such an application. There are two primary ways to build this type of systems either self-reconfigurable robotics or collective robotics. The main focus in this thesis is upon a novel hybrid algorithm, which can be categorized under collective robotics approach in which multi robots commonly collaborate with each other and with the environment to accomplish the required job.

They [10, 11] mentioned that in self-configurable field, by joining one or more small components together, a very complex element with a very high degree of freedom could be built. Self-reconfigurable robot has the capability to deal with complex problems through reshaping itself into legged robot or a snake-like structure. Swarm robots should have the ability to get across rough terrains as well as through narrow passages and cavities. Articulated rovers are commonly used for navigation through rough terrain conditions.

Also, Rahmani [11] announced that self-assembling robotics has lots of usages especially in space. For example, installation of a 10km long SSPS1 (Figure 1.10) costs more than 2500 hours

of astronaut space walk and more than $5 billion cost. A feasible strategy could be used to let most of the jobs done by the self-assembled robots and critical jobs done by astronauts.

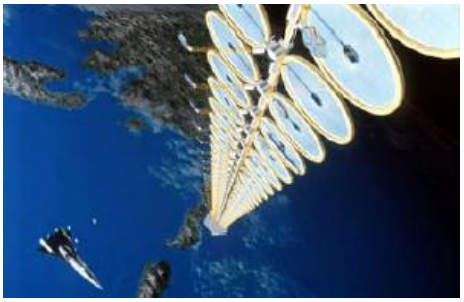


Figure 1.10: SSPS tower [11].

1.3.1 Challenges in search and rescue application

Rahmani [11] stated that there are different harmful situations that might damage the robots in the search and rescue application such as fire, water, large obstacles, explosions, fissures, deep vertical holes, small pebbles, large rocks, wires, walls, long tubes, compact blocks or even other agents within the disaster field. For example, Sometimes robots need to be introduced into small holes, and once inside they need to overcome large gaps, to descend a vertical duct ending in a large void, and finally to pass in other narrow passageways. Thus, to pass successfully through such situation, robots have to be very adaptable versatile depending on the situation they face.

Due to the high likelihood of robots' damaging within the unstable and unstructured arena, robustness is one of the most important characteristics in swarm search and rescue application. The control system must be able to work properly and continue its execution under any situation even if a considerable part of the system is damaged. Redundancy is a promising technique [11] that enables the system to have less dependency to a failure of a particular part of the whole system. The swarm must be robust enough to deal with the inevitable loss of robots in the swarm. In this research, a hybrid approach, which concentrates on a distributed control to employ redundancy, is proposed.

Rahmani [11] reported that while centralized control is suitable for systems with infrequent failure such as communication networks, distributed control is used a lot in the systems that require high fault tolerant, such as search and rescue application in extreme harsh environment using swarm robotics system.

Also, it was notified by Rahmani [11] that since any human operator usually has very restricted and limited perception from the arena in which the robot is navigating, thus pure tele-operating system is not sufficient to provide an acceptable performance of remote controlling. In addition, the sensing capability is limited by time delay. Therefore, autonomous control, which based on sensors that are mainly used for sensing the system's internal state allows performing the job efficiently and raises the system's capabilities.

The small robots advocated in this research tend to have poor odometry because the distance the robots move is only evaluated by dividing the speed of the robot over the time it moves to reach the new position and there is no feedback like encoders to enhance the accuracy of the readings. Moreover, it is extremely hard in the highly unstructured rescue environment to have high quality odometry even if we depend on encoders. Thus, as a future work, the swarm robots must attempt to determine their relative pose to some landmark using sensors that are more sophisticated or camera.

Also, Robin Murphy of the Center for Robot-Assisted Search and Rescue (CRASAR) [9] made the observation that "rescue workers refuse to consider fully autonomous systems designed to act as 'yes/no there's something down there' search devices. However, this skepticism regarding autonomous robot swarms and their integration into the rescue workers' hierarchy is understandable and well justified nowadays given the current capabilities of autonomous rescue robots.

1.4 Foraging Problem

A huge number of applications are involved in swarm robotics to make use of its significant characteristics, such as, post-disaster relief, geological survey, military applications, UAV

controlling, cooperative transportation, and mining. Swarm robotics is very promising in the applications, which contain environmental danger to human.

The focus of this research is upon a specific problem called the foraging problem. This problem [12] can be considered as one of the most significant problems in swarm robotics field because most of the swarm robotics applications somehow include this type of issue. For example, in the mining application, we need to inspect the environment for the mines first and then transport them from place to place and this is specifically called foraging. Same could be seen in search and rescue application and agriculture application and so on.

So, the cooperative foraging problem [13] can be illustrated as a team of robots that need to transport a set of objects, which are scattered in the arena to their final destination. The final destination of the objects called the nest. To be more precise, in this research we are trying to solve the assignment of the objects on the robots to transport them to the nest in an optimized way.

These set of objects can be one of two types [14], either a single-robot object, which needs only one robot to carry it from its current location to the nest or a multi-robot task, which requires at least two robots working simultaneously in order to retrieve the object to its final destination.

Therefore, the foraging task [13] can be addressed as a searching job followed by a transportation job. Each job consists of a number of tasks. Each robot in the foraging problem has to decide whether it would explore a new object or it is going to transport an explored object to the nest.

Also, if the robot chooses the transportation job, it has also to decide which one of the scattered objects it should select to return to the nest. The robot choice is not random. However, each robot chooses the most appropriate object to maximize the overall performance of the group. Also, it is worth to mention that, the DTA problem in a cooperative foraging scenario with shared task execution by multiple robots is an NP-hard problem.

Finally, the thesis structure can be outlined as follows. First, chapter one presents an introduction in robotics and swarm robotics in general. In addition, the properties of swarm robotics are listed. Also, a brief introduction about the foraging problem and optimization techniques, which are used in this thesis to tackle the DTA problem, is provided. Finally, a mapping between our experimental scenarios and a real application (search and rescue application) is illustrated in chapter one. Then, dynamic task allocation problem is introduced and a mathematical model of the problem is formulated in chapter two.

After that, chapter 3 provides a literature review for a set of techniques and results for solving the dynamic task allocation problem in swarm robotics field. After that, chapter 4 focuses on the thesis statement and the methodology. Furthermore, Chapter 5 presents the design and implementation phases of the proposed hybrid algorithm. In addition, the framework, which is used to discretize the Particle Swarm Optimization technique, is also addressed in this chapter. Moreover, chapter 6 discusses both the simulation and the real experiment results. Finally, chapter 7 presents the conclusion and the future works.

Chapter 2

Dynamic Task Allocation

In this chapter, the dynamic task allocation problem, which we are trying to tackle, is well defined within the framework of the foraging problem. Also, a mathematical model for the dynamic task allocation problem is formulated using integer linear programming.

2.1 Dynamic Task Allocation Problem Definition

Over the past few years, task allocation (TA) [15] has emerged as one of the most common issues that need to be optimized in the swarm robotics' field. Since in almost all swarm robotics applications there are a number of tasks that need to be performed by a group of robots, we need an optimized way by which we can assign those tasks to the robots in order to accomplish the whole job in a perfect way. Thus, TA problem can be considered as one of the most important direction in multi-robot systems nowadays.

So, TA problem [15] can be broadly defined as follows, given a set of tasks, a set of robots, that can perform those tasks, and an objective function, that measures the performance efficiency of different combinations of robots in performing tasks, a suitable matching or allocation needs to be found between the set of tasks and the group of robots, which optimizes the value of the objective function while satisfying operational constraints. The task allocation problem specifies the allocation rules used by the robots in such a scenario to perform the tasks in an efficient manner.

Dynamic Task Allocation (DTA) [16] implies that robots have no prior knowledge about the number of robots, the number of tasks, and the temporal and spatial distribution of tasks in the environment. However, they should communicate periodically with each other to adapt online to the environment. Minimal Time Dynamic Task Allocation (MTDTA) seeks to allocate tasks dynamically to the robots while minimizing the total time needed to accomplish the whole task.

With technical progress and the declining cost of robotic mobility, interest in this area of applications has grown significantly in recent years. The DTA problem [16] is encountered in different application domains of multi-robot systems including human rescuing, demining, warehousing, toxic waste cleanup, collection of terrain samples, cooperative transportation, autonomous exploration and mapping, distributed monitoring and surveillance, etc.

Since, we have just said that the minimal time dynamic task allocation (MTDTA) problem can be addressed as the allocation of a set of tasks to a group of robots to perform the whole job in a minimal time. Thus, The MTDTA algorithm can be described as follows:

let

$$T = \{t_1, t_2, \ldots, t_m\}$$

Equation (2-1)

be the set of task identifiers to be allocated to the robots in the swarm, and let

$$R = \{r_1, r_2, \ldots, r_n\}$$

Equation (2-2)

be the set of robots' identifiers in the robotic swarm. Therefore, the problem is composed of m valid tasks and n robots. The swarm allocation is represented by:

$$A = \{a_1, a_2, \ldots , a_n\}$$

Equation (2-3)

where aj identifies the task allocated to robot rj. Hence, the solution of the minimal time dynamic task allocation problem is achieved by finding an allocation A*, which represents the allocation of the group of n robots to the set of m tasks in a minimum completion time of the whole tasks.

The number of available allocations between a set of tasks and a group of robots [16] is one of the most important characteristics while dealing with the dynamic task allocation in swarm robotics. The number of allocations Q depends on the number of robots and the number of tasks.

If the robots in the swarm are heterogeneous, the number of allocations Q is defined by equation (2-4):

$$Q = \prod_{i=1}^{n} T_i^{Ri} = T_1^{R1} \times T_2^{R2} \times T_n^{Rn} \ ,$$

Equation (2-4)

Where R is the number of robots in the swarm, n is the number of groups to which those robots belong, where a group of robots Ri is responsible for performing the set of tasks Ti. Hence, if the robots in the swarm are homogeneous and we only have one group in the swarm, then the number of allocations Q is defined as expressed in equation (2-5) [15]:

$$Q = T^R$$

Equation (2-5)

In this research, a minimal time dynamic task allocation algorithm in swarm robotics is designed and implemented. The allocation decisions procedures are taken by using a hybrid algorithm combined both the Discrete Particle Swarm Optimization (DPSO) technique and Simulated Annealing technique.

The hybrid DPSO/SA is a stochastic algorithm based on swarm intelligence that can be applied to search iteratively for a solution for optimization problems within the search space. Each particle in the swarm updates the position in which it is located and the velocity by which the particle travels in the environment in each iteration of the algorithm.

The algorithm uses every collected data from the environment, which is provided by the robots' sensors, like infrared proximity sensors and ground sensors in order to come up with the optimal allocations. In the first step, the environment information should be gathered. Then, the transportation time required by each robot to carry out each task must be calculated. After that, the optimal paths for all subtasks should be selected.

2.2 Problem Formulation (Minimal Time Dynamic Task allocation)

The Minimal Time Dynamic Task Allocation of a set of tasks among homogenous robots in swarm robotics system is a minimization problem that should satisfy specific constraints. Hence, this problem [17] can be considered as a constrained optimization problem. The multi-robot task assignment problem in which every task should be allocated to only one robot, and each robot must assign itself to exactly one task, can be considered as an instantaneous assignment instance. This problem is well known as the single-task robots, single-robot tasks problem. More officially, given a set of n available robots R = {r1, r2, . . . , rn}, a set of m available tasks T = {t1, t2, . . . , tm}, and let C = (Cij)n×m be the cost matrix, where Cij represents the cost of allocating robot i to transport task j, then the target is to find a one-to-one mapping which minimizes the overall cost.

Let Xij be a binary variable denote the allocation of a robot i to a task j, so that Xij is equal to 1 if robot i is assigned to task j and 0 if unassigned. Therefore an assignment matrix can be presented as in equation (2-6):

$$X = (Xij)_{n\times m} = \begin{pmatrix} X_{11} & \cdots & X_{1m} \\ \vdots & \ddots & \vdots \\ X_{n1} & \cdots & X_{nm} \end{pmatrix}$$

Equation (2-6)

Also, taken into consideration that the allocation problem [17] is a one-to-one mapping, so there must be only one element equals 1 in each row and each column of the matrix X and the rest of the elements should be 0s.

The target of the optimization technique is to find a feasible solution, which minimizes the objective function. There are many parameters that could be minimized depending on the requirements of the problem itself, for example, the cost that need to be minimized could be the traveled distance, energy consumption, or execution time. In the MTDTA, the selected objective is to minimize the distance the robots travelled in order to transport the tasks from their current location to the nest. In other words, during performing the tasks, J defined as the distance

travelled by all the robots to accomplish all the tasks in the search space at any given time instance. It is also worth to mention that the time to complete the task would also be minimized as the distance minimized because all the robots are moving with a constant speed [18]. The constrains which must be satisfied during the optimization can be outlined as follows:

1. A specific number of robots carry out the inspection, and the rest of the robots engage in the transportation job. If no objects need to be transported at any time instant, the robots which are responsible for the transportation turnover to the sleep mode in order to save as much power as we can.
2. A robot cannot be assigned to more than one task at the same time.
3. Each task is executed only once with exactly one robot.

Now, the optimization of the multi-task assignment problem can be formulated in terms of an integer linear programming as follows [17]:

$$minimize\ f(R,T) = min \sum_{i=0}^{n} \sum_{j=0, j\neq i}^{m} C_{ij} X_{ij}$$

Equation (2-7)

Subject to:

$$\sum_{j=0}^{m} X_{ij} = 1, \qquad \forall i \in n,$$

Equation (2-8)

$$\sum_{i=0}^{n} X_{ij} = 1, \qquad \forall j \in m,$$

Equation (2-9)

$$x_{ij} \geq 0, \quad \mathrm{x_{ij}} = \{0,1\} \quad \forall i \in n,$$

Equation (2-10)

where Equation (2-7) represents the objective function which is used to minimize the travelled distance, and x_{ij} gives a primal variable. Therefore, the problem [17] as an outcome of the constraint matrix's structure turns into an integer problem and $x_{ij} = \{0, 1\}$ implies that if the robot i is assigned to the task j, then the value of the x_{ij} is one, otherwise zero. Also, these constraints guarantee that a robot cannot be assigned to two tasks at the same time and a task cannot be executed by to two robots simultaneously which guarantees that every task is executed only once.

Chapter 3

Thesis Statement and Methodology

In this chapter, both the thesis statement and methodology for this research are addressed.

3.1 Thesis Statement

Although using global optimization techniques in solving discrete optimization problems like DTA problem, provides a prominent convergence characteristics, and using local optimization techniques yields better quality solution due to the probabilistic jumping mechanism, however, using a hybrid approach which combines both local and global searching capabilities could result in outstanding performance (the minimal time in which the whole tasks is accomplished in the DTA problem) since it increases the scalability, stability, and efficiency of the swarm.

3.2 Methodology

The main steps that are followed to accomplish this research could be summarized as follows:

- Implement a discrete version from the Particle Swarm Optimization technique, test it on a set of different swarm sizes and compare it with the Simulated Annealing technique.
- Formulate the dynamic task allocation problem using integer linear programming, and provide an exact solution to the problem using the Hungarian algorithm as an exact solver.
- Design and implement a hybrid approach which combines both the Discrete Particle Swarm Optimization technique and the Simulated Annealing technique.
- Simulate the proposed algorithm using MATLAB on different problem dimensions to prove the scalability of the proposed algorithm.
- Compare the execution time and fitness values between both the exact solver and the proposed meta-heuristics optimization techniques.
- Deploy and test the proposed hybrid algorithm into a real system using 4 Elisa-III robots.

- Tune the proposed algorithm's parameters to enhance the performance of the system and to improve the accuracy of the outcomes.
- Evaluate the total communication overhead, and the system throughput while transmitting and receiving packets between the robots and the base station.

In details, Simulated Annealing technique was used in some contexts to achieve a minimal time dynamic task allocation solution in swarm robotics. Therefore, a discrete version from the Particle Swarm Optimization technique is implemented, tested, validated and compared in this research with the Simulated Annealing technique. A comparison between SA and DPSO is performed in terms of computational time, number of iterations needed and quality of solution to demonstrate the robustness and efficiency of the DPSO algorithm in such an optimization problem.

Two case studies are considered while comparing the DPSO and the SA techniques to prove the proficiency of the DPSO over the SA algorithm in small swarm sizes. The first study-case is implemented by introducing 7 robots to transport 10 prey to the nest, while the second study-case is implemented using 10 tasks and 20 robots.

In addition, a novel hybrid approach that fuses the DPSO technique with the SA technique is also implemented and tested using larger problem dimensions. Then, this approach is compared with both the Discrete Particle Swarm Optimization and the Simulated Annealing technique independently to highlight the advantages and disadvantages that could result from combining DPSO with SA. This comparison also investigates the challenges that could be addressed using the proposed algorithm such as getting trapped into a local optimal solution problem, time complexity problem, and scalability problem.

About 15 different swarm sizes are tested in this simulation using all the three algorithms to investigate the scalability and robustness of each one of them and to show the superiority of the proposed algorithm in comparison with the other traditional optimization techniques (SA and DPSO techniques). These study-cases range from 7 robots and 10 tasks to 2000 robots and 3500 tasks.

Moreover, the DTA linear equations are formulated using integer linear programming. Then, using these linear equations, the Hungarian algorithm is used as an exact solver in order to provide an exact solution to the problem. After that, a comparisons in terms of the execution time and quality of solutions are performed between the exact solver and the other optimization techniques. Also, the exact solutions of the Hungarian algorithm are used as a model to normalize the outcomes from the other techniques.

Furthermore, the hybrid DPSO/SA algorithm is deployed and extensively tested into a real system using 4 Elisa-III robots. The robots' peripherals are examined first to make sure everything is working properly. Thus, all the sensors such as ground sensors, the infrared proximity sensors, the 3-axis accelerometer are tested. Also, peripherals like the transceiver module, the infrared emitter and receiver, the dc motors, the central RGB LED, and the green LEDs are all examined to make sure that they are ready to be used.

In the real experiment, objects are scattered randomly in the environment at distinctive positions and they are perceived by the robots' ground sensors. Also, the Infrared sensors are used for collision avoidance.

Finally, the communication overhead in the real experiment is evaluated in both the transmitter side and the receiver side to make sure that there is enough throughput for the robots to exchange information periodically during the experiment.

Chapter 4

Algorithms For Solving Dynamic Task Allocation Problem

In this chapter, the most commonly used taxonomies for the classification of optimization techniques in terms of research and publications are addressed and examples from literature review for each taxonomy are outlined. Then, we address the discrete optimization techniques that could be used for solving discrete optimization problems and some applications on these discrete problems are outlined. Also, challenges that could face such discrete optimization technique are illustrated and solutions from the literature are summarized. After that, the most outstanding optimization techniques in solving the dynamic task allocation problem are introduced which are the Simulated Annealing technique and the Particle Swarm Optimization technique.

At the beginning, it is worth to mention that discrete optimization problems like the DTA problem [19] are showed up extensively in applications related to computer science and engineering. The heuristic optimization techniques' significant has increased a lot in last decades. They can be considered as a very good and flexible candidate for solving the real life problems, as they are most commonly non-differentiable and nonlinear. Thus, the design of a stable efficient algorithm for finding near global minimum solution has attracted the attention of many researchers in the last few years.

Recently, several algorithms have been proposed in the literature for solving discrete optimization problems. Based on the taxonomy presented by Zhang and Liu [8], the classification of algorithms is implemented according to either its behavioral approach, market laws or bio-inspired approach.

4.1 First Taxonomy

4.1.1 Behavioral approach

In the behavioral approach, tasks to be executed are differentiated in groups, called behavioral groups. Groups have a set of tasks to be executed, which have a relation between them. One of the most common used behavioral algorithms is ASYMTRE presented by Tang and Parker [20].

4.1.2 Market Laws approach

Algorithms based on market laws aim at maximizing the income (information, speed), while minimizing costs (convergence, communication time). For example, as cited by Guerrero and Oliver [21], a market laws based algorithms in which the system is partially inspired by auction and thresholds-based methods and tries to determine the optimum number of robots that are needed to solve specific tasks, or sub-goals, in order to minimize the time needed to complete all tasks. Since the time to perform tasks by robots is considered to be much less than robot navigation times between tasks, so the order in which robots perform tasks is important. Thus, the distance between two subsequent tasks needs to be minimized and as a result, the navigation time would be minimized. The algorithm constraints were as follows:

1. Same task cannot be allocated to the same robot more than once.
2. The total number of robots allocated to perform a task equals the demand for the task.

While, the cost functions' mathematical equations were as follows:

$$min \sum_{ri \in R} (\left\| p_{ri}^{0} - p_{T_{ri}^{1}} \right\| + \sum_{(T_j, T_K) \in A(R)} \left\| p_{Tj} - p_{Tk} \right\|)$$

Equation (4-1)

Subject to:

$$T_j \neq T_K \quad , \forall (T_j, T_K) \in A(R)$$

Equation (4-2)

and

$$| A(R)| = d_{Ti} \quad , A(R) = T_i \quad \forall T_i$$

Equation (4-3)

Another example proposed by Zhang et al. [22] is a Stochastic Clustering Auction solution, which uses a Markov chain search process along with Simulated Annealing. This is the first time in which a stochastic auction technique is used along with a global optimization technique. A swapping movements and transfers among tasks' groups, which are allocated to distinct robots is the main principle in this technique in which both the downhill movements and uphill movements are taken into consideration, in order to reach the global minima and avoid the local ones. The essential difference in this algorithm is that, after reaching a proper convergence, the algorithm performance could slide slightly in the small region between the global optimum and a random solution, which is tuned based on the Simulated Annealing technique. The cost function for this algorithm was chosen to be "Minimizing the time taken to accomplish the tasks or the mission length (the total distance traveled)". Also, the cost functions' mathematical equations were as follows:

$$\min_A C(A)$$

Equation (4-4)

$$C(A) = \sum_{s=1}^{k} c_s(a_s)$$

Equation (4-5)

Where cs(as) is the minimum cost for robot hs to complete the set of tasks as.

$$\begin{cases} C\left(A_i^{(s,p)}\right) = C(A) + \left(c_s\left(a_s^{(-i)}\right) - c_s(a_s)\right) + \left(c_p\left(a_p^{(+i)}\right) - c_p(a_p)\right) & Single\ Move \\ C\left(A_{i,j}^{(s,t)}\right) = C(A) + \left(c_s\left(a_s^{(-i,+j)}\right) - c_s(a_s)\right) + \left(c_t\left(a_t^{(+i,-j)}\right) - c_t(a_t)\right) & Dual\ Move \end{cases}$$

Equation (4-6)

One more example is the heuristic search-based task allocation algorithm formulated by Nagarajan and Thondiyath [23] for the task processing in heterogeneous multiple robot system, by maximizing the efficiency in terms of both communication and processing cost. The cost function in this algorithm was chosen to be the summation of static overhead cost of robots, assignment cost or execution cost, and the communication cost between the dependent tasks, if they are assigned to different robots. While the optimization constraints were:

a. Each task is allocated to only one robot.
b. The capability of robots to cover all tasks must be guaranteed.
c. The precedence relation between the tasks must be taken into consideration. If there is a precedence, time for task I must be before time for task j.

Therefore, the mathematical equations for the objective function were defined as follows:

$$Z = \sum_{i=1}^{N}\sum_{k=1}^{m} S_k X_{ik} + \sum_{i=1}^{N}\sum_{k=1}^{M} S_k R_{ck} X_{ik} + \sum_{i=1}^{N}\sum_{k=1}^{M} E_{ik} X_{ik}$$

$$+ \sum_{i=1}^{N-1}\sum_{j=1}^{N}\sum_{k=1}^{M} Pr_{ij}\, C_{ij}\, X_{ik} X_{jk}$$

$$+ \sum_{i=1}^{N-1}\sum_{j=1}^{N}\sum_{k=1}^{M} Pr_{ij}\, C_{ij}\, X_{ik} X_{jk}$$

Equation (4-7)

Subjected to constraints:

$$\sum_{k=1}^{M} X_{ik} = 1 \qquad , i = 1,2, \dots \dots, N$$

Equation (4-8)

Where X_{ik} implies task i is allocated to robot k then the value is 1 else 0.

4.1.3 Bio-Inspired approach

One of the most trend techniques, which are used to solve dynamic task allocation problem is the bio-inspired techniques, which have been used a lot in recent years in terms of research and publications. These techniques are derived from social insect's behavior like bees and ants. A set of beneficial properties, such as flexibility and self-arrangement property are existed in the Particle Swarm Optimization technique.

As mentioned by Krieger et al. [24], it is shown that a higher level of energy could be maintained in sets of robots, which use ant-inspired optimization techniques than in single robots. They could also forage more expeditiously.

Liu et al. [18] have presented approach that is based on artificial bee colony algorithm to address dynamic task assignment problems in multi-agent cooperative systems. Also, Liu et al. [18] have enhanced an optimal assignment method based on a task-swap mechanism. They have proposed an algorithm in which local search processes are executed simultaneously without any connections among these processes. In addition, this technique is totally decentralized in which messages' broadcast and a multi-hop communication would never be used. The solutions resulted from the local processes is related to a shortest path routing problem on a graph subject to the network topology. This conclusion was resulted after the analysis of the formulation by using the uncommon tools from the group and the optimization duality theories.

The assignment problem was formulated using a pair of linear equations. The cost minimization formulation called the primal program P(R, T):

$$minimize\ f(R,T) = \sum_{i \in R, j \in T} c_{ij}\, x_{ij}\ ,$$

Equation (4-9)

Subject to:

$$\sum_{j \in T} x_{ij} = 1, \qquad \forall i \in R,$$

Equation (4-10)

$$\sum_{i \in R} x_{ij} = 1, \qquad \forall j \in T,$$

Equation (4-11)

$$x_{ij} \geq 0, \quad \forall i \in R, \qquad J \in T,$$

Equation (4-12)

where each x_{ij} represents a primal variable. Because of the structure of the constraint matrix, the problem turns into an integer problem and eventually each x_{ij} should equal 0 or 1 in the solution when solved via some combinatorial optimization algorithm.

The constraints:

$$\sum_{j} x_{ij} = 1$$

Equation (4-13)

and

$$\sum_{i} x_{ij} = 1$$

Equation (4-14)

This constraint guarantees that a robot cannot be assigned to two tasks at the same time and a task cannot be allocated to two robots simultaneously.

Momen and Sharkey [25] have evolved The ant's behavior of switching tasks in order to meet the changing demand to increase their ability to respond to task demand effectively. Nedjah et al. [15] have proposed a distributed control algorithm inspired by the Particle Swarm Optimization implemented to perform dynamic task allocation in a swarm robotics environment. The fitness evaluation of an allocation A is implemented by an objective function f(A), defined by equation (3-15):

$$f(A) = \frac{\sum_{i=1}^{T} |C[i] - C_A[i]|}{T},$$

Equation (4-15)

where T represents the whole number of task, C represents the desired quantity of robots allocated to each task, according to a desired rate, and CA represents the quantity of robots allocated to each task, according to A.

Finally, Tsalatsanis et al. [26] have derived Fuzzy-logic-based utility functions. These utility functions are used to assign a group of robots to a set of tasks through determining the robots' capabilities to implement a task in real time by using a bounded look ahead control technique, which based on the main rules of discrete event supervisory control theory. The cost function that was used in this algorithm is maximizing the overall performance of the robot team. Robot's abilities such as endurance, designer's choice, cost of the robot, number and types of sensors, the distance a robot has to travel to perform a task, the cost of assigning a robot to a task and efficiency can be described as fuzzy variables. The utility function value of the events {start_ jk} is equal to the ability of Robot j to perform Task k. In other words,

$$u(\sigma) = ability_{jk} \quad , \text{where} \quad \sigma \in \{\text{start\_jk}\}$$

Equation (4-16)

Both the designer's choice and the robot's efficiency depend on the robot's firmness. Thus, the designer's choice is high and the robot's efficiency is high when the robot's firmness is large, then the capability of the robot j to implement the task k is high. While, the designer's choice is

medium and the robot's efficiency is medium, when the robot's firmness is acceptable. In this case, the capability of robot j to implement task k is medium. Finally, the designer's choice is low and the robot's efficiency is low, when the robot's firmness is small, then the ability of robot j to implement the task k is low.

4.2 Second Taxonomy

Whenever an alteration is sensed in the environment, a task allocation process needs to be performed between the robots and the tasks. In this case, the task allocation process is considered to be a dynamic process. The centralized approach can be considered as a fast solution to this type of problem. However, if a quite preferable solution is needed, a distributed assignment technique will be better. No centralized leader or controller is needed in this approach, and that is why tasks' scheduling in swarm robotics must be performed because of this distribution process. However, one of the main disadvantages, that result from this decentralization is that problem complexity is increased, because the robot view in the environment becomes limited.

4.2.1 Centralized approach

In centralized approach, there is a leader or central unit which is responsible for tasks assignment to the robots. Gigliotta et al. [27] have evolved a dynamic task allocation rules by communicative interactions in a group of homogeneous robots. They focused on the development of a team of robots in which one and only one individual robot (the 'leader') must differentiate its communicative attitude from that of all the others ('non-leaders'). The robots evolve their capabilities to distinguish their roles by the discrimination of their signals. The leader robot has to maximize the value of its communicative signal, while all other robots have to minimize their signals' value. The leader robot tends to send high signal's value, while the non-leaders tend to send a low signal's value. The fitness of a group of robot was calculated in the following way. The average of the differences between the output signal of the current leader which has the maximum value and the output signals of all other non-leader robots was calculated every iteration. A number of trials need to be implemented. At the end, the average of the calculated value for all iterations of all the trials can be considered as the fitness value.

Formally, this is how fitness value was calculated:

$$F = \frac{\sum_j^C \sum_i^N Max - O_i}{C(N-1)},$$

Equation (4-17)

where the number of robots in each group represented by N, such as 10. While, the total number of iterations of each individual called C, such as 1000 iterations * 40 trials = 40000. Also, the signal's value of the current leader is Max and the signal's value of robot j is O_j.

4.2.2 Distributed approach

In distributed approach, there is no central unit to take care of the task allocation. So, each robot in the swarm has to identify the task it must perform. Brutschy et al. [13] have developed a self-organized method for allocating the individuals of a swarm to tasks that are sequentially interdependent. The proposed method does neither rely on global knowledge nor centralized components. Moreover, it does not require the robots to communicate.

De Mendonça et al. [16] have proposed a simple yet efficient distributed control algorithm to implement dynamic task allocation in a robotic swarm, in which every robot in the swarm has to identify the task it must perform. The swarm task assignment is represented by A = {a1; a2;…; aρ}, wherein aj identifies the task assigned to robot identified by idj. Note that given a task assignment, say A, it is possible to set counters associated with it, which denoted as CA, as described in the following equation:

$$C_j = C_A[t_j] = \sum_{r=1}^{p} \emptyset(a_r, t_j),$$

Equation (4-18)

Wherein function ∅ is defined as follows:

$$\emptyset(a_r, t_j) = \begin{cases} 1 & if\ \ a = t; \\ 0 & otherwise \end{cases}$$

Equation (4-19)

Thus, at the end of the construction, the counter set C_A would reflect the distribution of the robots as assigned to each task and defined by assignment A. Solving the dynamic task allocation problem consists of finding the task assignment A* = {a*1; a*2;…; a*ρ}, that verifies:

$$\forall\ t_j \in T\ \ and\ \ \ \forall\ p_j\ \in P.$$

Equation (4-20)

Keshmiri and Payandeh [12] have presented the percentile values of the distributional information of the tasks to reduce the task space into a number of subgroups that are equal to the number of robotic agents.

Liu et al. [17] have enhanced an optimal assignment method with a decentralized leadership. They have proposed an algorithm in which local search processes are executed simultaneously without any connections among these processes. In addition, this technique is totally decentralized in which messages' broadcast and a multi-hop communication would never be used.

Zhang et al. [28] have established hierarchical allocation architecture for the set of robots in the population. Two algorithms are implemented in this hierarchy. First, a simple self-reinforcement learning model is utilized in the top level of the hierarchy which turned the initially identical particles into specialists for distinctive task types by using the social insects' method. This method gives a robust and flexible labor's division as can be seen from figure 4.1. However, when the single type of task is considered in the lower level of the hierarchy, the Ant-colony technique is implemented to solve the task assignment issue. A local communication technique is used to share information among robots to avoid using a leading or a centralized device.

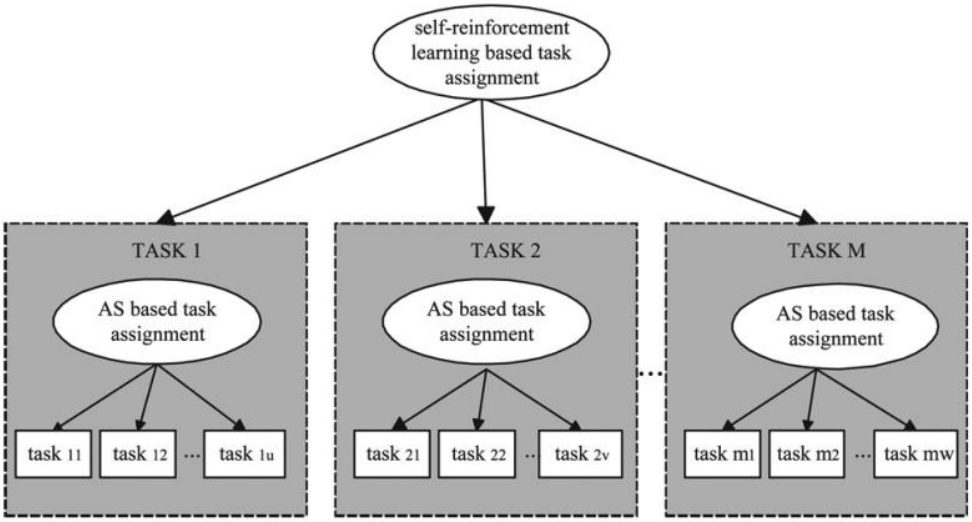


Figure 4.1: Architecture of the relationship among tasks

4.3 Third Taxonomy

In addition, taxonomy in [29] by Gerkey and Matarić has classified tasks in dynamic task allocation problems into single-robot tasks (SR) and multi-robot (MR) tasks.

4.3.1 Single-robot task approach

A single-robot task requires only one robot to perform a specific task. This dynamic task allocation problem is an NP (non-deterministic polynomial-time) hard optimization problem.

4.3.2 Multi-robot task approach

However, a multi-robot task has to be cooperatively carried out by several robots. Dynamic task allocation problems with multi-robot tasks are strongly NP-hard and the complexity significantly depends on the number of multi-robot tasks and the number of robots required by each multi-robot task.

Campo and Dorigo [30] have identified an efficient multi-foraging behavior, where is efficiency defined as a function of the energy that is spent by the robots during exploration and gained

when a prey is retrieved to the nest. A set of differential equations was used to model the flows of robots exchanged among the states as follows:

$$\frac{dE}{dt} = -\beta E + \gamma I + \sum_{i=1}^{2}(-\pi_i E N_i \lambda + \mu_i R_I)$$

Equation (4-21)

$$\frac{dB}{dt} = +\beta E - KB$$

Equation (4-22)

$$\frac{dI}{dt} = +KB - \lambda I$$

Equation (4-23)

$$\frac{dR_i}{dt} = \pi_i E N_i \lambda - \mu_i R_i - pR_i \qquad \forall i \in [1,2]$$

Equation (4-24)

$$\frac{dN_i}{dt} = \varphi_i - \pi_i E N_i \lambda - \varepsilon_i N_i + pR_i \qquad \forall i \in [1,2]$$

Equation (4-25)

Where the number of robots in Explore state is referred as E, and the number of robots in the Back state is referred as B. Robots in the Back state is going to the nest in order to have some rest and recharge their batteries. In addition, the number of robots in the Rest state is referred to l, the number of robots in Retrieve state is referred as R_i, in which "i" is the prey's type, and the number of prey which have the type i in the environment is referred as N_i. While, the total number of robots in the experiment is referred as T, the rate by which objects were found in the environment by a single robot is referred as λ, and the probability for a single robot to find the nest is referred as K. Moreover, φ_i is the incoming rate per second of prey of type i, while the probability constant over time for a prey of type i to disappear is referred as ε_i. In addition, μ_i is

the inverse of the average time, required to retrieve a prey of type i, the probability to give up an ongoing retrieval is referred as ρ, the probability for a robot to return to nest is referred as β, γ is the probability for a robot to leave the nest and look for prey, and finally the probability to take a prey of type i upon encounter is referred as π_i.

Dasgupta [31] has developed an algorithm in which each task required multiple robots to share the task's execution to complete the task. A set of heuristics that can be used by the algorithm were described to select the order of the tasks so that the tasks are performed in an efficient manner.

Liu and Kroll [14] have discussed that, it is more challenging to allocate a huge set of tasks to a group of robots if the tasks are tightly coupled than if the tasks is loosely coupled. For example, it is more challenging to find a suitable allocation for tasks that need the cooperation of at least two robots in order to carry out the task concurrently. The resulting complexity is due to the environmental temporal and spatial constraints. Moreover, the task assignment complexity rises exponentially with growing the number of task's types.

Liu and Kroll [14] have focused on multi-robot task allocation in inspection problems with both single- and two robot tasks. They have proposed a novel Mimetic algorithm combining a genetic algorithm with two local search schemes. The optimization parameter that is used in this research is the Completion time of all tasks. The optimization was done with the following constraints:

- N robots carry out the inspection.
- Every subtask is executed only Once.
- Each robot starts and ends at the home base.
- Robots must be different for the same task.
- All robots in the same task must work together to accomplish the task successfully.
- No one robot assigned to two tasks at the same time.
- Finally, tasks, assigned to robots must be executable.

In addition, the optimization mathematical equations that were used are as follows:

$$J = \max_{K\epsilon\{1,\dots,N^R\}} \sum_{i=0}^{N^P} \sum_{j=0, j\neq i}^{N^P} C_{ijk} X_{ijk}$$

Equation (4-26)

Subject to:

$$\sum_{k=1}^{N^R} \sum_{j=0}^{N^P} X_{0jk} = N^R$$

Equation (4-27)

$$\sum_{k=1}^{N^R} \sum_{j=0, j\neq i}^{N^P} X_{ijk} = 1 \qquad \forall i, i = 0,1, \dots\dots, N^R$$

Equation (4-28)

$$\sum_{j=1}^{N^p} X_{0jk} = \sum_{i=1}^{N^p} X_{i0k}\ , \qquad \forall k, k = 0,1, \dots\dots, N^R$$

Equation (4-29)

4.4 Discrete Combinatorial Optimization Techniques

Since, the dynamic task allocation problem demands the allocation of N robots to M tasks given the distance matrix (C) between the robots' positions and the tasks' locations; the target is to find the allocation, which results in a minimum cost. The minimal time dynamic task allocation could be mathematically outlined as follows:

$$f(R,T) = \sum_{i=0}^{n} \sum_{j=0, j\neq i}^{m} C_{ij} X_{ij}$$

Equation (4-30)

where N represents the total number of implemented robots, M is the number of currently detected tasks, Cij represents the distance between robot i and task j, and x_{ij} ={0,1} implies task i is allocated to robot j. The dynamic task allocation problem is considered to be an NP hard problem and is believed to be one of the most complex optimization problems; particularly for large swarm size (n > 100).

4.4.1 Combinatorial discrete applications

In the literature, local search techniques like Simulated Annealing and Tabu Search are located to be more effective than the global searching techniques for solving the dynamic task allocation problem. Many applications in real life can be classified under the discrete DTA category such as, facility layout problems.

4.4.1.1 Facility layout problem

As can be identified in [32], in the facility layout problem, an optimal location of facilities must be chosen with an objective of minimizing the estimate cost of transportation while considering certain constrains such as preventing the placement of hazardous materials near housing or the placement of the facility close to the other competitors' facilities.

4.4.1.2 Scheduling problem

Another example could be the scheduling problem as defined in [33], in which a finite number of resources with different processing power should allocate themselves to various tasks of different processing time, while trying to minimize the total time to accomplish the whole tasks. The whole tasks will be accomplished when the processing of all tasks is completed using the available resources. In most real-life problems, the problem is introduced as a dynamic problem (online scheduling). Hence, the task must be introduced to the algorithm for the tasks' allocations decision process to be taken online.

4.4.1.3 Travelling salesman problem

Also, one of the most well-known applications is travelling salesman problem as specified in [34], in which all the cities should be visited exactly once using the shortest way and the

algorithm must returns back again to the first city visited after the tour. In this problem, the distances among cities and a list of cities locations are given to be able to find the shortest tour.

4.4.1.4 Maximum clique problem

Maximum clique problem is also one of the most famous optimization problems in the field of information sciences as clarified in [35]. It is a discrete computational problem, which aims at finding a collection of vertices which are all adjacent to each other. This is also termed as complete sub graphs in a full graph.

In addition, various formulations established based on how to choose the cliques and what are the knowledge and data we have about each clique. For example, finding all the optimal cliques that cannot be magnified more, finding the most optimal clique among all the other cliques which contains the maximum number of vertices, trying to find out a solution to the decision issue by checking if a given clique is bigger than a specific size, and listing the most weighted cliques in a given weighted graph, are the most commonly used formulations in the maximum clique problem.

4.4.1.5 Bin packing problem

Another popular problem is the bin-packing problem where as set of containers or bins must be filled with objects that have various volumes as explained in [36]. The packing must be done in such a way that tries to minimize the used number of containers. This problem considered to be a combinatorial NP-hard in the discrete complexity theory.

Many alternatives found in this problem such as linear bins packing problem, packing by cost, packing by weight problem and 2D bins packing problem, etc. The applications for the bins packing problem include, weight capacity constraints trucks' loading, generating media file backups, bins' fill up, and technology mapping in Field-programmable gate array semiconductor chip design.

The bin-packing problem can be also considered as a special category from the cutting stock problem as mentioned in [36]. There is also another problem similar to the bins packing problem

called the knapsack problem. However, there is only 1 container and every object is specified by a volume and a value instead of just a volume in the knapsack problem. This problem aims at maximizing the objects' value that could fill in the container.

4.4.1.6 Graph partitioning problem

One last example is graph partitioning which is defined in [37] as information symbolized in the form of a graph G = (V,E), in which V represents vertices and E represents edges. In such problem, we must be able to divide a specific graph into a set of small partitions with given properties. For example, a k-way problem divides a large vertex graph into k smaller groups. The smaller the number of edges among the various groups the better the partition is.

Examples from the computer engineering field include, minimizing the wires among the electronic devices, which try to connect the backboard and optimizing the layout of the digital signal processors' memory. Moreover, a more significant application could be the analysis of the reaction chemistry.

4.4.2 PSO limitation in solving discrete combinatorial problem

As I mentioned before, it is proven that meta-heuristics optimization techniques are fascinating options to solve the dynamic task allocation problem. Through the last decade, various natural based meta-heuristics optimization techniques have been used by Benlic [38, 39] to solve the dynamic task allocation problem. Also, a lot of hybrid and memetic techniques that combine both local searching approaches with a natural swarm based techniques were proposed and proven to be very robust and efficient.

Mangat, Kulkarni, and AlRashidi [40, 41, 42] showed that almost all of these approaches need to be adapted to fulfill the requirements of these combinatorial discrete optimization problems and lots of researchers spend lots of effort and time to be able to find adequate solution to these combinatorial problems. However, Particle Swarm Optimization technique, which has proven to give a superior performance in lots of continuous applications, has not received similar attention at the combinatorial discrete direction.

They also illustrated [40, 41, 42] that the superiority of the Particle Swarm Optimization technique over the other natural-based techniques is due to its simplicity. Since the Particle Swarm Optimization technique is not inherently discrete in its basic canonical shape, it is not the most suitable choice to be applied to solve the discrete optimization problem, specially the dynamic task allocation problem. This problem is the reason after always applying the Particle Swarm Optimization technique in lots of continuous domain applications and avoiding using it with the discrete domain.

In the dynamic task allocation problem in swarm robotics, each robot has to assign itself to the nearest and most appropriate task of the all the other tasks in the problem. However, unlike the ant colony optimization technique which is inherently discrete in its basic form, PSO cannot be applied to the allocation problem directly in its known form as it is not inherently discrete.

Hafiz and Abdennour [43] explained that the Euclidian-distance-based learning concept that is used in PSO while updating the values of the particles' positions and velocities is the essential cause behind this restriction. If the Particle Swarm Optimization technique or any of its alternatives are to be used to solve the dynamic task allocation problem, it is radical to develop and apply a different learning technique than the Euclidean distance-based learning concept. This area can be considered as an enormous untapped prospect.

In order to be able to use the already accumulated researchers' efforts, which are previously developed within the continuous domain of the Particle Swarm Optimization, a new learning concept must be introduced along with the fundamental framework of the Particle Swarm Optimization. The social learning concept by sharing the particles' experience is the core process in the Particle Swarm Optimization technique.

The basic idea of the Particle Swarm Optimization technique remains stable and firm from the very beginning. However, as specified by Hafiz and Abdennour [43], a lot of refinements and invariants have been introduced to Particle Swarm Optimization through the last decade. The focus of most of these improvements was on the experience-sharing concept such as the learning exemplar, the neighborhood topology, or the parameter control. At any given iteration, the basic

update equations for the particles in the PSO can be given using equation (3-31) and equation (3-32).

4.4.2.1 Challenge

As can be shown from equation (3-31) and equation (3-32), the distance is the base for the learning concept in the Particle Swarm Optimization. The cognition experience and the social experience are both used to determine the next step for the particle. The Euclidean distance between the current position of the particle itself and the learning exemplar represents its next move. The particles start moving in the direction of the learning exemplars in order to minimize the Euclidean distance.

While the particles' searching procedure, new promising regions could be found within the search space. Since the Euclidean distance cannot be used as a fitness function in the discrete problems like dynamic task assignment, thus the considerable challenge of this type of problem is to find another fitness measure than the Euclidean distance.

4.4.2.2 Solutions from literature

In order to solve this problem, various alternatives of binary versions from the fundamental Particle Swarm Optimization [43] have been proposed. The main difference among these alternatives is the methods by which the updated position is generated from the velocity as appeared in [44, 45, 46].

Kennedy and Eberhart have developed a binary version from the basic version of the PSO (1997). In this Particle Swarm Optimization's binary version, the velocity reflects the particle's probability of changing the position state instead of the speed by which the particle moves towards the target.

Also, the rank-based approach is another alternative to discretize the Particle Swarm Optimization which has been developed by Liu [47]. The word "discretize PSO" means a special version from PSO that could be used to solve the combinatorial discrete problems. In this approach, the fundamental updating rules for both the position and the velocity kept unchanged.

Thus, in order to find a new feasible solution, the velocity value is used to give a ranking to the dimension of the new state. However, since the ranking based concept doesn't necessarily give an indication to the quality of solution in the dynamic task allocation problem, it is not suitable for this type of problems.

In [48], another variant for Discrete Particle Swarm Optimization is also proposed in 2003, in which the velocity represents the probability of swapping with the learning exemplar and each particle indicates a permutation. However, in this approach, either the social or cognition learning capability is lost, since the swapping can be applied with only one learning exemplar, either the personal best or the global best. Furthermore, this approach restricts the swarm's searching ability because only one swap can be done for each particle.

One last popular approach is to rely on the set-based approach to discretize PSO, which is commonly used in data mining problems for attribute choices. A two-dimensional array represents the velocity and an attribute set indicates the position, were used in [49]. However, the technique is not generic and was specially developed for the problem of attribute choice.

In addition, another set-based variant was proposed by Neethling and Engelbrecht [50] where the new position of the particle is calculated using the personal best, the global best, and another random variable depending on three various probabilities. However, these techniques suffers from limitation in the searching ability. The mathematical operators that are used within the updating rules of the velocity in the Discrete Particle Swarm Optimization have also been redefined by Clerc [51] to be used in the travelling salesman problem.

In [52, 53], the same redefined operators are utilized in the multi-dimensional problem of Knapsack. Despite of the generality of these approaches, they result in increasing the complexity of an otherwise quite simple Particle Swarm Optimization technique.

4.4.2.3 Limitations of previous discretization techniques

The common restriction of these previous techniques is that they are not generic and each one of them is specifically designed as a special algorithm to find a solution to a special category of

problems. Hence, they lose the advantage of utilizing the already existing research efforts that have been discovered through years of improving the Particle Swarm Optimization technique's performance in the continuous domain.

As discussed by Hafiz [43], it is essential to try to maintain the simplicity of the Particle Swarm Optimization technique while implementing a discretized version from PSO to be able to find solutions to combinatorial discrete optimization problems in a simple way. However, those previous techniques add some sort of complexity to the simple structured classical PSO technique.

4.5 Optimization Techniques for Solving DTA Problem

DTA in homogeneous swarm robotics is a typical non-polynomial (NP) optimization problem. In order to find an optimum or a near optimum solution for nonlinear problems, two major categories could be followed in [54]. The first one based on the mathematical formulation, such as integer linear programming, mix-integer programming, nonlinear programming. It is well-known that exact solvers used to solve optimization problems are fast, deterministic, and can obtain exact solutions. However, it is unsuitable for large size problems, because the exponential time complexity for exact search methods on large size NP-hard problems cannot be altered. Also, the most obvious cons for this category are that it mainly demands that the problem could be formulated as mathematical equations.

In addition, the algorithm required for such a problem should be a mathematical derivative technique, such as the modified one's assignment method and Hungarian algorithm. This type of techniques is quite reliable in the sense that they have been implemented and used for decades now as marked in [19]. The most promising mathematical method to solve the MTDTA is called the Hungarian method. However, this technique is only suitable for small size problems science it is time-consuming.

The second category contains the local and global meta-heuristic optimization algorithms. Heuristic techniques such as Tabu Search, Genetic Algorithm, Particle Swarm Optimization, Ant Colony, and Simulated Annealing has been studied and developed heavily in the past few years

for solving various optimization problems. Shieh [19] declared that some of these techniques are in the form of probabilistic discrete heuristics, with local search abilities and others with global search capabilities.

However, one of the common disadvantages of these techniques as identified by Wang and Liu [54] is that it always gets trapped in a local optimum solution and it could only reach the global optimum solution in some special cases in which the swarm sizes are small. In 2001, Wang and Zhang [55] proposed a hybrid optimization algorithm based on a combination of both Genetic Algorithm and Simulated Annealing.

Also, One of the best algorithms that, has been implemented efficiently through years to find solution for lots of complex problems is genetic algorithm. However, many shortages have been detected in GA's performance in the last decade. These deficiencies have been showed up in a highly epistatic objective function's efficiency in which highly correlated optimization variables are considered.

In addition, Shieh, Kuo and Chiang [19] illustrated that because of the availability of similar structures in the population chromosomes, the mutation and crossover procedures suffers many deficiencies too. Moreover, the fitness values become larger at the evolutionary process end. Also, the GA's early convergence reduces its search ability and thereby decreases the algorithm performance, which results in a higher chance of becoming trapped in a local minimum solution.

Recently, the Particle Swarm Optimization technique has been the prime technique committed by many researchers [19] as a global searching algorithm due to its fast convergence and simplicity. It has been used to find solutions for many real time problems. By using PSO, many issues, which could be found in the old greedy algorithms can be easily eliminated, such as the restrictions in the variables' continuity and the restrictions in the acceptance of some forms of fitness functions. PSO has been verified to be one of the most promising algorithms to solve nonlinear continuous optimization problems.

Generally, PSO algorithm owns a better convergence time because it is a global parallel optimization technique. However, similar to the case of GA, the prime cons of the PSO is its early convergence that may happen when the global best and local best particles got trapped into a local optimum solution in the search space. Shieh, Kuo and Chiang [19] explained that the main reason for the problem is the tendency of the particles to move towards its near local optimum solution in the search space. Thus, most of the particles tend to gather in a very small search space, and they own a very weak global searching capability.

On the contrary, one of the most essential properties in SA is its probabilistic jumping characteristic, termed as the metropolis process. This process could be controlled by adapting the temperature parameter. It has been demonstrated by Shieh, Kuo and Chiang [19] that the SA algorithm could easily find the global minimum solution given that a proper parameters' selection is provided. Thus, a hybrid PSO/SA algorithm is developed in this thesis. The prominent characteristic of this innovative hybrid technique, which combines both DPSO and SA is the ability to find higher quality solution within a small and stable convergence time.

Thus, this section is dedicated to provide a background on both PSO and SA, which are both going to be used to build the proposed novel hybrid approach. So, in the next two subsections, the definition, flowchart, pseudo code, advantages and disadvantages for each one of those two techniques are addressed.

4.5.1 Particle Swarm Optimization technique

First, the Particle Swarm Optimization (PSO) technique which is inspired by the birds' swarming or fish's schooling patterns was developed by Russell Eberhart and James Kennedy in 1995 [56]. At the very beginning, computer software simulations of birds swarming around food sources were implemented by those scientists, then later they determined how could this algorithm been utilized on optimization problems.

Derivative from social insect's behavior, the swarm intelligence approach presents some praiseworthy characteristics, such as self-organizing capacity and flexible behavior to environmental changes. Swarm intelligence are now widely used for solving various applications

due to good global and local search capability, e.g. for traveling salesman problems with hotel selection, for staff scheduling in airport security service, for fast rescheduling of multiple workflows, for the optimal winner determination problem, and for vehicle routing problems.

PSO is really a very straightforward technique as specified by [56] despite how much it might appear complex. As can be seen from figure 4.2, a set of variables' values are altered to become closer to the variable whose value is the nearest to the target's value after a hundreds of cycles. For example, if a swarm of birds flocking around a specific position in which a hidden target, like a food source could be sniffed. The nearest bird to the target tweets loudly and the remaining ones run towards its direction. When any of the other flocking birds becomes closer to the food source than the previous bird, it tweets loudly and the remaining birds swing in its direction. Once one of the birds reaches exactly to the position of the target, all the birds stop. This technique is very straightforward and thus its implementation is very easy.

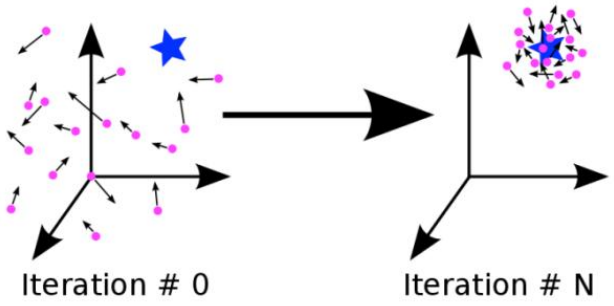


Figure 4.2: First and last iterations in PSO [57].

Three global variables need to be taken into consideration carefully during the algorithm run:

- The nearest particle to the goal, "Global best"
- The criteria in which the algorithm must stop "Stopping criteria", such as a maximum number of iterations
- The goal, "target"

As can be seen from figure 4.3, three variables could be used to describe each particle:

1. The personal best, which denotes the best position ever the particle pass through.
2. The position, which indicates the particle current solution.
3. The velocity, that represents the amount by which the position is going to be changed.

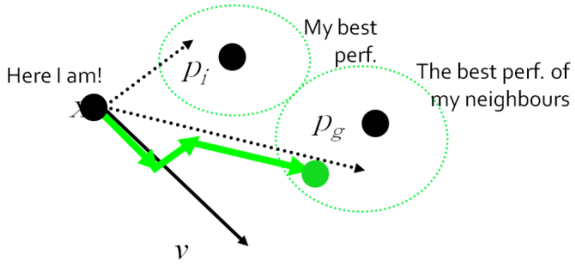


Figure 4.3: PSO position update.

The position of the particle can be in any form. It could be one dimension, two dimensions, or three dimensions. For example, the position of the birds in the previous example of the swarming birds is in the form of (X, Y, Z), where X is the x-dimension coordinate, Y is the y-dimension coordinate, and Z is the z-dimension coordinate. Each bird in the swarm keeps trying to come closer to the global best bird, which is the nearest one to the food source. The position sequence or coordinate keeps changing until it reaches to the same position of the target as explained by Hu [56].

The velocity value determines how much the particle's position is far from the target's position. The velocity becomes larger, if the particle is far away from the target. However, the more the particle comes closer to the target, the more the particle's velocity decreased. If the particle's position matches the target's position, its velocity becomes zero. By considering the previous birds' example, the birds that is far away from the food source have to fly faster to the best particle, and that's why its velocity is always larger which gives it more power to reach to the target. To sum up, the velocity represents how much is the range between the particle position and the target position and thus how fast the particle needs to travel to reach the target exactly.

Each particle's pBest value only indicates the closest the data has ever come to the target since the algorithm started.

The gBest value only changes when any particle's pBest value comes closer to the target than gBest. Through iterations of the algorithm, gBest gradually moves closer and closer to the target until one of the particles reaches the target, see figure 4.4.

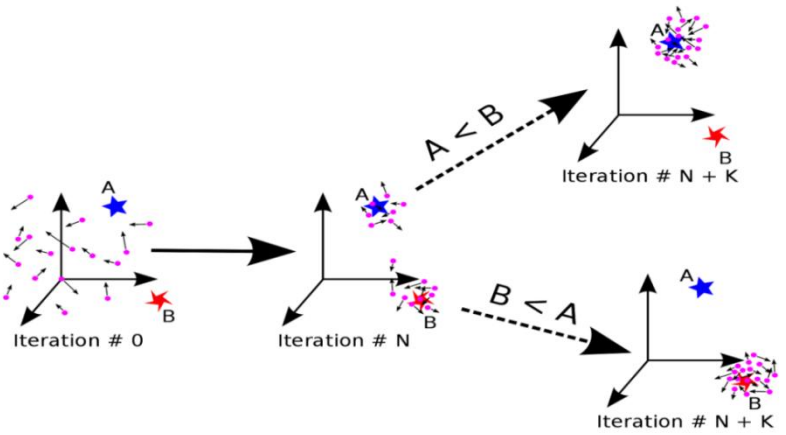
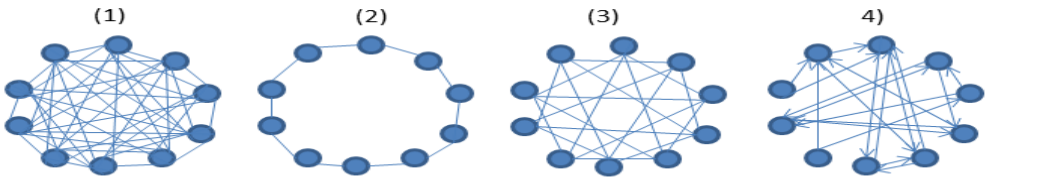


Figure 4.4: PSO global best update over different iterations [57].

A different set of topologies or neighborhoods can be used with PSO algorithms' populations as discussed in [57]. These topologies could be used to simplify and divide the whole population into simpler groups. Each one of these subsets contains a set of particles (two or more) which are pre-decided to cooperate and they often share the same part of the search space. To reach the global minima and avoid getting stuck in local minima, it is often better to use topologies. Examples of these topologies can be shown in figure 4.5.



Graphical representation of (1) fully connected, (2) ring, (3) von Neumann and (4) random topology

Figure 4.5: PSO topologies [57].

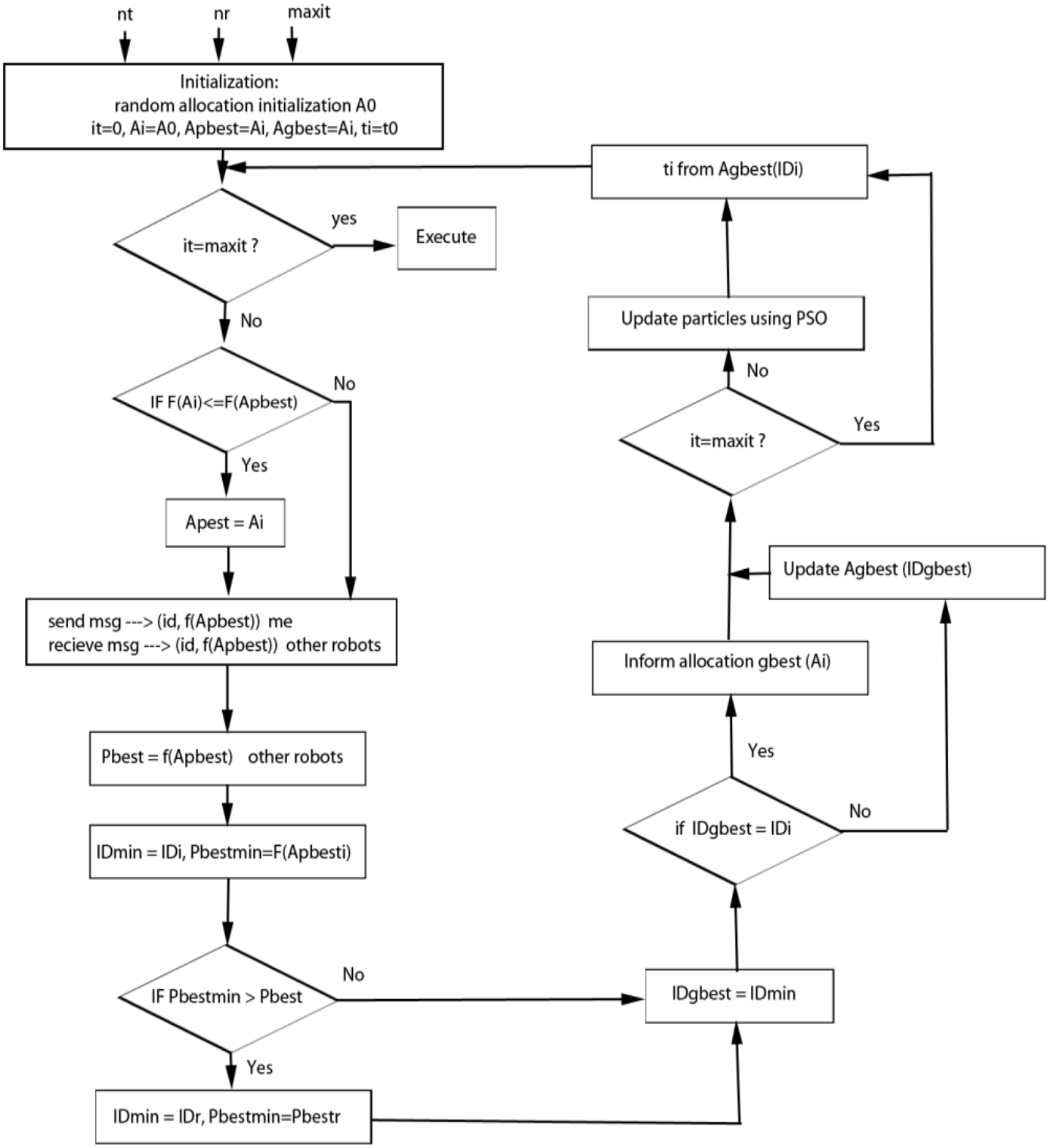


Figure 4.6: Flowchart of the Particle Swarm Optimization technique

As can be seen from figure 4.6, the PSO algorithm steps are as the follows:

- First, a random allocation is chosen for each particle in the swarm A0.
- Then, the personal best (A_{pbest}) and the global best (A_{gbest}) for each particle is initialized to be equal to its own initial position.
- Then, the algorithm starts to iterate for a specific number of iterations (maxit).
- The fitness value of each particle allocation is calculated according to the objective function, which is the minimum distance from each robot to its corresponding task (Euclidean distance).
- Then, comparison is made between the current calculated fitness value for each particle and the previous calculated best fitness. If the new allocation fitness is smaller, it is chosen as the particle personal best. Then, the smallest fitness among all the particles is chosen as the global best.
- After that, the particle, which has the global best allocation, sends its allocation to the other particles.
- Furthermore, each particle calculates the velocity, which it uses to move towards its new allocation according to the velocity's update equation of the PSO as specified in equation (3-31).

$$v = w \times v \ + c1 \ \times rand \ \times (pbest - x) + \ c2 \ \times rand \ \times (gbest - x)$$

Equation (4-31)

Where, constants c1 and c2 is chosen to be 2, and rand is a random number from 0 to 1.

- Then, the new position or allocation for each particle is updated using equation (3-32).

$$x = x + v$$

Equation (4-32)

- Finally, if the number of iterations reaches the maximum iteration, the algorithm stops. Then, each robot begins its task execution according to the global best allocation solution found in A*. If not, steps from 4 to 9 are repeated.

After designing the flowchart, the pseudo code for the PSO is developed as follows:

a) Main code

```
1: Initialization (Nt, Nr);
2: While it ≤ maxit {
3: Personal best update (A, Apbest, Agbest);
4: Global best update (Pbest);
5: Global best allocation diffusion (idgbest);
6: Calculate particle's velocity;
7: Update particle's position;
8: }
9: Execute;
```

b) Initialization

```
Input: Nt, Nr
Output: A, Apbest, Agbest
1: For i = 1 to Nr {
2:                   A[i] = random generator (1, …, Nt);
3: }
4: Apbest = A, Agbest = A;
```

c) Personal best update

```
Input: A, Apbest, Agbest
Output: Pbest
1: Calculate fitness value f (A);
2: if f (A) ≤ f (APbest) {
3:                   Apbest = A;
4: }
5: msg ← (id, f(APbest)); send fitness value to all other robots
6: For every other robot {
```

```
7: Exchange the fitness value;
8:}
9: For i = 1 → Nr {
10:                 Pbest[i] = f(APbest) for robot id = i;
11: }
```

d) Global best update

```
Input: Pbest
Output: id_gbest
1: id_gbest = 1;
2: Pbest_min = Pbest [1];
3: for i = 2 → Nr {
4:                  if Pbest_min > Pbest[i] {
5:    id_gbest = i;
6:    Pbest_min = Pbest[i];
7:  }
8: }
```

e) Global best allocation diffusion

```
Input: id_gbest
Output: Agbest
1: if id_gbest is my id {
2: Send (Agbest);
3: }
4: Else {
5:                  Receive (Agbest);
6: }
```

The collective behaviors of social insects, such as the honeybee's dance, the wasp's nest building, the construction of the termite mound, or the trail following of ants, were considered for a long time strange and mysterious aspects of biology. In the most recent years, it has been

deduced that particles do not need precise information or a complex representation of this information to generate such complex attitudes. In the search space, the particles do not have exact knowledge about the whole population. The particles reach to their destination without the need to be coordinated by a single leader. However, the information about the swarm and the environment is shared among all the particles. All the particles need to cooperate with each other in order to accomplish the whole task and no particle could finish its task on its own as determined by Navarro and Matía [7].

Social insects are able to exchange information, and for instance, communicate the location of a food source, a favorable foraging zone or the presence of danger to their mates. This interaction between the individuals is based on the concept of locality, where there is no knowledge about the overall situation. The implicit communication through changes made in the environment is called stigmergy. Insects modify their behaviors because of the previous changes made by their mates in the environment. This can be seen in the nest construction of termites, where the changes in the behaviors of the workers are determined by the structure of the nest as stated by Navarro and Matía [7].

Navarro and Matía [7] have mentioned that the particles communicate among each other and with the environment in order to form a social arrangement. The information, which is resulted from this cooperation, is distributed along the population and thus a complex tasks that cannot be solve with a single particle can be accomplished in this way. These global attitudes are described as self-organizing attitudes. The complex attitudes which results from the communications among very simple single particles can be explained by the self-organization theories, which are mainly, deduced from physics and chemistry domains. Positive feedback, negative feedback, randomness, and multiple interactions are the four main principles, which produce the self-organization behavior.

Some properties seen in social insects as desirable in multi-robotic systems are:

- Flexibility, the swarm must be able to create different solutions for different tasks, and be able to change each robot role depending on the needs of the moment.

- Scalability, the robot swarm should be able to work in different group sizes, from few individuals to thousands of them.
- Robustness, the robot swarm must be able to work even if some of the individuals fail, or there are disturbances in the environment.

4.5.2 Simulated Annealing technique

Simulated Annealing (SA) is a probabilistic technique for approximating the global optimum of a given function. Specifically, it is a meta-heuristic to approximate global optimization in a large search space. SA is one of the most proper techniques if we deal with a discrete search space, such as the problem of choosing the order of cities to be visited by tours in order to minimize the overall cost. Geltman [58] specified that if an approximate global minima is require more than an accurate local minima and the time to find the solution is a limited amount of time, SA is preferred than other existing optimization techniques like ant-colony.

The name of the technique is inspired from the annealing process of the metal, which involves heating and controlled cooling of a metal to decrease their defects and increase its crystals' size. Thermodynamic free energy controls these two properties. To sum up, both the temperature and the thermodynamic free energy are affected by Heating and cooling the material.

Geltman [58] illustrated that accepting bad solutions is an essential characteristic of meta-heuristics as it helps to explore new regions in the search space in order to find the most optimum solution. The slow cooling of the metal can be translated as a slow reduction in the probability of how much could we accept bad solutions while exploring the search space. The energy function which need to be minimized and the state of some physical systems are similar to the internal energy of the system in that state. The target is to translate the system from an initial state with a high energy to another state, which has less amount of internal energy.

The SA algorithm could decide to leave its current solution and move to a neighboring solution at each iteration, if and only if the neighboring solution drives the system for more stability state in which the internal energy of the system is decreased. The stopping criteria could be either the

arrival to an acceptable state for the application or the arrival to a maximum number of iterations. Thus, the stopping criteria often depend on the application itself, see figure 4.7.

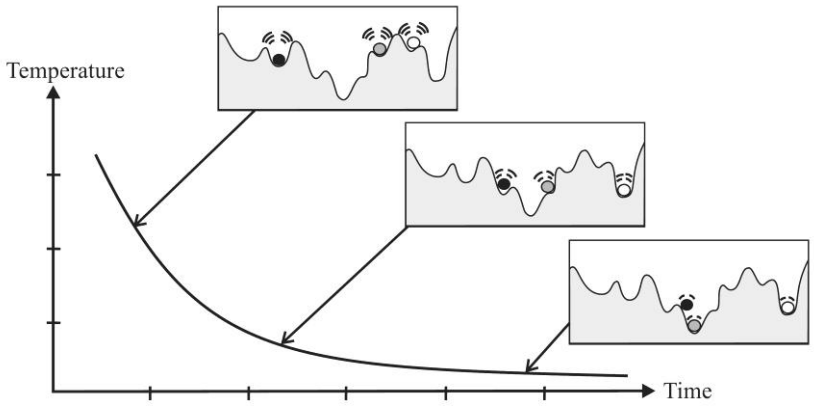


Figure 4.7: Simulated Annealing technique demonstration [59].

Geltman [58] discussed that a predetermined well-defined way must be used in order to drive the system from the current state to the neighboring state. A permutation of a visited city is a very common method for states' changes, which is always used in the traveling salesman problem. Swapping any two successive cities, reverse the order of any number of successive cities, or choose a specific city and insert it between two successive cities are examples of common methods that could be used to determine a new neighboring state. The predetermined common way, which is decided to be used in choosing the new neighboring state, is referred to as a "move" and each move resulted in a distinctive new neighbor. It is better for the moves to be chosen to change as minimum part as it could in order to preserve the system stability and change only the worse part. For example, the connections between the cities are the states of the systems in the traveling salesman problem.

One of the most common issues in the optimization problems is that the algorithm could get stuck in a local minimum and never reaches to the global optimum solution. Therefore, sometime while searching for new solution, a worse solution could be accepted to explore new search

spaces as the final best solution is found after a tour of different neighbors in which some of them could be bad solutions. Some old heuristics, the algorithm moves only to a better solution. However, this way the algorithm could reach a state some times in which no better solutions are found. This does not mean that no better solution could be found at all. However, it means that the system gets stuck in a local minimum. If a worse solution is accepted sometimes in order to avoid local minimum and the run time of the algorithm is infinite, most probable the global optimum would be found.

However, by comparing the SA algorithm with the other technique, why to choose SA:

Genetic algorithm, hill climbing, ant-colony, clustering auction, and more are very common optimization algorithms. However, most of them get stuck in a local optimum and never reaches the global optimum. What Distinguish SA is that it has a technique by which SA sometimes accept worse solutions in order to avoid local optimum and explore new areas from the search space.

You can visualize this by imagining a 2D graph like the one shown in figure 4.8. X-coordinate represents the system states "solutions", and the Y-coordinate represents how good that solution is.

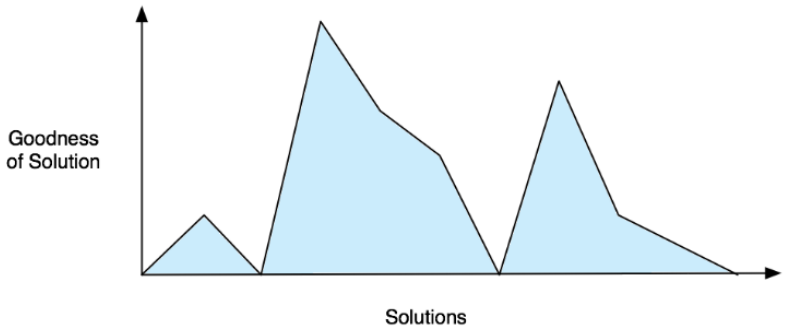


Figure 4.8: Simulated Annealing cost function [60].

At the beginning, optimization algorithms produce an initial state randomly. Then, the algorithm searches for a neighbor, which owns a better solution. If a better state is found, it moves to this state. However, the algorithm stops if no more better states can be found around.

This actually makes sense. However, Busetti [60] showed that it could lead to a local optimum solution rather than the global optimal one. As shown in figure 4.9, the left yellow star is the optimal solution at the beginning. Then, the green right star, which is a local minimum solution, is found. Once a simple algorithm finds this solution, it will never move to any other solution because all its neighbors are larger solutions.

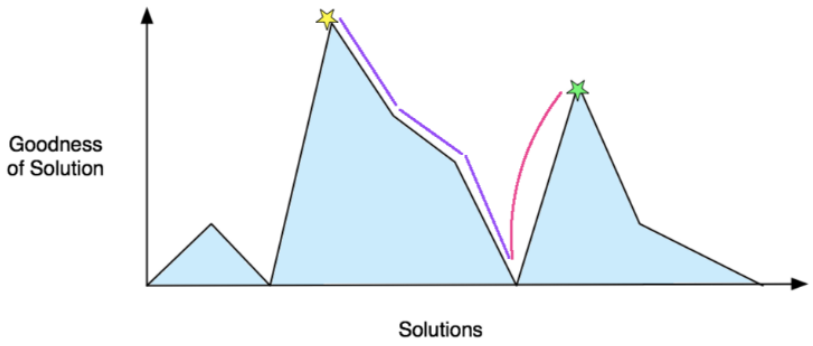


Figure 4.9: Local minimum vs. global minimum [60].

SA sometimes moves to a new random solution, which is worse with a probability function in order to avoid getting stuck into a sub-optimal. In addition, it tries not to get much away from its current state because the target may be nearby. Regardless of the starting point of the algorithm, this technique capable of tracking the best solution and attain the optimal state at the end.

Here is a really high-level overview, see figure 4.10. It skips some very important details, which will be gotten to in a moment.

The algorithm steps:

- At the beginning, a new state must be produced randomly.
- By choosing a specific fitness function for the problem, find the fitness value for this first state.
- Produce another neighboring state randomly.
- Find its fitness value too.
- Then, a comparison has to be made:
- The algorithm leaves the current state to the new state, if $C_{new} < C_{old}$.
- The algorithm may decide to move and may not, if $C_{new} > C_{old}$.
- Continue executing the previous steps 3-5 until finding a proper state with a reasonable fitness value or until the number of iterations reach its maximum value.

Let's break it down:

- **At the beginning, a new state must be produced randomly.**

It must be chosen randomly; it does not have to be an optimal choice or ever a quite good solution. Any one of the available states could be chosen.

- **Find the fitness value for this first state.**

Depending on your problem, this cost function is calculating the total number of meters the robots travel. Calculating the cost of each solution is often the most expensive part of the algorithm, so it pays to keep it simple.

- **Produce another neighboring state randomly.**

The new generated state must be distinct from the old one in only one change, for example, two elements from the old state could be swapped or a single element could be removed from its place and inserted between two successive elements. "Neighboring" means there is only one thing that differs between the old solution and the new solution.

- **Find its fitness value too.**

Use the previous fitness function, which has been used to calculate the fitness value for the first state. It should be used for the rest of the problem.

- **If Cnew < Cold: take a step to the new state**

Move to the new state if its fitness value is better than the fitness value of the previous state. In this case, the system becomes more stable and therefore happier as it becomes closer to the bet state. Preserve this state as the main state for the next cycle.

- **If Cnew > Cold: maybe move to the new solution**

The system always avoids moving to the new state, which is worse than the previous state. In this case, the system gets stuck in a local minimum and would never reaches to the global optimal solution. In order to escape from getting stuck in a sub-optimal solution, the algorithm could sometimes choose the worse state. A probability of whether to accept the new state or not must be calculated which is referred to as the (acceptance probability).

After that, a comparison between this probability and a generated random number must be done to decide whether to accept it or not.

In order to calculate the acceptance probability value, the temperature current value, the current fitness, the new fitness and a random number between zero and one are needed. The random number is a probability to choose whether to move to the new state or wait in the old state.

For example:

- 0.0: 100% stay in the old solution because it is better than the new one.
- 1.0: 100% move the new solution because it is better than the old one.
- 0.5: move with probability 50%. Therefore, it may move and may not.

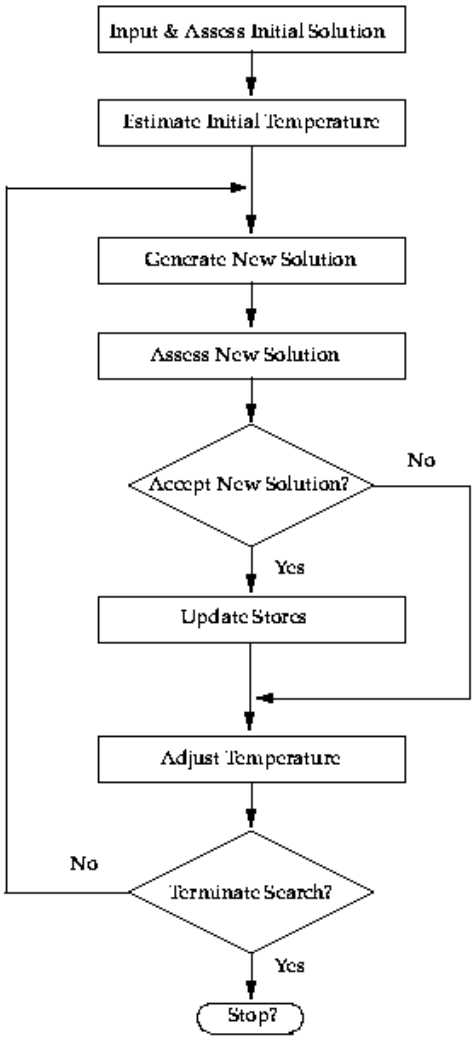


Figure 4.10: Simulated Annealing flowchart [60].

After that, a comparison between the randomly generated number and the acceptance probability has to be done and the algorithm moves to the new state if the random number is smaller.

As defined by equation (3-33), this equation is typically used for the acceptance probability is:

$$a = \mathbf{e}^{(\mathbf{Cold}-\mathbf{Cnew})/\mathbf{T}}$$

Equation (4-33)

In the previous equation, the acceptance probability is referred to as a, the variation between the current fitness and the new fitness is referred to as (Cold−Cnew), the temperature is referred to as T, and e is 2.71828.

The previous equation is a part of SA, which was inspired by the metal annealing process. This equation characterize the process of heating the metal to increase the temperature of the internal energy and increase the speed of the internal particles, and then cooling it down slowly. After cooling down the temperature, the particles' speed start decreasing and the metal internal energy start also to decrease. This process is used by the Computer engineering researchers, by which the algorithm moves from a higher fitness state to another lower fitness state.

Hence, it could be concluded from the SA's equation that the acceptance probability becomes:

A. Lower when the current state is better than the new state.
B. Lower when the temperature decreases.
C. Greater than one if the current state has a worse fitness than the new state. In this case, $\alpha=1$ is used to avoid having a probability greater than 100%.

Thus, as reported by Busetti [60] the algorithm tends to accept worse solutions at the beginning when the temperature is high. In addition, if two bad moves exist, the algorithm tends to choose the one with the better fitness.

Chapter 5

Minimal Time Dynamic Task Allocation Through Hybrid DPSO/SA Algorithm

In this thesis, we propose a hybrid DPSO/SA algorithm that fuses the concept of SA algorithm with the discrete DPSO algorithm to overcome the imperfections of both SA and DPSO. SA is used to promote the DPSO's local searching capability. The hybrid algorithm not only possesses the pros of DPSO and SA but also omits the cons of both of them such as the local minima problem in DPSO and the slow convergence time of SA technique.

Therefore, in the next four sections, we start with the design and implementation of the Hungarian algorithm as an exact solver for the MTDTA problem. Then, the DPSO technique is developed followed by SA technique and we end up with the implementation of the proposed hybrid DPSO/SA approach.

5.1 Hungarian Algorithm

One of the best optimization techniques that could make advantage of the special structure of the MTDTA problem and provide an exact solution to the problem is the Hungarian method. This exact solver is used as a model to compare between the hybrid PSO/SA algorithm with both classical DPSO and SA techniques.

Basically, the exact solution resulted from applying this technique is used to normalize the outcomes from the other techniques and then we classify these outcomes into three categories, small swarm size, medium swarm size, and large swarm size to provide more readable and clear results.

This combinatorial optimization algorithm could find solution to the MTDTA problem in a polynomial time. However, the time complexity for the large size problem is significantly large as mentioned in [61]. This method is basically based on previous works done be the Hungarian

mathematicians: Dénes Kőnig and Jenő Egerváry [62]. The algorithm was implemented and published by Kuhn [63] in 1955, and Kuhn was the one who called it the "Hungarian algorithm".

In 1957, the Hungarian algorithm has been revised by Munkres [64], and he noticed that the algorithm is substantially sufficient. After that, the algorithm became well known as "Munkers assignment algorithm". The Hungarian algorithm basically had time complexity of O(n^4) in the big O notation. Figure 5.1 shows various time complexity functions in the big O notation.

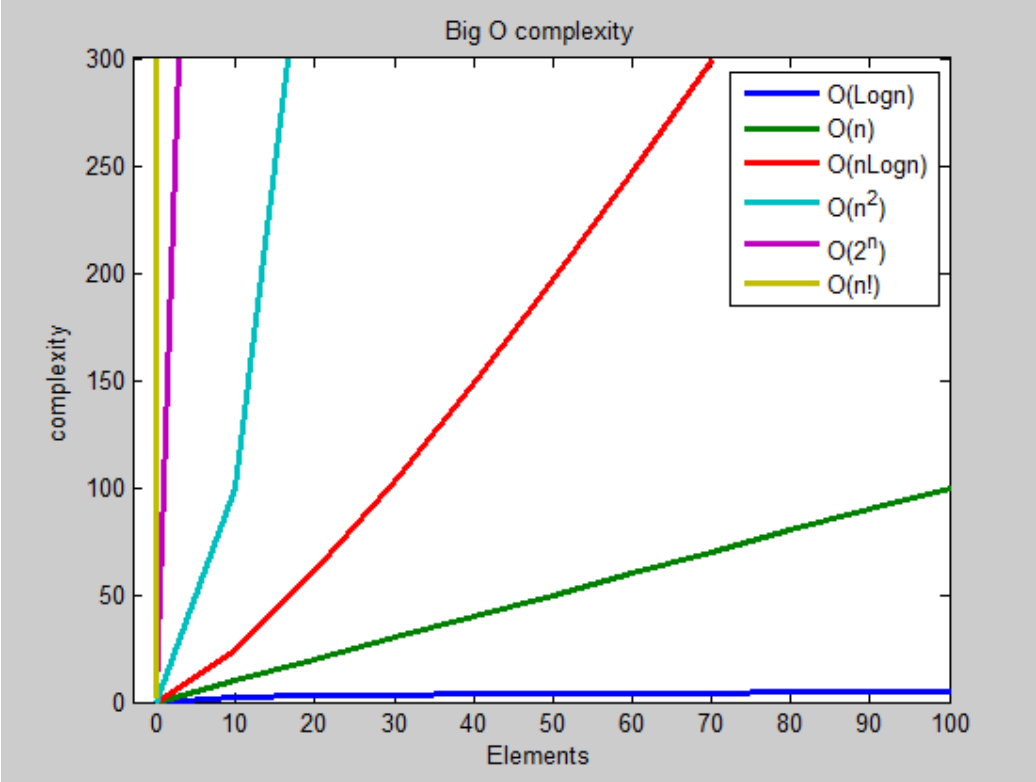


Figure 5.1: Complexity functions

The big O notation [65] is a well-known method used for evaluating the run time of algorithms. It can be considered as a way to differentiate between the performance of various techniques in solving a specific problem. In the big O notation, the algorithms' runtime is addressed in terms of

how quickly it grows relative to the input, as the input gets arbitrarily large. Let's illustrate this more:

1. It is extremely hard to determine the accurate runtime of a specific algorithm since there are a set of external factors that may have a direct impact on the time the algorithm takes to run a given function such as the other programs run at the same times on the computer, and the processor's speed. Thus, the big O notation evaluates how quickly the algorithm's runtime grows instead of the exact runtime.
2. ALSO, in the big O notation we calculate the runtime growth using the size of the input instead of an exact number.
3. For big O analysis, we concern more about things that grows fastest as the input grows, because everything else is quickly eclipsed as the input gets very large.

For example:

Assume that we have a swarm system consists of three robots and three tasks as shown in table 5.1. The robots termed as R1, R2, and R3, and the tasks termed as T1, T2, and T3. Each one of the robots should assign itself to one of the available tasks. However, the robots require paying differently for the various tasks because both the robots and the tasks exist at different locations. The goal is to find the best allocation among the robots and tasks such that the cost should be minimum.

The problem can be declared as a matrix of the costs (distances) between the robots and tasks. For example, as can be seen from table 5.1, cell (1,1) represents the cost of assigning the robot number 1 to the task number 1, cell (1,2) represents the distance between the robot number 1 and the task number 2 and so on.

Table 5.1. Example of system with 3 robots and 3 tasks.

Tasks / robots	T1	T2	T3
R1	2 m	3 m	3 m
R2	3 m	2 m	3 m
R3	3 m	3 m	2 m

The minimum solution of 6m is obtained by applying the Hungarian algorithm on the previous example. This solution is obtained by allocating robot 1 to task 1, robot 2 to task 2, and robot 3 to task 3.

5.1.1 Cost matrix representation

Problem definition: Given a collection of tasks and a set of robots, M * N matrix including the cost of allocating each robot to each task is set up. Thus, the goal is to find out the minimum cost for the allocation problem.

At the beginning, the matrix, which represents the problem, is formulated as follows:

$$\begin{bmatrix} a1 & a2 & a3 & a4 \\ b1 & b2 & b3 & b4 \\ c1 & c2 & c3 & c4 \\ d1 & d2 & d3 & d4 \end{bmatrix}$$

where a, b, c and d represent the robots which need to be allocated to task 1, task 2, task 3 and task 4. The costs of allocating robot a to task 1, task 2, task 3 and task 4 are termed as a1, a2, a3, and a4, respectively. Same definition is applied for the other rows. As specified in [66], by considering the problem formulation and the constraints, each task should be allocated to only one robot and each robot should be assigned to just one task.

Steps for solving the assignment problems using the Hungarian algorithm can be outlined as follows:

Step 1:

First, a set of operations should be implemented on the matrix as identified in [67]. We have to start by choosing the smallest element in each row and subtract all the row's elements from that number. Now, at least there is one zero in each row and may be more if the smallest number in the row is replicated in more than one element. Now, we should attempt to allocate tasks to

robots such that each robot is performing just one task. The matrix below demonstrate this step in which the allocated tasks are represented by the zeroes.

$$\begin{bmatrix} 0 & a2 & a3 & a4 \\ b1 & b2 & b3 & 0 \\ c1 & 0 & c3 & c4 \\ d1 & d2 & 0 & d4 \end{bmatrix}$$

Step 2

In some cases, we could find out that no possible assignment could be applied to the matrix after the previous step. Below is an example of such a case.

$$\begin{bmatrix} 0 & a2 & a3 & a4 \\ b1 & b2 & b3 & 0 \\ 0 & c2 & c3 & c4 \\ d1 & 0 & d3 & d4 \end{bmatrix}$$

At this stage and as there is no suitable assignment could be performed, because by looking at the previous matrix it could be easily figured out that the first task could be assigned as an optimal task for both the first and the third robots and this can't be happened because it contradicts the problem constraint. Also, task 3 couldn't be efficiently performed by any of the active working robots. Therefore, as mentioned by Ghadle and Muley [68], this issue can be solved by repeating the previous step but considering the columns instead of rows. So, the minimum element at each column must be determined and all elements of this column must be subtracted from this element. Then, the robots-tasks assignment should be attempted.

Now, if an assignment is possible, the robots should be assigned to tasks and the algorithm terminates. However, if still no possible assignment exists, then we have to proceed to the next step as illustrated in [66].

Step 3

At this step, a minimum number of rows and columns, which could cover all zeros in the matrix should be marked. The steps to attain this can be summarized as follows:

1. First, all possible allocations should be considered as follows:

    - The zero in the first row is allocated. However, the zero in the third row should be eliminated because each task must be assigned to only one robot. Another alternative could be to assign the zero in the third row and eliminate the zero in the first one instead.
    - The zero in the second row is allocated.
    - No tasks are allocated in the third row because the only zero is eliminated.
    - The fourth row has two possible allocations. So, any one of them could be allocated and the other zero should be eliminated.

$$\begin{bmatrix} 0 & a2 & a3 & a4 \\ b1 & b2 & b3 & 0 \\ 0 & c2 & c3 & c4 \\ d1 & 0 & 0 & d4 \end{bmatrix}$$

2. Second, the marking step:

    - Mark all rows with no allocations; in this case, the third row is marked.
    - All columns that own zeros in the marked row should be marked too. In this case, column one is marked.
    - All rows, which own allocations in the latter marked columns, should be marked too. In this case, the first row is marked.
    - Repeat the previous steps for all rows that are not allocated.

x				
0	a2	a3	a4	x
b1	b2	b3	0	
0	c2	c3	c4	x
d1	0	0	d4	

3. The drawing step: At this step, all unmarked rows and marked columns should be covered with lines.

x				
0	a2	a3	a4	x
b1	b2	b3	0	
0	c2	c3	c4	x
d1	0	0	d4	

There are a lot of other techniques that could be used to determine the minimum number of rows and columns that could cover all zeros in the matrix. This method is the simplest one of them.

Step 4

Now, determine the smallest number from the uncovered area. Then, subtract this number from all uncovered elements and add it to all elements, which are covered twice.

Finally, steps 3 and 4 should be repeated until a possible allocation could be found. To obtain this condition, the minimum number of rows and columns, which covers all zeros should be equal to the number of robots (rows).

Note that, as explained in [66], if the number of robots is larger than the number of tasks, extra number of columns should be added and filled with dummy values. These dummy values could be the maximum number in the whole matrix or the infinity value.

Example

In this example, we have four tasks need to be assigned to four robots, only one task could be assigned for each robot and only one robot could perform every task, see table 5.2. The following matrix represents the cost of allocating each task for each robot and the target is to minimize the cost needed for assigning the entire tasks. This assignment problem is solved using the Hungarian algorithm as explained in [68].

Table 5.2: Example of system with 4 robots and 4 tasks

	T1	T2	T3	T4
R1	82	83	69	92
R2	77	37	49	92
R3	11	69	5	86
R4	8	9	98	23

Step 1, the minimum element in each row of the matrix must be selected and written at the right hand side of the matrix. Then, this element should be subtracted from all the elements in the row. This process should provide at least one zero at each row. For example, the smallest number in first row is 69. This number is written in the right hand side. Then, we have to subtract 69 from the first rows' elements, which give us zero at the third column. Same process should be applied for the remaining rows.

	T1	T2	T3	T4	
R1	13	14	0	23	69
R2	40	0	12	55	37
R3	6	64	0	81	5
R4	0	1	90	15	8

After that, we have to check, if an assignment could be achieved at this step. Unfortunately, by looking at the previous table, task 4 could not be assigned to any of the available active robots. So, we have to proceed to the next step.

Step 2, now the minimum element in each column must be selected and written below the matrix. Then, this element should be subtracted from all elements in the column in order to make sure that we own at least one zero in each column. In this example, the minimum element in the fourth column is 15. Thus, it is subtracted from each element in this column. Now, we are sure that we have at least one zero in each row and one zero at each column. However, we have to

check if there are suitable assignments for the entire tasks. This could be done by calculating the minimum number of lines that could be drawn to cover all the zeros in the matrix.

	T1	T2	T3	T4
R1	13	14	0	8
R2	40	0	12	40
R3	6	64	0	66
R4	0	1	90	0

15

Step 3, in order to calculate the minimum number of lines needed to cover all zeros in the matrix, all rows that don't have a suitable allocation must be crossed out. So, since task 3 cannot be assigned to the third robots as it has been already assigned to robot 1, then the third row is crossed out. Also, as column three has a zero at third row, it should be crossed out too. Finally, all the rows, which own a zero at the third column, should be crossed out too. Thus, row one is crossed out.

Now, the entire crossed columns and the uncrossed rows are covered. Since the minimum number of line, which covers all zeroes in the matrix smaller than the number of rows, we have to proceed to the next step.

	T1	T2	T3	T4
R1	13	14	0	8
R2	40	0	12	40
R3	6	64	0	66
R4	0	1	90	0

Step4: now, the minimum number from the uncovered area should be selected. After that, it should be subtracted from the uncovered area, and added to the double covered area. Then, we have to check if the entire tasks could be assigned to the robots or not. If yes, stop the algorithm

and calculate the overall assignment cost. If not, repeat step 3 and 4 again until a suitable assignment could be found for all the robots in the swarm.

	T1	T2	T3	T4
R1	7	8	0	2
R2	40	0	18	40
R3	0	58	0	60
R4	0	1	96	0

The covered zeros in the previous matrix are the minimum allocations.

	T1	T2	T3	T4
R1	82	83	69	92
R2	77	37	49	92
R3	11	69	5	86
R4	8	9	98	23

Finally, since task 3 is assigned to robot 1, task 2 is assigned to robot 2, task 1 is assigned to robot 3, and task 4 is assigned to robot 4, then the total cost for this optimal allocation is (69+37+11+23=140).

In this thesis, the optimal solution to the dynamic task allocation problem is found within milliseconds using the Hungarian algorithm in both small and medium swarm sizes (from 7 robots till 90 robots). The Hungarian code is implemented using MATLAB. The dynamic task allocation Problem is basically a special category of the transportation problem, which is a special case of another case called the transportation problem, which is a special case of the minimum cost flow problem, which in turn is a special case of the linear integer programming. Although the optimal solution to any of these problems could be found using the

general simplex method, it is better to use certain algorithms for each category, which designed specifically to take the advantage of its special structure.

Many techniques are developed to solve the dynamic task allocation problem. The Hungarian algorithm is one of these algorithms that have the ability to find an optimal solution to the linear assignment problem within time complexity bounded by a polynomial expression of the number of robots (N). In addition, auction algorithm and simplex algorithm are other method that could be used in this kind of problems. However, when a very large swarm size is considered, a heuristics optimization technique with randomization like Simulated Annealing, Genetic Algorithm and Particle Swarm Optimization could be a better option to solve the problem and find a near optimal solution within a reasonable time.

Singh [69] states that many decision-making problems where the target is to allocate a collection of resources to a finite number of tasks to gain a maximum profit or to achieve a minimum cost are referred to as the assignment or dynamic task allocation problem.

To sum up, the pseudo code for Hungarian Algorithm could be outlined as follows:

```
Start
    1: Subtract each row's elements from the smallest number in the row;
    2: Subtract each column's elements from the smallest number in the column;
Check assignment possibility by drawing lines to cover all zeros in the
matrix
    3: If (the number of lines = the number of robots)
    4:       {
    5:       Perform the allocation;
    6:       }
    7: If (drawn lines < number of robots)
    8:       {
    9:       Subtract the smallest number of the uncovered area from all the
             uncovered elements and add it to elements that are covered twice;
    10:      }
```

```
    11:      Repeat the last two steps until a proper allocation is obtained;
End
```

Another numerical example:

The following matrix contains the cost values of assigning three robots to three tasks as shown in table 5.3, and it is required to find the optimal allocation among robots and tasks in order to minimize the overall cost.

Table 5.3: Matrix of 3 robots and 3 tasks

	T1	T2	T3
R1	250	400	350
R2	400	600	350
R3	200	400	250

Step 1: subtract 250 from the first rows elements, 350 from the second row's elements, and 200 from the third row's elements.

	T1	T2	T3
R1	0	150	100
R2	50	250	0
R3	0	200	50

Step 2: subtract 0 from the first column's elements, 150 from the second column's elements, and 0 from the third column's elements.

	T1	T2	T3
R1	0	0	100
R2	50	100	0
R3	0	50	50

Step 3: draw minimum number of lines to cover all zeros in the matrix. Since the number of lines equal three (the number of robots), then an optimal allocation is attained.

	T1	T2	T3
R1	0	0	100
R2	50	100	0
R3	0	50	50

Step 4: if there is only one zero in a row, assign it immediately to the robot. However, if there are more than one zero in the row, postponed the allocation of this robot till the end.

	T1	T2	T3
R1	0	0	100
R2	50	100	0
R3	0	50	50

After assigning all the robots to their corresponding tasks, start calculating the overall cost for the selected assignments.

	T1	T2	T3
R1	250	400	350
R2	400	600	350
R3	200	400	250

The overall cost in this case equals (400+350+200=950).

5.2 PSO

Particle Swarm Optimization (PSO) is a parallel population-based optimization technique proposed by Kennedy and Ebehart [70], that was stimulated by the collective attitude of organisms such as fishes' schooling and birds' flocking. Particle Swarm Optimization technique can be used to solve difficult large-size optimization problems.

Shieh, Kuo and Chiang [19] clarified that the PSO search process imports the swarming idea by which a specific fitness function could be optimized using a collection of particles. Through the communications and the co-operations among individuals, the particles tend to move in the direction of the best optimal solution. This algorithm tries to simulate the group behavior of birds and animals.

In addition, Shieh, Kuo and Chiang [19] reported that the decision-making in PSO is similar to the decision-making process done by human. The process of human learning and cultural transition based on two main sources of learning: learning from the human own experience itself and learning from other peoples' experience. This learning process is the basis for the human decision making process.

Both the fast convergence and the simple parallel structure of the PSO can be considered as its preferable characteristics as illustrated by Shieh, Kuo and Chiang [19]. The main procedures for the PSO algorithm are as follows: at the beginning, the swarm particles are generated randomly using a uniform distribution and these individuals represents a possible solution within the search space. Then, by using both the personal best for each particle in the swarm and the global best among all the particles, each individual could decide the direction and the speed by which it should move to a new position. After a finite number of iterations, the particle swarm would take a fancy towards the minimum solution.

Wang and Liu [54] explained that the main difference between the Particle Swarm Optimization and the other meta-heuristics techniques is that Particle Swarm Optimization technique utilized the physical location and movements of the particles in the swarm to update itself. It also has a

well-balanced and adaptable technique to promote and adapt via collaborating with the global best and personal best particle in the swarm, while other techniques like genetic algorithm utilizes genetic operators such as mutation and crossover. Another advantage of PSO is its simplicity in coding and its consistency in performance.

Wang and Liu [54] also notified that the Particle Swarm Optimization technique is somehow comparable to the evolutionary algorithms, and it is initialized by a random population of different particles. Every particle in the swarm moves in the multi-dimensional search space with a particular direction and speed named velocity. Based on the cognition experience of the particle itself and the social experience of the whole swarm, the velocity is iteratively adapted. The new position for each particle is also adjusted every iteration using equation (3-31) and equation (3-32).

As presented in [54], in Equation (3-31), the first portion indicates the old velocity's inertia. The second portion represents the cognition experience, which indicates the experience gained by the particle itself. While, the last portion denotes the particle's social experience, which indicates the experience gained from the communication and interaction with the other particles in the swarm.

You could refer to both the main implementation steps for the Particle Swarm Optimization technique and the pseudo code in (section 3.5.1).

Fitness function

The evaluation of the performance of the candidate solutions within the swarm is done using the fitness function. Function f: s --> R+ generally represents the fitness (where s represents a group of candidate solutions and R+ represents the positive real values set) as stated in [54]. In the dynamic task allocation problem, the objective of the fitness function is particularly the distance travelled by the robots in the swarm.

5.2.1 DPSO

Hafiz and Abdennour [43] announced that due to the significance of the dynamic task allocation in many real-life applications plus the complexity of this problem, it has drawn the attention of

considerable research efforts in most recent years. Roughly, most of the evolutionary and natural based optimization techniques have been tried to solve the dynamic task allocation problem such as, genetic algorithm, ant colony technique, the artificial bee colony technique, etc. In addition, a huge number of local meta-heuristics optimization techniques have been applied to solve this problem like, Simulated Annealing, Tabu Search technique, Hill Clamping, etc.

Although, the PSO technique can be considered as one of the most promising techniques in the swarm robotics' field and in other distinctive applications, it actually has not received the researcher's attention in the dynamic task allocation problem as the other meta-heuristic techniques did. The reason behind this could be attributed to the Euclidian-distance based learning concept in the Particle Swarm Optimization technique's structure as proposed by Hafiz and Abdennour [43]. This problem makes the Particle Swarm Optimization, in its inherently continuous basic form, inappropriate for discrete optimization problems.

5.2.1.1 PSO probabilistic learning mechanism

In this thesis, a probability-based approach is applied as a learning concept for the Particle Swarm Optimization instead of the Euclidian-distance based learning concept. So, a general structure to discretize the Particle Swarm Optimization is attained in order to be able to implement the Particle Swarm Optimization in its combinatorial form on the problem in our hands, which is inherently discrete problem. In addition, comparisons with other local meta-heuristic approaches are presented in this thesis.

5.2.1.2 PSO discretization

The procedures needed to discretize the Particle Swarm Optimization are outlined in this section. In addition, both the particles' position and velocity updating rules are introduced with a number of illustrative examples.

5.2.1.2.1 DPSO position representation

The particles in this algorithm are represented using a particular permutation. The simplicity of the encoding scheme of the particles' position is the main reason for the algorithm superiority. In

this scheme, the particle is represented as a row of a number of elements. Every element number or dimension in the particle indicates a robot and every value recorded in each element denotes the corresponding task.

Example, assume that we have four robots and four tasks, and the particle is represented as {4 3 2 1} which denotes that the 4th task is allocated to the 1st robot, the 3rd task is allocated to the 2nd robot, the 2nd task is allocated to the 3rd robot, and the first task is allocated to the 4th robot.

5.2.1.2.2 DPSO velocity encoding

However, the velocity encoding scheme is different because each velocity element indicates the probability of choosing a specific task to a given robot and so on. The discrete version from the Particle Swarm Optimization that has been developed by Kennedy and Eberhart [71] applies a similar approach. However, in this technique, according to which task is going to be assigned for which robot, a specific map in the form of matrix is created for each particle, which represents the particles' velocity.

According to each particle's performance, a velocity is calculated for each particle in every iteration. For example, in order to get familiar with the update procedures of the velocity assume that we have a dynamic task allocation problem with n robots and m tasks. Thus, in this case the velocity map for each particle is introduced as an n*m matrix as follows:

$$v = \begin{bmatrix} p11 & \cdots & p1m \\ \vdots & \ddots & \vdots \\ pnm & \cdots & pnm \end{bmatrix}$$

where p indicates the probability of choosing a particular task for a specific robot. For example, every ith row has m elements pij, where j equals {1, 2, ..., m}, and each one of those elements indicates the probability of selecting a jth task for an ith robot.

Through the iterations, the particles continue learning new experience from both the social learning process and the cognition learning process. As a result, the velocity is updated for each

particle in every iteration. Based on this algorithm, by finding a profitable allocation of the task to a given robot, the learning operation is successfully accomplished.

The learning process to allocate various tasks to a group of robots is done as proposed by Hafiz and Abdennour [43] using various exemplars such as global best, personal best and local best particles. As the beneficial allocations give further promising results, the probability of their choice is raised through the update operation. An operator called a set difference operator and termed as, setdiff, needs to be calculated In order to determine the profitable components from the learning exemplar. The setdiff operator of both x and y particles can be calculates as follows:

$$\text{setdiff}(x, y) = \{e \in x \text{ and } e \notin x \cap y\}$$

Equation (5-1)

Four steps are necessary to accomplish the update procedure for the velocity as explained by Hafiz and Abdennour [43]. First, using the setdiff operator, the profitable tasks allocations must be calculated from the learning exemplars. According to the evaluation of these beneficial allocations, a set of probabilities are calculated as a second step. After that, as a third step, the performance for each particle is calculated against the swarm and then a comparison must be done between the current fitness value and the previous one. Finally, probabilities based on its position are updated according to the performance of each particle.

An illustrative example is provided in order to get familiar with the update procedure of the velocity. Assume a dynamic task allocation problem with four tasks, m=4 and four robots, n=4. Then, assume that an allocation for a specific particle xi equals {4 3 2 1}, while the global best allocation equals {2 3 4 1} and the personal best allocation equals {3 4 2 1}. So, the operator, setdiff must be generated in order to accomplish the process of the learning from exemplars.

$$S1 = setdiff(\text{pbesti}, \text{xi}) = setdiff(\{3\ 4\ 2\ 1\}, \{4\ 3\ 2\ 1\}) = \{3\ 4\ 0\ 0\}$$

Equation (5-2)

while,

$$S2 = setdiff(\text{gbesti}, \text{xi}) = setdiff(\{2\ 3\ 4\ 1\}, \{4\ 3\ 2\ 1\}) = \{2\ 0\ 4\ 0\}$$

Equation (5-3)

where S1 indicates the cognition experience, and S2 indicates the social experience and 0 represents the null value. Then, as a second step in the learning procedures, after the recognition of the beneficial allocations, the probability of choosing these allocations must be raised.

By applying this step on the corresponding example here, the social learning experience, S2, tells us that allocating the second and fourth tasks to the first and third robots is most likely to afford substantial outcomes. In addition, the cognition learning experience, S1, indicates that allocating the third and fourth tasks to the first and second robots, respectively, are profitable. In order to apply these learning results, the first step is to convert the position vector xi and the two learning experiences S1 and S2 to a matrix of n × m dimension according the following rule:

$$\lambda ij(x) = \begin{cases} 1, & if\ xi = j \\ 0, & otherwise \end{cases}$$

Equation (5-4)

Based on equation (5-4), the learning experience sets and the position matrices can be calculated as follows:

$$\lambda(xi) = \lambda(\{4\ 3\ 2\ 1\}) = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Equation (5-5)

$$\lambda(S1) = \lambda(\{3\ 4\ 0\ 0\}) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Equation (5-6)

$$\lambda(S2) = \lambda(\{2\ 0\ 4\ 0\}) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Equation (5-7)

Then, the update process for the ith particle's velocity is accomplished as follows:

$$\text{vi} = \text{vi} + (\text{c1 r1} * \lambda(\text{S1})) + (\text{c2 r2} * \lambda(\text{S2}))$$
$$= \begin{bmatrix} P11 & P12 + c2r2 & P13 + c1r1 & P14 \\ P21 & P22 & P23 & P24 + c1r1 \\ P31 & P32 & P33 & P34 + c2r2 \\ P41 & P42 & P43 & P44 \end{bmatrix}$$

Equation (5-8)

As can be shown from equation (5-8), the probability of the profitable allocations is raised according to the learning sets. So, there are rises in the values of p12, p34 (from social learning) and p13 and p24 (from cognitive learning) by the c1r1 and c2r2 coefficients, respectively.

After calculating the effect of the social learning and the cognition learning on the velocity update equation. Now, the third step is concerned with the contribution of the particle itself to the velocity update equation as illustrated in [43]. Thus, the probability of a specific particle is raised when the current fitness value of the particle is better than the fitness value of its previous position. If no enhancement is detected in the particle fitness value, then the probability has to be reduced in order to increase the exploration capability of the algorithm. Both the inertia weight and the improvement rate in the fitness value of the particle determine the decrease and increase amount in the probability of the velocity matrix. The update process in particle number i can be evaluated as follows:

$$\Delta i = \begin{cases} +(w * \delta i), & if \ \frac{f_i^{t-1}}{f_i^t} < 1 \\ -(w * \delta i), & if \ \frac{f_i^{t-1}}{f_i^t} \geq 1 \end{cases}$$

Equation (5-9)

where $\delta i = 1 - \left(\frac{f_i^t}{\max(F^t)}\right)$, f_i^t and f_i^{t-1} represent the fitness value for the particle number i in the current and previous iterations. Also, F^t represents a vector contains the fitness values for the whole swarm in the current iteration.

Finally, evaluate the velocity using the following equation (5-10) in order to update the probabilities of the previous position.

$$vi = vi + (\Delta i * \lambda(xi)) = \begin{bmatrix} P11 & P12 + c2r2 & P13 + c1r1 & P14 + \Delta i \\ P21 & P22 & P23 + \Delta i & P24 + c1r1 \\ P31 & P32 + \Delta i & P33 & P34 + c2r2 \\ P41 + \Delta i & P42 & P43 & P44 \end{bmatrix}$$

Equation (5-10)

It is essential to note that all the previous steps could be briefly summarized using the following update equation:

$$vi = (\Delta i \times \lambda(xi)) + (\mathrm{c1\ r1} * \lambda(S1)) + (\mathrm{c2\ r2} * \lambda(S2))$$

Equation (5-11)

From the previous equation, it can be shown that the update equation for the velocity in this discretized version of the Particle Swarm Optimization technique is similar to that of the continuous normal version of the PSO. Thus, the previous efforts that were spent by researcher on improving the original Particle Swarm Optimization is preserved as has been declared in [43].

Now, the algorithm steps for updating the particles' velocity can be outlined as follows:

1. Calculate the setdiff learning experience.

   $S1 = setdiff(\text{pbesti}, \text{xi})$

   $S2 = setdiff\ (gbest, xi)$

2. Calculate current position impact on the velocity update value, Δi.
3. Finally, calculate the velocity according to equation (5-11).

5.2.1.2.3 Position evaluation

The evaluation of the position demands choosing a task from the m available tasks to be allocated to each robot. To achieve this, the fundamental idea of the Particle Swarm Optimization that has been proposed by Kennedy and Eberhart is extended to utilize the probability of the velocity within this framework. The velocity values indicate the learning experience that is accumulated from both the social and the cognition learning process through the whole searching time. The velocity matrix must be turned into a probability matrix first, using equation (5-12), before calculating the particle's new position as explained by Hafiz and Abdennour [43].

$$nvi = \begin{bmatrix} \frac{v11}{\sum_{j=1}^{4} vij} & \frac{v12}{\sum_{j=1}^{4} vij} & \frac{v13}{\sum_{j=1}^{4} vij} & \frac{v14}{\sum_{j=1}^{4} vij} \\ \frac{v21}{\sum_{j=1}^{4} vij} & \frac{v22}{\sum_{j=1}^{4} vij} & \frac{v23}{\sum_{j=1}^{4} vij} & \frac{v24}{\sum_{j=1}^{4} vij} \\ \frac{v31}{\sum_{j=1}^{4} vij} & \frac{v32}{\sum_{j=1}^{4} vij} & \frac{v33}{\sum_{j=1}^{4} vij} & \frac{v34}{\sum_{j=1}^{4} vij} \\ \frac{v41}{\sum_{j=1}^{4} vij} & \frac{v42}{\sum_{j=1}^{4} vij} & \frac{v43}{\sum_{j=1}^{4} vij} & \frac{v44}{\sum_{j=1}^{4} vij} \end{bmatrix}$$

Equation (5-12)

So, the velocity matrix in equation (5-12) should be normalized, and it can be used as an index to calculate the new particles' position as proposed in Hafiz and Abdennour [43]. The elements in every row of the previous velocity matrix indicate the probabilities of selecting m tasks for a specific robot that owns the index of this row. For example, the probabilities for selecting the m tasks to the ith robot are represented in the ith row of the velocity matrix. The evaluation of the new position demands selecting a single task for each robot. Thus, it is required to choose a single element from every row of the velocity matrix.

In the dynamic task allocation problem, it is important to check the validity of the new evaluated position solution. In some cases, same task may be allocated to more than one robot while using the proposed framework in the new position evaluation. For example, when two or three robot is assigned to perform the same task, this gives a non-feasible allocation to the dynamic task allocation problem. It is essential to ensure that each robot has a unique allocation to a specific task in order to make sure that we have a valid solution.

Hafiz and Abdennour [43] specified that three steps need to be followed while evaluating the new position for every particle in order to get over this problem. The choice of a specific task for each robot based on various source set in every step. The velocity is the 1st source set, the current position of the particle is the 2nd source set, and the universal set is the third and last source set. The universal source set holds the whole tasks (m tasks).

5.2.1.2.3.1 Position evaluation pseudo code

Now, the pseudo code for the new position evaluation can be summarized as follows:

```
1: Calculate the normalized velocity;

2: Initialize the position vectors by assigning zeroes to all elements,
pos={0};

3: for i= 1 to m

4:    Choose the largest task probability within the ith row and allocate
      it to the ith location of  the particle's new position vector, pos;

5: end
```

```
6: Assign zero to the repetitive elements of the position vector;

7: If there are zero elements in the position

8:    Determine the zero elements in the position vector;

9:    Substitute these elements with their corresponding elements from the
      previous position vector;

10: End

11: Assign zero to the repetitive elements in the position;

12: If there are zero elements in the position vector

13:   Determine the zero elements in the position vector;

14:   Substitute these elements with their corresponding elements from the
      universal set;

15: End

16:      Update the particle's position xi=pos;
```

Next is an illustrative example on the update process of the position in order to familiarize ourselves more with these procedures. Let the velocity matrix vi of the particle xi after the velocity update operation is given as:

$$vi = \begin{bmatrix} 0.54 & 0.36 & 1.27 & 0.64 \\ 0.58 & 0.42 & 0.87 & 1.52 \\ 0.71 & 0.04 & 1.82 & 0.94 \\ 2.44 & 0.53 & 1.05 & 0.88 \end{bmatrix}$$

In order to find a probability of choice for every element in the matrix, the velocity matrix needs to be normalized first:

$$nvi = \begin{bmatrix} 0.19 & 0.13 & 0.45 & 0.23 \\ 0.17 & 0.12 & 0.26 & 0.45 \\ 0.2 & 0.01 & 0.52 & 0.27 \\ 0.5 & 0.11 & 0.21 & 0.18 \end{bmatrix}$$

So, this normalized matrix is utilized as the main and 1st source set for the new position evaluation to the particle number i. The largest element among all the probabilities in each row is chosen for the position vector, POS. So, in this case, POS = {3 4 3 1}T. Then, the validity of the

POS solution must be checked, and just the unique allocation could be accepted and preserved in the new position vector. Since the third task is allocated to the first and third robots simultaneously, zero should be allocated in the 3rd element. Thus, the position vector POS becomes {3 4 0 1}T.

Then, the elements with the zero value are determined. After that, the previously determined zero elements are allocated with the corresponding tasks from the current position vector which represents the 2nd source set in order to accomplish the new position evaluation. So, the current position of the particle which is equal to {4 3 2 1} must be checked out in order to allocate another task to the 3rd element. Since, the use of the current position to allocate the 3rd element gives a valid solution which is {3 4 2 1}T, it is enough to be used and the universal source set is not necessarily. Finally, the ith particle's new evaluated position is evaluated to be: xi = POS = {3 4 2 1}.

It is worthily mentioned that a valid solution could also be found using only the velocity without referring to both the current position and the universal set and limit ourselves with the membership mapping as proposed by Pang et al. [72] and Liu et al. [73, 74]. However, this method results in a lower quality solution since some smaller probability values could be involved in the evaluated position vector.

5.2.1.3 DPSO pseudo code

Now, the complete pseudo code to solve the dynamic task allocation problem based on the discretized version of the PSO could be summarized as follows:

```
The initialization step:

    1: Initialize the algorithm's parameters, w, c1, c2, refresh gab
    (rg), and the stagnation count (cnt).

    2: Initialize the particles' positions randomly, {x1, x2, ......,
    xi, ......, xn}.

    3: Initialize the particles' velocity using normal distribution
    [0,1].
```

```
4: Calculate the fitness value for all particles in the swarm, and
initialize the personal best and the    global best to be equal to
the initial position of the particles.
```

The evaluation step:

```
1: For i =1 to maxit
2:      If cnt >= rg  // stagnation detected
3:           Reinitialize the particles' velocity;
4:           cnt=0;
5:      End
6:      Update the weight, w(i) = w0 -  (w0−wf)∗i / maxit  ;
7:      For j= 1 to n
8:           Calculate the ith particle's velocity;
9:           Calculate the ith particle's position;
10:          End
11:          Calculate the fitness values;
12:          Save the previous fitness values for the whole swarm;
13:          Calculate the new position's fitness value;
14:          Update the personal and the global best positions;
15:          Increase the stagnation counter if the global best
             kept unchanged, cnt = cnt + 1;
16:     End
```

5.2.1.4 DPSO parameters' setting

One of the most essential parameters to improve the searching capabilities of the Particle Swarm Optimization technique as announced by Wang and Liu [54] is the inertia weight w. A big inertia weight increases the exploration capabilities of the algorithm by opening the research within new areas whereas a kind of small inertia weight increases the exploitation capabilities via precise searching within the same current area.

Thus, an appropriate choice of the value of the inertia weight results in a sufficient balance between the global exploration and the local exploitation, and leads to a kind of less number of iterations to discover an efficient outcome as clarified in [54]. Therefore, reducing the inertia weight linearly from a kind of large value to a small value through the path of the Particle Swarm Optimization run, leads to increase the global search capability of the algorithm at the beginning of the search process, while increasing the local search capability close to the termination of run.

Actually, in this thesis the inertia weight is chosen to be as follows:

$$w = w_{max} - \frac{w_{max} - w_{min}}{I_{max}} * I$$

Equation (5-13)

where w_{min} represents the inertia weight's final value, w_{max} represents the inertia weight's initial value, I_{max} represents the maximum number of iterations, and I indicates the current iteration number.

5.3 SA

Hasan, Locatelli, and Mitra [75, 76, 77] explained that unlike the steepest descent method, Simulated Annealing technique can be considered as a local meta-heuristic searching algorithm that uses a mechanism to avoid getting trapped into a local minimum solution. It adapts a metropolis approach, which is a random probabilistic acceptance technique, which enables some worse solutions to be selected as a next step with a specific probability.

The molecular movement in materials over the annealing process is stimulated using the SA algorithm. If there is enough time for the molecules during the cooling process to move, they will move towards the highest stability and lowest energy state. The annealing operation can be implemented in a mathematical way to find the global optimum solution, which is correlated with the highest stability state by applying the probability acceptance technique of the metropolis mechanism and disturbance technique.

Simulated Annealing confirmed its superiority through years in solving many combinatorial discrete problems. Simulated Annealing technique's steps can be summarized as follows:

1. Another solution from the neighborhood of the current solution(s) is selected.
2. The fitness value for this new solution is evaluated since the fitness will change for sure when the solution changes.
3. After that, the difference between the new and current solutions' fitness is calculated, D=E(s') _ E(s).
4. Then, the new solution is unconditionally accepted if D < 0.
5. However, the new solution is accepted with a probability of exp(-D/t) where t represents the temperature, if D > = 0.

It is worth to mention also that, the algorithm begins with a high temperature value, and decreased progressively until ending with a very small value close to zero. The algorithm runs for a given number of iterations at every temperature value. These iterations are called the temperature length. Finally, the algorithm stops after finishing a specific number of iterations or after satisfying the end condition.

5.3.1 SA parameters' selection

A great attention must be given to the parameters' selection of the Simulated Annealing technique since it has essential consequences on the efficiency of the algorithm's performance. Better outcomes could be found out of a slow searching. However, it could lead to worse convergence characteristics. Thus, it is always a compromise between both of them. Parameters' choice somehow depends on the characteristics of the problem itself.

There is a collection of essential parameters associated with the SA algorithm, such as the disturbance mechanism, cooling process, initial condition and equilibrium condition. Shieh, Kuo and Chiang [19] clarified that by the proper adjustment of these parameters, the quality of the outcomes of the algorithm and the algorithm performance could be significantly enhanced.

5.3.1.1 Initial condition choice

During the optimization process and especially at the beginning of the searching procedure, it is preferred to accept some bad solutions with a high probability in order to increase the exploration function of the algorithm. Therefore, it is better to choose a high temperature at the beginning. If a very small initial temperature is selected, there will be a high possibility for the algorithm to be trapped into a local minimum solution.

On the other hand, if a very high initial temperature is selected, the algorithm's time complexity will be extremely large. Thus, through years of experience, it has been discovered by Wang and Liu [54] that, by choosing an initial temperature equals to the maximum difference between two neighboring solutions from the whole solutions set, it could lead to better convergence time and cost value. In addition, Shieh, Kuo and Chiang [19] reported that a multiple of this largest difference could be also a very good choice for the initial temperature value. However, it is better to keep it in a reasonable value, not too large and not too small.

5.3.1.2 Establishing a disturbance mechanism

The disturbance process is one of the most significant parts in the SA algorithm because most of the convergence time is spent in the metropolis mechanism and it has an essential effect on the performance of the algorithm and the quality of solution. Thus, it is necessary to have a tight control over the disturbance process to be able to find a good solution.

The main function of the disturbance mechanism is to search for a feasible solution within the neighborhood of the current solution. Then, the metropolis process decides whether to accept this worse neighboring solution with a certain probability or not. This mechanism gives SA the ability to jump out of a local minimum solution.

However, it could also steer the searching process towards a direction, which is in the opposite direction of the global optimal solution, which in turns could raise the time complexity of the algorithm as illustrated in [19].

5.3.1.3 Equilibrium condition determination

The state of the system at a certain temperature in which there is no more energy flow is called the equilibrium condition. This means that, the energy flow in the system becomes insignificant after a finite number of disturbances during the algorithm run. This process is referred to as the acceptance rate, and it can be formulated as follows:

$$\text{Acceptance rate} = \frac{\text{Total number of acceptance}}{\text{Total number of disturbances}}$$

Equation (5-14)

When a disturbance mechanism produces a new neighboring state to the current state, the metropolis mechanism decides whether to accept this new state or not. The total number of acceptance in the previous equation should be increased by one, if the new state is accepted. After a finite number of disturbances, the acceptance rate would indicate the amount by which the algorithm does accept new worse solutions. Therefore, stepping forward to a new temperature depends on the equilibrium condition of the system.

If the system reach to the equilibrium condition at a certain temperature and the system is not changing that much anymore, this means that the acceptance rate is very small "near zero", and the system should step forward to the next temperature. However, it has been explained by Shieh, Kuo and Chiang [19] that if the system is still in an active state and there are many disturbances occurs which means that the acceptance rate is near to one, the system should continue processing at this temperature until an equilibrium state is obtained.

5.3.1.4 Method of cooling

As the Simulated Annealing simulates the annealing process, then the cooling rate must be reduced in accordance with the decrease in the temperature similar to what happened in physics. Hence, the temperature is progressively reduced during the Simulated Annealing operation. Shieh, Kuo and Chiang [19] specified that this process helps the system to be transferred from an active state to an equilibrium stable state. In other words, the system should be cooled faster at the beginning of the annealing process and cooled slower during the advanced stages.

The equation tk = λ tk-1 is a great selection to determine the temperature value at each iteration since it satisfies a compromise between a good quality solution and fast convergence time. Of course, according to the previous equation, the temperature decreased faster when the cooling rate parameter λ is small.

5.3.1.5 Temperature length

The temperature length indicates the number of runs which taken place at each iteration with the same temperature value. The best choice for this parameter is the number of the neighborhood for a specific solution.

5.3.1.6 Temperature end parameter

The last essential parameter in the Simulated Annealing algorithm is the temperature end parameter (tend). Wang and Liu [54] reported that this parameter is used to stop the algorithm when the current temperature value becomes less than the temperature end value. It is worth to mention that tend is always selected to be a very small value.

5.4 DPSO/SA

Both the exploration and exploitation searching properties are required to be able to find a global or near global minimum solution as has been clarified by Shieh, Kuo and Chiang [19]. The exploring characteristic is essential for applying the search process across the whole search space to be able to find the area in which the global optimum solution exists. The exploiting characteristic is also significant in order to apply a gradient search mechanism in a certain local area to search within this region for the global optimum solution.

The use of both the exploring and the exploiting techniques together enhances the searching ability to be done in the whole search space and provides a way of jumping out of a certain local minimum solution, thereby enhancing the ability of exploring the global minimum solution.

However, nothing in the heuristics searching techniques could guarantee finding out the global minimum solution. One of the most obvious disadvantages of the PSO technique is that if there

are a local and a global optimum solution in a certain objective function, and taboth of them are separated from each other with a specific distance. Then, if the initial random generated population is closer to the local optimal solution than the global optimal one, the particles' personal and global best will move towards the local optimal solution and it will lose the ability to jump out of this local optimum solution. Hence, it will lose the global searching capability. Moreover, the speed of convergence towards the final stages of iterations becomes obviously slow.

In addition, the DPSO technique is a problem-independent algorithm. This means that the only information pertinent to the problem itself that is required while searching for the solution is the fitness function. This characteristic provides more simplicity and robustness for the DPSO algorithm than the other optimization techniques. However, the DPSO also suffers from the disadvantage of the possibility of being trapped in a local optimum solution especially when the problem to be solved is quite complex because at the end of the day it is a stochastic heuristic searching technique. Another disadvantage could be the reduction of the global search capabilities at the final iteration in the algorithm.

On the other hand, it has been notified by Wang and Liu [54] that the searching capabilities of the Simulated Annealing technique could be improved by the precise selection of the cooling rate, and the problem of getting trapped into a local optimal solution could be restricted by applying a specific acceptance rate. Thus, we thought of trying to limit the issue of the local optimum solution and provide a better control for the searching ability through the careful design of the cooling parameter and neighborhood topology structure of the Simulated Annealing technique.

Thus, in this thesis, a hybrid algorithm is developed in order to try to address and solve the main two problems of the PSO technique. Shieh, Kuo and Chiang [19] reported that when small variations are applied to the temperature parameters of the SA technique, and an equilibrium condition is obtained for every temperature value during the searching process, the SA technique could have a probability of one for finding the global minima. Moreover, the SA algorithm owns a great capability of being able to avoid becoming trapped into a local optimum solution due to

the metropolis process. However, SA has a disadvantage of a slow convergence time due to the small temperature variations to ensure finding the good-quality solution after running the algorithm.

The essential adjustment of the hybrid PSO/SA algorithm is that it fuses the parallel movement structure of the PSO technique into the disturbance searching mechanism of the SA algorithm. In this way, the algorithm can explore for solutions in more paths, increasing the probability that the global optima could be found. In addition, the parallel processing capabilities of PSO reduces calculation time. Also, a good quality solution could be easily found as a result of combining the advantage of the global search ability of the SA algorithm with the fast convergence characteristics of the PSO algorithm. Thus, this hybrid technique owns very easily realizable characteristics.

5.4.1 DPSO/SA pseudo code

The most significant characteristics and properties of the hybrid PSO/SA are summarized in the following notes:

Note1. Since the Particle Swarm Optimization technique owns a simple implementation and an inherent parallel structure, just few adjustments are needed to develop the hybrid algorithm.

Note2. The hybrid PSO/SA algorithm could have both a powerful global exploration and local exploitation abilities due to the dynamic adaptation of the temperature value of the SA. The particles at the early steps in each temperature value own a high probability of accepting worse solution as the next solution, which improve the exploration ability of the algorithm. Then, through the iteration, the temperature becomes lower and the probability for accepting a worse solution as the next solution is decreased which generates a balance between the global exploration and the local exploitation behaviors. In addition, near the end of the iterations and as the temperature becomes closer to zero, nearly no more worse solutions is accepted which enhance the ability of exploiting the promising solutions in the optimal areas.

Note3. In the literatures [52, 53], there is a mathematical proof that the Simulated Annealing algorithm could find the global optimum solution with a reasonable convergence time with a probability equal one. On the contrary, Particle Swarm Optimization technique does not always guarantee an optimal solution. Thus, the hybrid algorithm can be considered as a parallel structure SA algorithm. As a result, the proposed algorithm guarantees finding the global optimal solution with a probability equals to one and at the same time enhance the time complexity and convergence characteristics due to its parallel structure.

Note4. Finally yet importantly, this hybrid PSO/SA algorithm could be used to optimize any other problems due to its generality. In this research, the proposed hybrid algorithm is deployed to optimize fifteen different swarm sizes to ensure its robustness and scalability.

Based on the above illustration, the complete Pseudo code for the hybrid DPSO/SA algorithm can be outlined as follows:

```
Start

Step1. Initialization step

      1:    Initialize position, velocity and number of population for each
            particle in the swarm;

      2:    Calculate the fitness value for each particle;

      3:    Initialize the personal best position with the initial position
            of the particle;

      4:    Initialize the global best with the minimum particle in the
            swarm;

      5:    Initialize the acceleration constants c1, c2;

      6:    Initialize the inertia weight constants wmax, wmin;

      7:    Initialize the maximum number of iteration constant;

      8:    Initialize the initial temperature, temperature end and the
            cooling rate parameter for the SA technique;

Step2. Evaluation step

(2.1) PSO

      1:    While (the termination condition is not satisfied)
```

```
2:          {
3:          Iteration++;
4:          Generate new position and velocity using Eqs.(11)and
            (12);
5:          Evaluate the new fitness values;
6:          Update the swarm {
7:                Adjust the personal best for each particle;
8:                Adjust the global best of the swarm;
9:                }
10:         }
```

(2.2) SA

```
1:    Initialize the algorithm's solution with the global best particle
      in the swarm.
2:    T = T0;
3:    While (T > Tend)
4:    {
5:          Find a new neighboring solution from the old one;
6:          Evaluate the fitness of the new neighboring solution;
7:          Calculate the fitness difference D = E(s') - E(s);
8:          If min[1,exp(-D/T)] > rand[0, 1]
9:          {     Accept the new solution s';
10:         }
11:         Adjust the global optimum solution if the new solution is
            the best solution found so far;
12:         Update the temperature t = λ t_k_1;
13:   }
Step3. Output the best solution and its fitness value.
End
```

It can be shown from the pseudo code that the algorithm begins with the PSO technique to find an initial solution followed by the SA algorithm. If the Simulated Annealing part is cancelled,

the algorithm simply turned into a PSO algorithm. In addition, the algorithm can be turned into the simple well-known SA algorithm if the PSO part is cancelled out or by setting the population size of the swarm to one. Using this algorithm maintains the generality of both the SA and PSO. Furthermore, by just modifying the fitness function within the algorithm, it could be utilized to solve any other combinatorial discrete optimization problems.

5.4.2 DPSO/SA parameters' selection

Because of the shortage in the divergence within the final iterations of the search operation in the Discrete Particle Swarm Optimization technique, it sustains from early convergence. Thus, it is substantial to keep the variety within the swarm over the whole search space in order to find a promising solution for NP-hard problems like the dynamic task allocation problem. There are many methods to do so. One of the easiest ways to achieve this as proposed by Hafiz and Abdennour [43] is to use the restart technique if stagnation happens.

The restart technique can be translated in this discretized version from the Particle Swarm Optimization as velocity re-initialization for each particle in the swarm. This velocity re-initialization increases the capability of exploration within the swarm, because the basic source for the new position construction is the velocity. However, the only part, which is reinitialized in the whole process, is the velocity, and neither the personal best nor the global best positions are changed.

The notion of the refresh gap is utilized to solve the problem of the stagnation and all the parameters, which are somehow involved with this concept, are calibrated. Therefore, various refresh gaps' values are tried in the algorithm until a suitable value is detected. Furthermore, the algorithm is tried on different swarm dimensions in order to make sure that the performance of the algorithm becomes stable and unbiased.

The performance of the algorithm is observed while adjusting the refresh gap's value to be five, ten, twenty-five, fifty and one hundred percent of the total number of iteration. We figure out that the refresh gab has a considerable effect on the outcomes of the algorithm.

A large number of restarts are permitted if a small refresh gap is chosen which of course results in a preferable performance. However, if a large refresh gap is selected, the performance of the algorithm becomes worse. Also, no restarts is supplied if the refresh gap is given the value of 100% of iterations. The best performance is achieved by setting the refresh gap within the range of 30 percent of the total number of iterations.

5.4.3 LPSO VS. GPSO

It is very substantial to make some sort of compromise between the exploitation ability and the exploration ability of the swarm in order to increase the robustness of the algorithm, see figure 5.2. Many efforts have been spend in last recent years in an attempt to find a solution to the issue of the algorithm early convergence. It has been clarified by Hafiz and Abdennour [43] that these efforts can be divided into categories according to either the used learning exemplars, the way the information exchanged among particles (neighborhood topology), or the use of different parameters control.

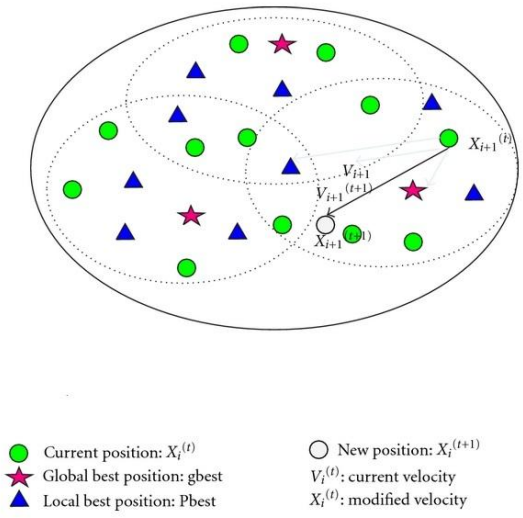


Figure 5.2: PSO example [78].

The basic idea behind the Particle Swarm Optimization technique is to seek for learning and gaining experience through the exchange of information among the particles in the swarm. The way by which the particles within the smarm share the information has a direct influence on the outcomes of the algorithm. The proposed algorithm in this thesis is based on the global approach with the star topology where the global best particle in the whole swarm shares its information with all the other particles in the swarm. By using this topology, the convergence time of the algorithm becomes very fast. However, there is an increasing possibility for the algorithm to get trapped into a local minimal solution.

This problem actually is solved using the proposed algorithm by combining the Simulated Annealing with the global version of the Particle Swarm Optimization technique. As discussed earlier, Simulated Annealing has the ability to avoid getting trapped into a local optimal solution using its metropolis technique. Also, the refresh gap approach is applied into the proposed algorithm to ensure that the algorithm could easily reach the global optimal solution and avoid the local optimal one. Thus, there is no need to use a local neighborhood topology and it is better to use the generic global Particle Swarm Optimization technique in order to increase the algorithm convergence capability.

Some researchers solve this issue by restricting the knowledge sharing among the particles to a limited number of particles instead of the whole swarm via using one of the most common neighborhood topologies. The most commonly used topology in terms of research and publications, is the local best approach. This approach decreases the algorithm's possibility of being trapped into a local minimal solution since it increases the exploration capabilities of the particle in the swarm through the reduction of the speed of the knowledge sharing among the particles in the swarm.

In this approach, instead of learning from the global best particle within the whole swarm particles, the particles start learning from the local best particle within its neighborhood. Figure 5.3 shows some of the most commonly used neighborhood topologies. It is actually very easy to apply the Discrete Particle Swarm Optimization framework to this approach. The steps for

discretization remain the same except the part of evaluating the learning sets. The learning sets could be modified as follows:

1. Determine the best particle in the local neighborhood. Then, choose it as the local best solution to be used for the velocity evaluation.
2. $S1 = setdiff\ (pbest, xi)$
3. $S2 = setdiff\ (lbest, xi)$

However, the method by which the position is evaluated, is the same for all different approaches.

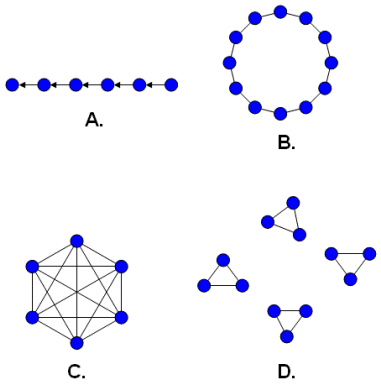


Figure 5.3: Neighboring topology [79].

Despite of the simplicity of applying a local neighborhood topology to the swarm, and the advantages, that could be achieved out of using this approach, like increasing the exploration capabilities of the particles within the swarm. However, it is better to use the global star neighborhood topology to gain the advantage of the fast convergence meanwhile increasing the exploration capabilities using SA technique with the metropolis mechanism and the concept of the refresh gab.

Chapter 6

Results and Discussion

This chapter discusses the outcomes from both the simulation and the real experiment. Also, the proposed algorithm parameters are precisely chosen using a suitable delicate method. Finally yet importantly, the communication overhead and the overall throughput for the system are evaluated.

6.1 Introduction

The simulation is implemented using MATLAB to prove the efficiency of the proposed hybrid DPSO/SA algorithm in solving the minimal time dynamic task allocation problem in swarm robotics. The specifications for the platform which is used in the simulation are as follows:

1. Intel core i7 processor 2.4 GHZ.
2. 8 GB RAM.

The results of the proposed algorithm are compared with both the SA algorithm and the DPSO algorithm as heuristic algorithms and with the Hungarian algorithm as an exact solver. Fifteen various swarm dimensions are used to compare between the algorithms' performance. Due to the randomness in most of the meta-heuristic algorithms, they obtain a different solution every time the algorithm runs. Thus, each swarm instance is replicated 50 times and the arithmetic mean is calculated in order to ensure the stability of the results. The prime parameters that are used for the comparisons are the quality of solution, the execution time of the algorithm, and the stability of the algorithm. To ensure the fairness during the simulation among all the four algorithms, all of them continue running until an equilibrium state is obtained.

6.2 Methodology

This thesis proposes a hybrid PSO/SA algorithm, which fuses the concept of SA algorithm to overcome the PSO algorithm's imperfections. SA is used to promote the PSO's local search capability. The hybrid algorithm not only possesses the pros of PSO and SA but also omits the cons of both of them.

The basic concept of the DPSO algorithm is introduced in the main framework of the hybrid PSO/SA algorithm. Thus, a finite number of initial particles' velocities and positions are generated and they kept updated through iterations. Then, after a specific number of iterations, the SA algorithm takes the global best solution and starts performing local search operations. The SA continues processing repeatedly until the satisfaction of the termination condition.

The most significant properties of the proposed algorithm can be highlighted in the following notes:

6.2.1 Initial population of DPSO

Since Meng, Zhang and Fan [80] clarified that it is better to raise the degree of randomness at the beginning of the process to maintain the diversity of the population, thus the PSO/SA algorithm starts with a randomly generated candidate solutions (particles).

6.2.2 Local searching ability of SA

In the hybrid PSO/SA algorithm, SA runs after an optimal solution is found by DPSO. As the SA's temperature decreasing, the algorithm only searches within the neighborhood of the current best solution. Then, when the new solution found by the algorithm is better than the previous one, the algorithm would accept it without any conditions. However, if the new solution is worse than the previous one, the algorithm will accept it with a certain probability. This process is continued till the temperature of the algorithm declines under a certain temperature.

Hafiz and Abdennour [43] illustrated that unlike PSO, SA operates on a single solution. The working solution is updated by performing a move (swapping of solution elements) in each iteration. The reason for choosing SA specifically to combine it with the DPSO is that Simulated Annealing technique is inherently discrete unlike the DPSO and it is interesting to study and compare between the searching behaviors of two such different techniques.

6.2.3 DPSO/SA decentralization

The proposed algorithm is decentralized which means that neither a global knowledge, nor messages' broadcast nor a multi-hop communication would ever be used. There is no central unit

to take care of the task allocation, and each individual in the swarm has to identify the task it must perform via the communication with only their neighbors. This decentralization is essential to avoid the restrictions that could result from the communication range limitations.

It is also essential to mention that, the simulation in this research is an offline simulation, so that the robots and objects positions are predetermined prior to the simulation run. However, the real experiment is performed online and the robots and objects positions and the population size could change repeatedly during the experiment.

6.3 Parameters' Optimization

Since the parameters of the heuristic optimization techniques affect both the quality of solution and the performance of the algorithm as declared by Shieh, Kuo and Chiang [19], they have to be precisely chosen using a suitable delicate method. It has been demonstrated in [19] that the efficiency of both the DPSO and the SA algorithms significantly depends on five main parameters. These parameters are c1, c2, T0, λ, w.

The simulation in this research is done for 7, 15, 30, 40, 50, 60, 70, 80, 90, 100, 500, 600, 800, 1000, and 2000 swarm sizes. Each case of theses swarm sizes is replicated 50 times. The parameter value, which obtains the highest performance, is chosen for the algorithm. Each one of the five parameters is tested independently while keeping the other five parameters constant. This process continues until a suitable value is found, then same procedures are replicated to find proper values for the other five parameters.

First, to select the optimal value for the initial temperature, simulations are implemented with different values for the initial temperature. In this step, the other parameters' values, which are not optimized, yet should be kept constant.

6.3.1 Initial temperature 'T0'

Due to the change in the value of the initial temperature, the initial acceptance rate of the hybrid algorithm is also affected. As shown in equation (6-1), this acceptance rate guarantees that the algorithm searches for the solution globally in the whole search space, and helps the global best and personal best solution to jump out of the local optimum solution. Thus, it is good to

sometimes accept worse solutions to avoid getting trapped into a local optimum solution, thereby enhance the exploration property for the algorithm. This fixed acceptance probability that is used for accepting new worse solutions are termed as the metropolis acceptance rule.

Time could be wasted in exploring new unnecessarily regions when a very high initial temperature is selected. On the contrary, the algorithm becomes very slow and large number of iterations is needed to find a suitable solution if a very low initial temperature is selected. Thus, the time complexity increases. Therefore, the simulation is run with different values of initial temperature ranging from 1 to 100 with an increase by 10 in the proceeding runs. The best performance and quality of solutions are detected using an initial temperature of 100.

$$P1 = \mathbf{e}^{\left(\frac{-\Delta \mathbf{X_{max}}}{\mathbf{T0}}\right)}$$

Equation (6-1)

where

T0, initial temperature

ΔX_{max} , the difference between the fitness value of the worst solution and the initial solution

P1, acceptance rate of the worst new solution

6.3.2 Cooling rate ' λ '

The acceptance rate close to the end of iterations is affected by the value of λ. If the value of λ is very small close to the end of the iterations, the algorithm could get trapped into a local minima and lose the ability of jumping out of this local minimum solution which may enforce us to rerun the algorithm until reaching a reasonable global minima. This problem results in raising the time complexity of the algorithm. Therefore, the simulation is run with different values of λ ranging from 0.4 to 0.95 with an increase by 0.05 in the proceeding runs. The best performance and quality of solution are detected using a cooling parameter value of 0.9.

6.3.3 Inertia weight 'w'

The momentum weight of the hybrid PSO/SA algorithm affects the amount of reference to a new particle's solution in comparison with the reference to the old one. Thus, the larger the momentum weight is, the larger effect of the old solution is considered. In the last few years, researchers have observed that it is better to dynamically reduce the value of the inertial weight as the iterations proceed. It is also found that a better performance and a better quality of solution could be yielded from a maximum inertial weight of 0.9 and a minimum inertial weight of 0.4.

6.3.4 Acceleration constants 'c1, c2'

The parameters of learning factors, c1 and c2 affect the moving speed in the direction of the global and personal best as notified by Wang and Zheng [56]. The larger the learning factors are, the bigger steps the particles take towards best solutions locally and globally. Through simulations and experiments, it is detected that better outcomes could be obtained by setting the value of c1 to 2 and the value of c2 to 2.

To sum up, the parameter constants are chosen to be as follows:

SA's parameter:

1. T0 = 100.
2. $\lambda = 0.95$.

PSO's parameter:

1. c1=c2=2.
2. $w_{max} = 0.9$.
3. $w_{min} = 0.4$.
4. Population size=25.
5. Maximum number of iterations depends on the problem dimension.

6.4 Simulation Results and Comparisons

6.4.1 DPSO vs. SA

In our simulation, we first compare between DPSO and SA techniques. So, this comparison is going to be presented in this section before extending our problem dimension to include an increasing number of swarm sizes.

So, in this section only two study-cases are considered. Since we need to have at least more than two robots in the environment to be considered as a swarm, Thus, in the first study-case, we started with only 7 robots and 10 tasks as the smallest swarm size in our simulation. In such case, only 7 tasks were allocated while the remaining 3 tasks were waiting in the waiting list. The algorithm is executed periodically. Thus, if any robot finished its allocated task, left for recharging or new tasks were explored which need to be allocated, the algorithm should calculate a new allocation for the robots in the swarm. We start the simulation with 7 robots since we have in the lab only 7 robots, so we considered it the minimum swarm size.

Then, we increased the swarm size gradually to test the algorithms' behaviors in different problem dimensions. So, in the second study-case, we considered 10 robots and 20 tasks. Thus, only 10 tasks were allocated at the beginning. The allocation for each particle in the swarm is represented as a raw with a number of elements equals to the number of robots in the swarm and the number, which is written in each cell, represents the task allocated to the corresponding robot.

For example, let the task allocation be as shown in the second row of table 6.1. The first row is just an identifier for the robot number to which the corresponding underneath task is allocated. In this case, task 3 is allocated to robot 1, task 5 is allocated to robot 2, task 8 is allocated to robot 3, task 2 is allocated to robot 4, task 9 is allocated to robot 5, task 4 is allocated to robot 6, and task 1 is allocated to robot 7. This allocation is corresponding to the first case study.

Table 6.1.Allocation solution for the first case study

r1	r2	r3	r4	r5	r6	r7
t3	t5	t8	t2	t9	t4	t1

Another example on the allocation corresponding to the second case study is as shown in table 6.2. 20 tasks are allocated to 10 robots. Therefore, task 6 is allocated to the robot 1, task 14 is allocated to the robot 2, task 9 is allocated to the robot 3, task 11 is allocated to the robot 4, task 2 is allocated to the robot 5, task 19 is allocated to the robot 6, task 3 is allocated to the robot 7, task 17 is allocated to the robot 8, task 8 is allocated to the robot 9, and task 5 is allocated to the robot 10.

Table 6.2.Allocation solution for the second case study

r1	r2	r3	t4	r5	r6	r7	r8	r9	r10
t6	t14	t9	t11	t2	t19	t3	t17	t8	t5

In this simulation, tasks were displayed as yellow circles, while robots were represented as red squares to be able to differentiate between tasks and robots. Allocations between robots and tasks are addressed as a black line extended from each robot to its allocated task. Figure 6.1 and 6.2 show the best solution A* resulting after 500 iterations for both PSO and SA, respectively. Figure 6.1 show that PSO gives a better allocation as each robot is assigned to its nearest task. However, in figure 6.2, some robots select a far task rather than the nearest one.

Simulation results corresponding to the second case study are shown in figure 6.3 and 6.4 for PSO and SA, respectively. If the distance between the robots and their corresponding tasks is tracked in both cases, it could be easily explored that the distance traveled by robots in figure 6.3 is much smaller than the traveled distance in figure 6.4.

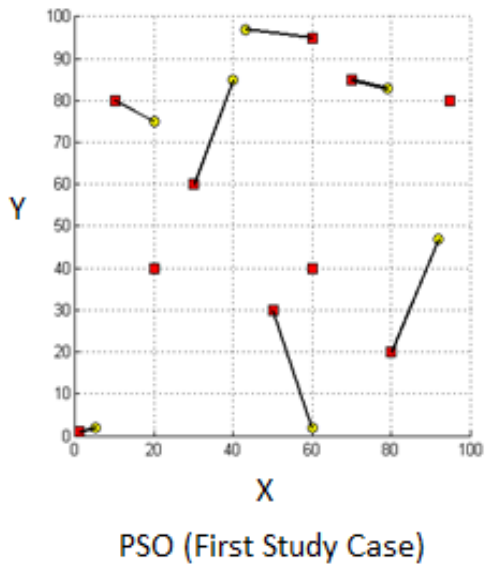


Figure 6.1: PSO solution resulting after 500 iterations in the first study case

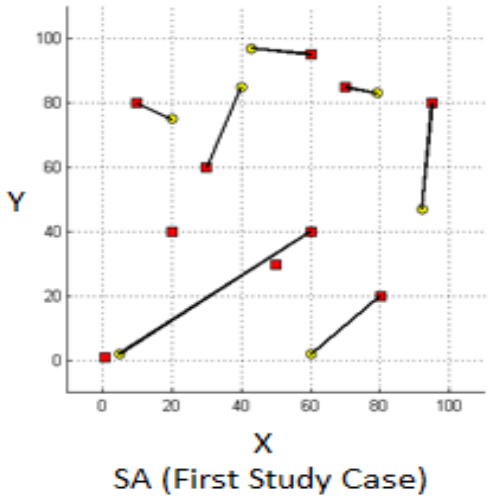


Figure 6.2: SA solution resulting after 500 iterations in the first study case.

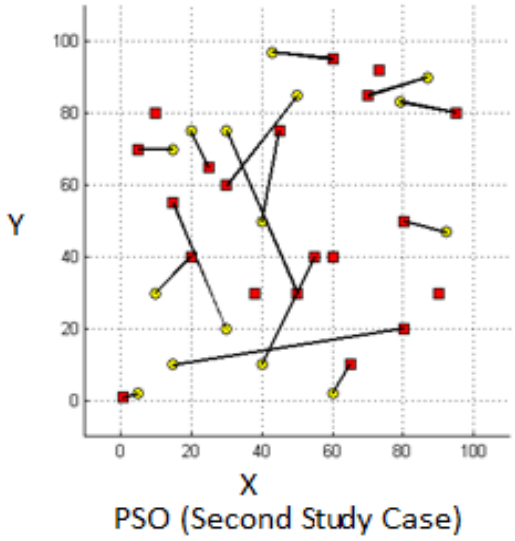


Figure 6.3: PSO solution resulting after 500 iterations in the second study case.

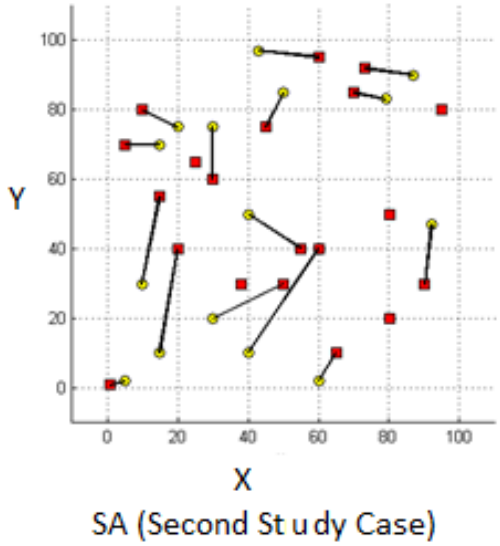


Figure 6.4: SA solution resulting after 500 iterations in the second study case.

A comparison about the MTDTA performance is made between PSO and SA. Both of them are executed for 100, 200, 300, 400, and 500 iterations and the corresponding algorithm run time for each case is also calculated.

Table 6.3 and 6.4 show the results from the first and second case studies, respectively. It can be deduced from table 6.3 that the distance travelled using PSO is smaller than that of the SA in almost all cases and the gap between the PSO's and SA's fitness kept increasing till reaching its maximum value after 500 iterations. For example, the difference between the travelled distance by the robots in PSO and SA after 100 iterations equals (207.9 cm - 165.4 cm= 42.5 cm). However, the difference after 500 iterations equals (199.6 cm - 128.6 cm= 71 cm). Furthermore, it can be shown that after 150 iterations, SA's fitness gets trapped in a local minimum at 199.6 cm and it never changes. However, PSO's fitness kept changing until reaching the global minimum value at 128.6 cm.

Table 6.3: Simulation experiment data for the first case study

Optimization technique	Number of iterations	Algorithm run time	Distance travelled by robots
PSO	100	4.9 sec	165.4 cm
	200	5.8 sec	157.7 cm
	300	9.5 sec	153.5 cm
	400	9.6 sec	143.2 cm
	500	14.6 sec	128.6 cm
SA	100	14.3 sec	207.9 cm
	200	18.9 sec	199.6 cm
	300	18.9 sec	199.6 cm
	400	24.1 sec	199.6 cm
	500	23.8 sec	199.6 cm

By looking at table 6.4, it can be figured out that the PSO's travelled distance (419.3 cm) is smaller than that of the SA (426.7 cm) for small number of iterations. However, when the number of iterations increases above 250, the PSO's travelled distance which is (356.5 cm) becomes larger than that of the SA which is (250.9 cm). Moreover, in this case study, the PSO's fitness value gets trapped in a local minimum at 356.5 cm after 400 iterations, while the SA's fitness value kept decreasing until reaching the global minimum solution at 250.9 cm.

Table 6.4: Simulation experiment data for the second case study

Optimization technique	Number of iterations	Algorithm run time	Distance travelled by robots
PSO	100	7 sec	431.9 cm
	200	6.4 sec	419.3 cm
	300	14.1 sec	411.8 cm
	400	12.7 sec	356.5 cm
	500	20.8 sec	356.5 cm
SA	100	19.1 sec	444.3 cm
	200	27.5 sec	426.7 cm
	300	25.3 sec	384.2 cm
	400	32 sec	319.2 cm
	500	31.6 sec	250.9 cm

In the first case study, it can also be seen from figure 6.5 that no matter how much we increase the number of iterations, the cost function gives the same results after 200 iterations. From this observation, it can be concluded that the algorithm based on the Simulated Annealing was trapped into a local minimum. Whether, the algorithm based on the Particle Swarm Optimization technique kept decreasing until it reached its global minimum after 500 iterations. So, it is better to use the algorithm based on Particle Swarm Optimization for a small number of populations.

However, in the second case study, it can be seen from figure 6.6 that if we need a fast performance, it is better to use Particle Swarm Optimization technique but it will only give a local minimum solution. To have a global minimum solution, it is better to use Simulated Annealing technique. However, the performance of the algorithm would be very slow.

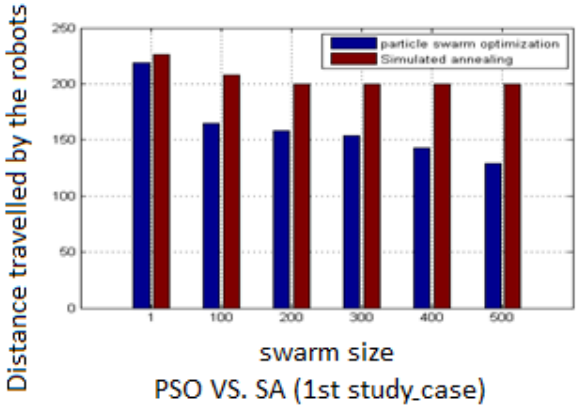


Figure 6.5: PSO vs. SA performance for the first case study

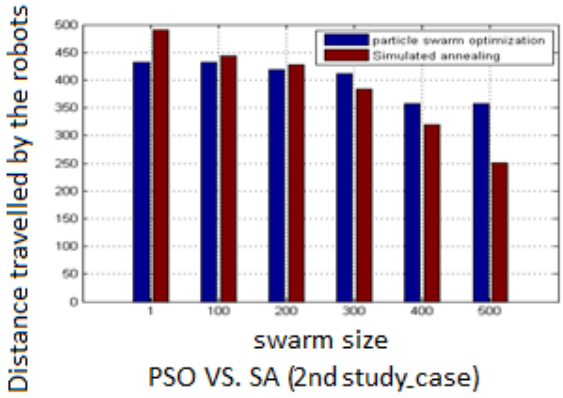


Figure 6.6: PSO VS. SA performance for the second case study

To sum up, it can be observed by looking at figure 6.5 and 6.6 that it is always recommended to use the PSO technique than the SA technique for a swarm with a small number of robots and tasks (1st case study). However, if the swarm size is large (2nd case study), techniques'

preference depends on the number of iterations. Thus, it is a compromise between a global optimal solution that takes long time to be detected, and a good but not optimum solution that could be found faster.

6.4.2 Comparison between the proposed hybrid algorithm vs. other techniques

Since the main objective of this work is to design a hybrid discretized algorithm combines both Particle Swarm Optimization and Simulated Annealing techniques, it is essential to compare the performance of the proposed algorithm with the other traditional algorithms. The proposed hybrid DPSO/SA, DPSO, SA, and the Hungarian algorithms are implemented and executed to test a suite of fifteen different swarm dimensions problems to see the effect of dimensionality on the algorithm's performance.

For fair comparison, the problem instances are selected to be generated randomly for all different problem dimensions. Also, all the simulations were carried out on the same platform (MATLAB 2016, Windows PC with Intel core i7 processor and 8GB RAM). In addition, each technique runs 50 times and each one of them is allowed to run for the same number of iterations in each specific problem dimension, especially the Simulated Annealing and the hybrid DPSO/SA techniques. Table 6.5 presents the outcomes acquired from applying each algorithm on the various swarm dimensions. Each problem instance is executed for 50 times and the average is evaluated.

In table 6.5, the comparison is made between the four algorithms; Hungarian, DPSO/SA, DPSO, SA. The convergence time and the fitness values (distance travelled by robots in the swarm) are evaluated in two columns for each one of these algorithms. From table 6.5, it is obvious that, as expected, the proposed hybrid algorithm outperforms both the DPSO and SA algorithms in almost all cases. Also, the convergence time for the proposed algorithm is always better than that of the discrete PSO algorithm. However, it is clear that Simulated Annealing algorithm gives better execution time than both the hybrid and the Discrete Particle Swarm Optimization algorithm.

Table 6.5: Simulation results

Robots	Tasks	Search Space	Hungarian		Hybrid		DPSO		SA	
			Cost (cm)	Time (sec)	Cost (cm)	Time (sec)	Cost (cm)	Time (sec)	Cost (cm)	Time (sec)
7	10	100	127.84	0.34	127.84	1.3	127.84	1.3	152.05	1.74
15	20	100	213.83	0.39	213.83	12.02	215.43	14.6	293.12	2.2
30	45	135	172	0.08	300	19.3	313	50	358	3.5
40	60	180	231	0.03	544	25.8	708	52	602	5.3
50	75	225	293	0.05	669	29.9	940	55	678	5.2
60	90	270	353.6	0.05	858	44.3	1882	55	977	7
70	105	315	414.3	0.04	1210	57.1	2866	60	1441	7
80	120	360	478.1	0.06	1362	69.7	3906	70	1990	7
90	135	405	541.6	0.07	1757	83.6	4952	90	2591	7
100	150	450	603.9	0.9	2438	105.2	6891	110	2766	7
500	1000	3000	2697	90	55368	182	659377	177	90328	24
600	900	2700	3581	120	63715	185.1	317352	193	82715	31.9
800	1200	3600	4794	330.7	125948	102.6	1064833	235	159669	34.5
1000	1500	4500	5921	520	223204	130.9	2138035	298	305243	37
2000	3000	9000	11819	1509	1191864	341	8722822	347	1351566	42

In this research, since the Hungarian algorithm gives us the exact solution for the problem, it is used as a model to enable us to compare between the different algorithms. To make the results more readable, the outcomes from all the algorithms are normalized, by dividing each algorithms' cost values in all dimensions by the Hungarian algorithm cost values.

Then, the various swarm dimensions are classified into three categories in order to simplify the comparisons between the algorithms, see table 6.6. In each category, we evaluate the geometrical mean of the normalized fitness values and the convergence time for each one of the optimization techniques.

Therefore, table 6.6 lists the time complexity and the fitness values for each algorithm. It has 8 columns corresponding to the fitness values and the time complexity for Hungarian, hybrid DPSO/SA, DPSO, and SA, respectively. The best result for each instance is highlighted using boldface. Also, the geometrical mean for each category is listed.

Table 6.6: Geometrical mean of different swarm sizes

Robots	Tasks	Search Space	Hungarian		Hybrid		DPSO		SA	
			Cost (cm)	Time (sec)	Cost (cm)	Time (sec)	Cost (cm)	Tim (sec)	Cost (cm)	Time (sec)
7	10	100	127.84	0.34	**1**	1.3	**1**	1.3	1.2	1.74
15	20	100	213.83	0.39	**1**	12.02	**1**	14.6	1.4	2.2
30	45	135	172	0.08	**1.7**	19.3	1.8	50	2.1	3.5
40	60	180	231	0.03	**2.4**	25.8	3.1	52	2.6	5.3
50	75	225	293	0.05	**2.3**	29.9	3.2	55	**2.3**	5.2
Small swarm size geometrical mean					**1.56**	11.8	1.78	19.3	1.84	3.3
60	90	270	353.6	0.05	**2.4**	44.3	5.3	55	2.8	7
70	105	315	414.3	0.04	**2.9**	57.1	6.9	60	3.5	7
80	120	360	478.1	0.06	**2.9**	69.7	8.2	70	4.2	7
90	135	405	541.6	0.07	**3.2**	83.6	9.1	90	4.8	7
100	150	450	603.9	0.9	**4**	105.2	11.4	110	4.6	7
Medium swarm size geometrical mean					**3**	68.9	7.9	74.4	3.9	7
500	1000	3000	2697	90	**20.5**	182	244.5	177	33.5	24
600	900	2700	3581	120	**17.8**	185.1	88.6	193	23.1	31.9
800	1200	3600	4794	330.7	**26.3**	102.6	222.1	235	33.3	34.5
1000	1500	4500	5921	520	**37.7**	130.9	361.1	298	51.6	37
2000	3000	9000	11819	1509	**100.8**	341	738	347	114.4	42
Large swarm size geometrical mean					**32.5**	148	264	242	43.3	33.3
Geometrical mean					**5.3675**	49.4	15.5	70.4	6.7763	9.1

Thus, as can be seen for the small swarm sizes, the hybrid approach gives a better performance than both the Simulated Annealing and DPSO algorithms. Also, the geometrical mean for the Discrete Particle Swarm Optimization which equals 1.78 cm, is less than that of the Simulated Annealing which gives geometrical mean of 1.84 cm.

In addition, for the medium swarm sizes, it is obvious that the hybrid approach gives a geometrical mean of 3 which outperforms both the DPSO and Simulated Annealing outcomes. Also, it is important to notice in this case that Simulated Annealing performs better than Discrete Particle Swarm Optimization since the geometrical means for both SA and DPSO are 3.9 and 7.9, respectively.

Furthermore, for the large swarm sizes, the geometrical mean for the proposed hybrid algorithm is 32.5 cm, which is also better than the geometrical mean for both SA and DPSO. Also, the table indicates that, in this case the result of Simulated Annealing technique which is 43.3 is a way better than the result of the Discrete Particle Swarm Optimization which is 264.

Finally, if we were to evaluate and compare all the algorithms in general for all the swarm sizes, it is also very clear from the last row in table 6.6, that the proposed hybrid approach outperforms all the other algorithms. We can see that, the geometrical means for the hybrid DPSO/SA, Discrete Particle Swarm Optimization and Simulated Annealing are 5.4, 15.5 and 6.8, respectively.

To sum up, we could say that it is always recommended to use the hybrid DPSO/SA approach than the other two algorithms for almost all problem dimensions. However to be fair, we should say that the Simulated Annealing execution time is better than the execution time of the proposed hybrid approach.

Also, it could be noticed that regardless of how bad the Discrete Particle Swarm Optimization performs when compared with the hybrid DPSO/SA and the Simulated Annealing approaches, it is worth to mention that it performs better than the Simulated Annealing algorithm in case of small swarm sizes. However, it never gives better results than the proposed hybrid approach even in small problem dimensions.

Finally yet importantly, we could say that in most of the cases, the hybrid DPSO/SA provides the best outcome followed by Simulated Annealing algorithm; whereas the Discrete Particle Swarm Optimization performed rather inadequately.

Actually, these results are somehow identical to the earlier observations by Loiola [81], which denotes that for the combinatorial discrete dynamic task allocation problems, the local searching techniques like the Simulated Annealing algorithm, which could be considered as algorithms with solution modification techniques, are more efficient, powerful and are likely to provide better results than algorithms with population-based approaches, such as the Particle Swarm

Optimization and other social behavior meta-heuristic techniques. This conclusion can lead us to refer to the superiority of our proposed memetic algorithm.

6.4.2.1 Performance analysis

The standard deviations for all the algorithms are also evaluated in order to measure the stability of the outcomes of the hybrid DPSO/SA, PSO and SA approaches, see tables 6.7, 6.8 and 6.9.

6.4.2.1.1 SA

Table 6.7 shows the arithmetic mean, geometrical mean, and more important the standard deviation after running the SA for 100 times.

Table 6.7: SA standard deviation

Swarm Size	Number of tasks	Search Space	Number of iterations	CPU time	Arithmetic Mean	Geometrical Mean	Standard Deviation
7	10	100	100	1.7	81.8	81.7	1.7889
15	20	100	200	2.2	206.4	204.1	33.5
30	45	135	300	3.5	673	668.5	87.8
40	60	180	400	5.3	1241	1236	122.1
50	75	225	400	5.2	2278	2277	242
60	90	270	400	7	3415	3405	308.1
70	105	315	500	7	5176	5171	261.6
80	120	360	500	7	6262	6261	156.5
90	135	405	2000	7	8553	8547	402
100	150	450	2500	7	10680	10660	597.5
500	750	2250	8000	24	48283	48283	2422
600	900	2700	9000	31.9	56999	56976	1807.1
800	1200	3600	10000	34.5	107640	107610	3333.8
1000	1500	4500	12000	37	168068	168000	5726.3
2000	3000	9000	15000	42	152605	152530	5727

It can be shown from the table that the standard deviation for the algorithm increases as the problem dimension increases. This conclusion actually makes sense because the mean values increase as the problem dimension increases and as a result the standard deviation increases too.

For example, when the swarm size equals 7 (small swarm size), it can be seen that the mean equals 81 and the standard deviation is 1.8. However, when the problem dimension increases to 2000 robots, the mean increases to 152605 and as a result, the standard deviation increases to 5727.

6.4.2.1.2 DPSO

Table 6.8 shows the arithmetic mean, geometrical mean, and more important the standard deviation after running the Discrete Particle Swarm Optimization for 100 times. It can also be shown from the table that the standard deviation for the algorithm increases as the problem dimension increases. For example, when the number of robots equals 7 (small swarm size), it can be seen that the mean equals 109 and the standard deviation is 17.9. However, when the problem dimension increases to 2000 robots (large swarm size), the mean increases to 878576 and as a result, the standard deviation increases to 2903.3.

Table 6.8: DPSO standard deviation

Swarm Size	Number of tasks	Search Space	CPU time	Arithmetic Mean	Geometrical Mean	Standard Deviation
7	10	100	1.3	109.1	107.8	17.8789
15	20	100	14.6	179.4000	178.2286	20.8231
30	45	135	50	362.3	360.2547	40.4531
40	60	180	52	531.9	528.4047	50.9358
50	75	225	55	2707.7	2706.2651	78.8278
60	90	270	55	3956.6	3947.7605	86.5927
70	105	315	60	6229.2	6223.9	97.2949
80	120	360	70	7419.8	7416.7	105.9184
90	135	405	90	9757.3	9752.4	157.9694
100	150	450	110	12090.8	12078.7	244.4212
500	750	2250	177	54693	54665	1648.6
600	900	2700	193	74418	74418	2781.3
800	1200	3600	235	136590	136580	2367
1000	1500	4500	298	218123	218120	2859
2000	3000	9000	347	878576	878580	2903.3

6.4.2.1.3 DPSO/SA

Similarly, table 6.9 shows the arithmetic mean, geometrical mean, and more important the standard deviation after running the proposed hybrid DPSO/SA algorithm for 100 times. It can be shown from the table that the standard deviation for the algorithm increases as the problem dimension increases. For example, when the number of robots equals 7 (small swarm size), it can be seen that the mean equals 80 and the standard deviation is 2.3. However, when the problem dimension increases to 2000 robots (large swarm size), the mean increases to 150590 and as a result, the standard deviation increases to 3339 too.

Table 6.9: DPSO/SA standard deviation

Swarm Size	Number of tasks	Search Space	Number of iteration	CPU time	Arithmetic Mean	Geometrical Mean	Standard Deviation
7	10	100	300	1.3	80.3	80.3	2.3
15	20	100	400	12.02	179.6	179.1	16.7
30	45	135	500	19.3	380.5	380.1	21.9
40	60	180	800	25.8	556.5	556.2	23.3
50	75	225	1000	29.9	726.5	725.1	61.5
60	90	270	1200	44.3	1028	1026	79.1
70	105	315	1500	57.1	1245	1243	76
80	120	360	1800	69.7	1430	1427	93.9
90	135	405	2000	83.6	1710	1707	130.8
100	150	450	2500	105.2	1941	1927	287.5
500	750	2250	8000	182	34879	34877	452.5
600	900	2700	9000	185.1	45746	45744	454.6
800	1200	3600	10000	102.6	97583	97579	1168
1000	1500	4500	12000	130.9	149520	149510	1990
2000	3000	9000	13000	341	150587	150590	3339

In figure 6.7, the X-axis represents the problem's dimension and the Y-axis represents the standard deviation. In addition, the blue bars in this figure indicate the outcomes of the hybrid DPSO/SA algorithm, the green bars represent the results out of applying the Discrete Particle

Swarm Optimization technique, and finally the red bars denote the standard deviation of the Simulated Annealing algorithm.

Therefore, by comparing the standard deviations of these three algorithms together, it can be shown that the standard deviation of the hybrid DPSO/SA algorithm is better than that of both the Discrete Particle Swarm Optimization and the Simulated Annealing techniques for almost all various problem dimensions. It is also obvious that the Discrete Particle Swarm Optimization gives smaller standard deviation than Simulated Annealing technique despite that the SA algorithm performs better than the DPSO especially in the large swarm sizes. This actually proves that the Discrete Particle Swarm Optimization technique always suffers from getting stuck into a local optimum solution but it seems more stable than the Simulated Annealing algorithm.

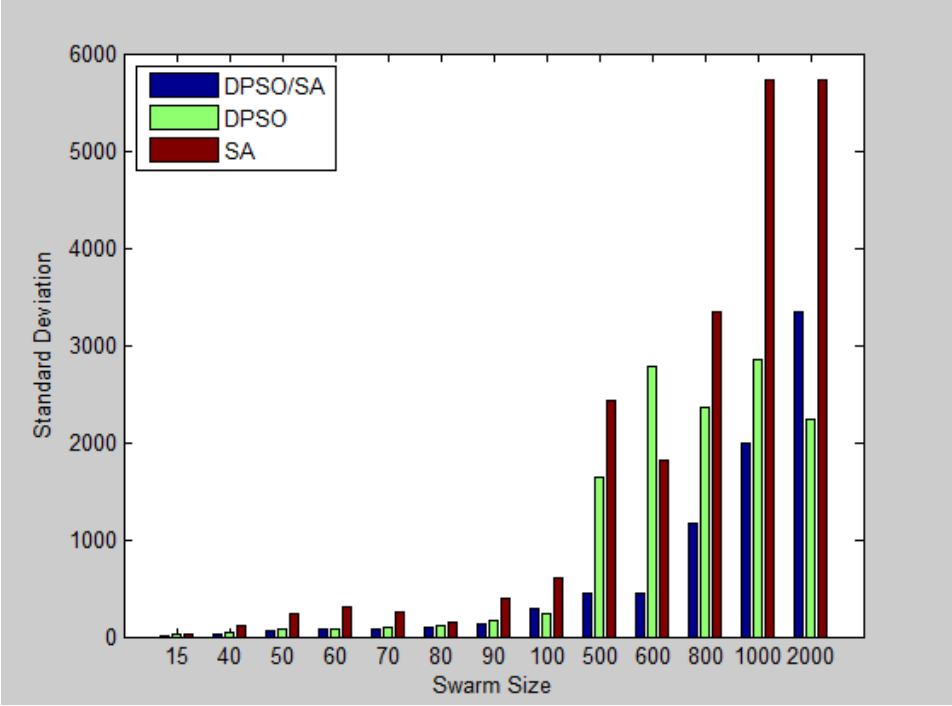


Figure 6.7: Standard deviation of the DPSO/SA, SA, and DPSO.

Thus, the previous remarks indicate that the proposed algorithm is more stable than the traditional Discrete Particle Swarm Optimization and Simulated Annealing algorithm.

6.4.2.2 Scalability analysis

Table 6.5 and 6.6 show the outputs from running the simulations for the DPSO, SA, PSO/SA and the Hungarian algorithm for the various 15 dimensions of the dynamic task allocation problem. It can be clearly deduced that the proposed hybrid technique obtains higher performance than the other classical DPSO and SA in almost all problem sizes.

The DPSO gives a better quality of solutions than the SA annealing in lower dimensional problems. However, in both medium swarm sizes which ranges from 50 to 100 problem dimensions and large swarm sizes which ranges from 500 to 2000 problem dimensions, it could be found that SA produces better performance than the DPSO as the Discrete Particle Swarm Optimization suffers from the disability of jumping out of local optimal solution in large swarm sizes. These results prove that the hybrid algorithm overcomes the disadvantages of DPSO, which suffers from the local minima and the SA, which suffers from poor convergence characteristics. In addition, we can conclude that the proposed algorithm is more scalable than both DPSO and SA algorithms.

6.4.2.3 Stability analysis

In order to compare between the proposed algorithms in different swarms' dimensions, it is better to run each experiment for a number of times and then compare the results to obtain more stable and robust performances. The average value after the 100 runs and the standard deviation are calculated. As can be seen from equation (6-2), this equation is used to calculate the average in which the outputs from a finite number of runs are summed up and then divided by the number of runs, which equals to 100 in our case. As shown in equation (6-3) the difference between the solutions obtained from different runs is calculated which is called the variance. A small value of variance indicates that the algorithm is more stable and the differences between solutions from different runs are small. On the contrary, a large value of variance, gives an indication of the instability of the algorithm, which in turns indicates that it could be extremely hard to implement this algorithm in real-systems.

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Equation (6-2)

where

xi: represents solution produced after an entire run

n: represents the number of runs

and,

$\bar{x}$: represents the average

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Equation (6-3)

where

σ: represents the standard deviation

Table 6.7, 6.8 and 6.9 show the statistics of the standard deviation values using the 3 algorithms while testing 15 various swarm dimensions. In order to prove that a specific algorithm is good enough to be used in a real-life system, the standard deviation should be taken into account along with the algorithm performance and the solutions' average values. The lower the standard deviation is, the more stability the algorithm provides.

As can be seen from these tables, the proposed PSO/SA algorithm almost always gives smaller standard deviation values than both the traditional DPSO and SA in almost all dimensions which means that the hybrid PSO/SA algorithm is more stable than the traditional DPSO and SA algorithms.

However, the comparison between the DPSO and SA shows that, at small and medium swarm sizes, the standard deviation for both SA and DPSO is comparable. On the contrary, in the large swarm sizes, the DPSO algorithm has smaller standard deviation than the SA algorithm. The reason behind the outstanding performance and stability of the proposed hybrid algorithm is that the SA allows the PSO/SA to globally explore for better solutions and avoid getting stuck into a local optimum solution.

6.4.2.4 Convergence analysis

Based on the simulation results shown in previous sections, the convergence characteristics of the DPSO is usually better than that of the SA. Also, it is clearly obvious that the quality of solution for the SA is relatively higher than the DPSO especially for large swarm dimensions. The proposed PSO/SA algorithm enhances both the searching capabilities over the global search space, which in turns prevents local optimum solutions, and increases the efficiency of finding the solution in lower convergence time.

6.4.2.5 Solution efficiency

It is fair enough to select the same stopping condition while comparing the efficiencies for a number of algorithms in finding a good quality solution. In this work, two conditions are chosen. If any of these two conditions are met, the algorithm stops and this specific point could be considered as a convergence state. These two conditions are either reaching the maximum number of iteration or the constancy of the solution for a certain number of iterations, which indicates that the algorithm saturates and the solution stays unchanged. However, as indicated from table 6.6, different number of iterations are selected for different problem dimensions, because the algorithm takes time to saturate in larger problem dimensions.

Also, it can be clearly deduced that the mean values of the solutions of the hybrid algorithm are also better than both the DPSO and the SA results. The reason behind this is that almost all solutions found by the PSO/SA algorithm are near the global minimum solution. Thus, we assume that the proposed hybrid PSO/SA algorithm is more efficient than both the DPSO and SA alone.

6.5 Simulation using Eliza-3 robots

We implemented two experimental scenarios with real robots using the proposed hybrid algorithm. All of the experiment scenarios contain 4 Eliza-3 robots divided into two teams, a searching team (red team) and a rescue team (blue team). The searching team inspects the arena to find victims and then the rescue team transports them to the designated rescue zone.

In the first scenario, only one robot is employed as a searching robot and the other 3 robots are applied to be used as rescuers (see figure 6.12). While, in the second scenario, two robots are used as searching robots and the other two robots are used as rescuers (see figure 6.21). We used the RGB LED in the robots to distinguish between the two different teams. Red color is used to denote the searching team while the blue color is dedicated as a reference to the rescue team.

As can be shown in figure 6.16, if one of the robots from the searching team finds a victim, its color should be turned into green and pause for a second then turns back to red color and starts searching randomly again for more victims (see figure 6.17). In addition, if one of the rescuers allocates itself to transport a specific victim, it starts moving towards its allocated victim. When the robot reaches to the victim's location, the 8 green LEDs around its body should be turned on and the robot starts transporting the victim to the rescue zone, (see figure 6.18 and 6.19). After that, the rescuer turns into the sleep mode again until the searching team finds more victims and so on.

6.5.1 The controller

In this thesis, we propose a distributed algorithm that could efficiently tackle the problem of victims' allocation in a search and rescue robotic system and then transport victims to the rescue zone. A distributed algorithm is a promising choice for such an application that might suffer from lots of failures while running since the robots are most probably navigating in an unstable and unstructured arena. A set of essential questions regarding the control level are commonly showed up if a distributed system is decided to be designed. Most of these questions have been already answered throughout chapter 3 and 4. These questions include:

1. How the robots interact with each other?

2. How could we control distributed systems?
3. How should tasks be allocated to the available robots?

The desired behavior of our proposed controller is to locate and transport the victims to the rescue zone. The proposed distributed controller is designed to utilize only local sensing and communication capabilities. The control logic executed independently by each Eliza-3 robot is shown in figure 6.8. The searching robots inspect the environment for victims by performing a random walk with their red LED illuminated (Searching state). As indicated before, when one of the searching robots detects a victim, it turns on the green led, pauses for a second and continue searching for other victims in the environment. While, the closest rescuer to the victim moves towards the victim to grip it (Rescuing state) and turns on the green LEDs around the robot's body. Then, the rescuer starts transporting its allocated victim to the rescue zone.

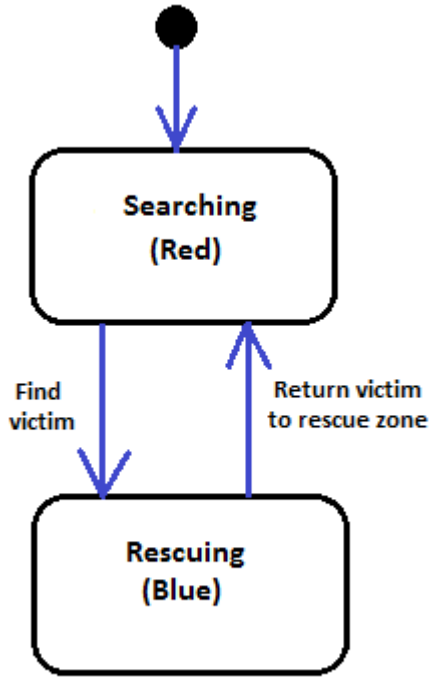


Figure 6.8: Controller's conceptual state diagram.

6.5.2 Elisa-3 robot

Eliza-3 robots are smart swarm robots with a very high cognition abilities inspired from swarm of birds when they transport objects to their nest. Each individual robot owns central RGB LED, 8 green LEDs around the robot's body, IR emitters, 8 IR proximity sensors, 4 ground sensors, 3-axis accelerometer, RF radio for communication (Nordic Semiconductor nRF24L01+), micro USB connector for programming, debugging and charging, IR receiver, 2 DC motors, top light diffuser, and selector. Eliza-3 robot is designed with capability of communicating with the other Eliza-3 robots, navigating smoothly in the environment using 2 differential wheels, and providing monitoring information for data analysis on an external computer through communicating with the base station.

Thus, our system consists of small, simple, but fully autonomous individuals that have the ability of sensing, making decisions, and acting in the environment. They work individually as well as collaborate with other units. There is not any centralized planning and control. The global task execution is realized through robot-robot and robot-environment interactions.

6.5.3 Experiment implementation

As mentioned before, each experiment scenario is defined by the number of rescue robots, and the number of searching robots. Both the searching and the rescue robots have no prior knowledge of the experiment configuration. In this research, the two experiments are implemented to show that our proposed hybrid DPSO/SA algorithm provides sufficient performance and displays robustness and adaptability characteristics.

6.5.3.1 First scenario

We conducted this scenario using three rescue robots and one searching robot. In this experiment, the system successfully allocated the nearest rescue robot to the found victim and the victim is transported to the destination rescue zone. In this scenario, possible incorrect behaviors could be allocating more than one rescue robot to the same victim or allocating the far robot to the victim instead of the nearest ones. However, in this experiment, nothing wrong

happened and the nearest rescuer allocated itself successfully to the inspected victim in the environment.

In this experimental scenario, all robots start from the same starting point to have the same reference in the arena. Figure 6.9 shows that the first rescue robot starts from our reference point (0,0). Then, the second robot starts from the same reference point (see figure 6.10) followed by the third and fourth robots (see figure 6.11 and 6.12).

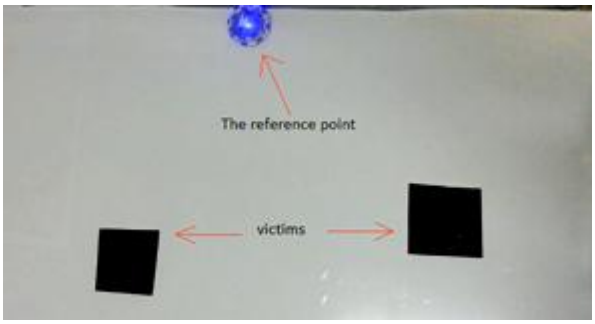


Figure 6.9: Starting 1st robot from the reference point (0,0)

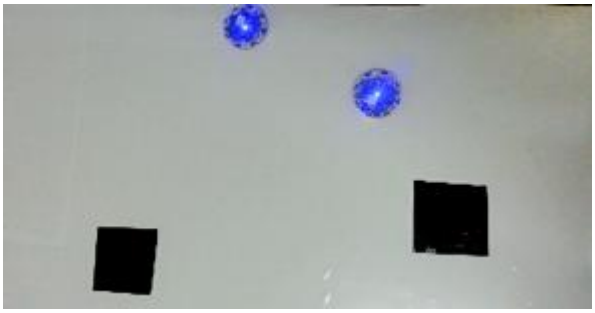


Figure 6.10: Starting the 2nd robot from the reference point (0,0)

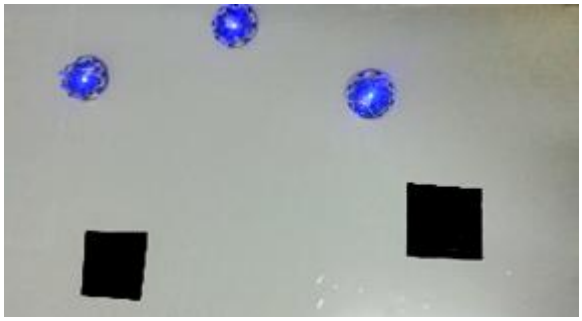


Figure 6.11: Starting the 3rd robot from the reference point (0,0)

Figure 6.12: Starting the 4th robot from the reference point (0,0)

Then, as shown from figure 6.13, 6.14, and 6.15, the red searching robot starts searching for victims in the arena. Once, the searching robot finds a victim as can be seen from figure 6.16, it turns on its green led, stops for a second and then turns into the red color again and keeps moving to search for other victims (see figure 6.17). The victims in our experiment is represented as small black sheets to be easily detected by the robots' ground sensors as the ground sensors in the bottom side of the robots could differentiate between different colors efficiently. It reads values from 512 to 1023 and the smaller the value the darker the surface. Finally, as can be shown from figure 6.18, the closest rescuer to the detected victim starts moving towards it and turns on the 8 green LEDs whenever it reaches to the victim's location.

After it pauses for a second, it continues moving to the rescue zone to leave the victim there (see figure 6.19).

Figure 6.13: The searching robot searches for victims

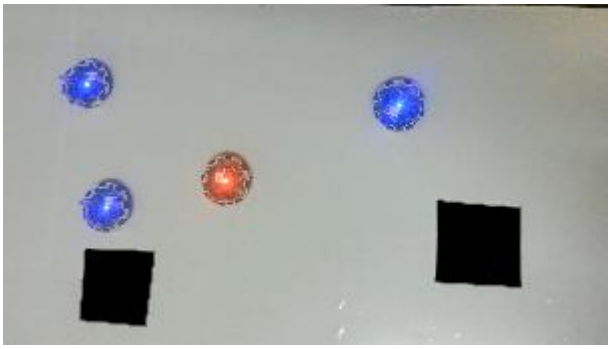


Figure 6.14: Searching for victims

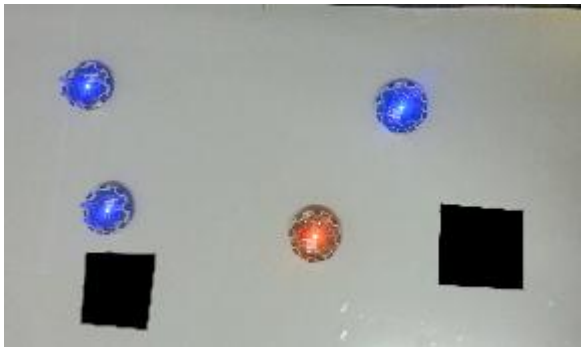


Figure 6.15: The rescue robot searches for victims

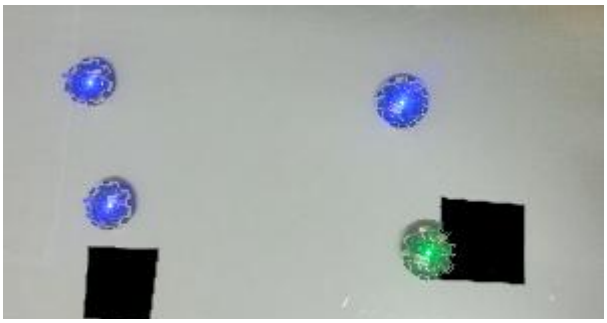


Figure 6.16: The rescue robot finds a victim

Figure 6.17: The rescue robot searches for more victims

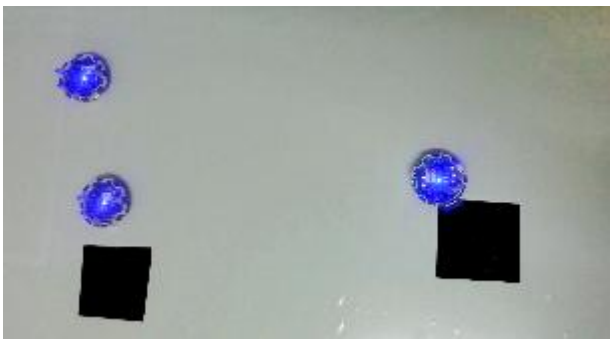


Figure 6.18: The closest rescuer to the victim finds the victim

Figure 6.19: The rescuer transports the victim to the rescue zone

6.5.3.1.1 First scenario results

In this scenario, we have 1 searching robots and 3 rescuers. The cost of allocating the nearest rescuer to the explored victim is 10cm (correct allocation). However, if we considered the wrong allocations by allocating any of the other two far robots to the victim, they would cost us either 29cm or 34cm. Note that the cost here is the distance between the robot and the victim. By looking carefully to these results, we can see that the wrong allocation costs around 3 times more than the cost of the right allocation. Since, our experiment works properly and the correct allocation achieved, it costs us only 10cm to allocate the robot to the victim. In addition, it is worth to mention that the execution time for the algorithm was 0.15 sec given that we only have 4 robots in our system.

6.5.3.2 Second scenario

We conducted the second scenario using two rescue robots and two searching robots. In this experiment, the system successfully allocated the nearest rescue robot to each of the founded victims and the victims were transported to the nest. Same as in the first experiment, possible incorrect behaviors in this experiment could be allocating the two rescue robots to the same victim or allocating a far robot to a given victim instead the nearest one. In addition, in this experiment, everything works properly and nothing went wrong, and the rescuers allocated themselves in the right way to the detected victims in the arena.

Unlike the implementation of the first scenario, in this scenario only the rescue robots start from a single starting point to have the same reference point in the arena. However, the searching robots started from other points on the right and left sides of our reference point. Hence, we added a small value to the x-axis of the starting point of the searching robots. Figure 6.20 shows that the first rescue robot starts from our reference point (0,0). Then, the second robot starts from the same reference point (see figure 6.21). As shown in figure 6.22, the rescue robots turned into the sleep mode at two different random locations waiting for victims to be found by the searching team.

Then, as can be seen from figure 6.23, the red searching robots start searching for victims in the arena. Once, the searching robot on the right side finds a victim (see figure 6.24), it turns on its green led, stops for a second and then turns on the red color again and keeps moving to search for other victims (see figure 6.26).

Similarly, when the other searching robot on the left side finds a victim, it turns on its green led, stops for a second and then turns on the red led again and continues searching for other victims (see figure 6.25 and 6.26).

After that, as can be shown from figure 6.27, the rescuers start moving towards the closest victims and turn on the 8 green LEDs whenever they reach to the victims' location. Therefore, the rescuer on the right side moves to the right victim and the rescuer on the left moves towards the left victim. Finally, the robots pause for a second on the victims' locations and then continue moving to the rescue zone to leave the victims there (see figure 6.28).

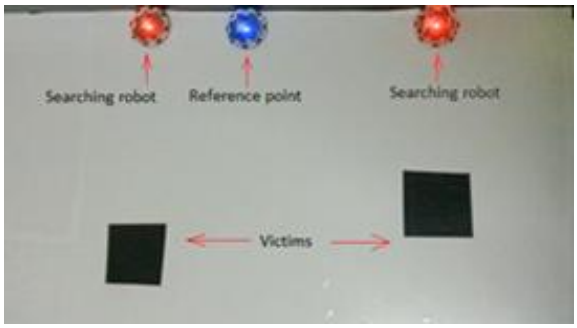


Figure 6.20: The first rescuer starts from the reference point (0,0)

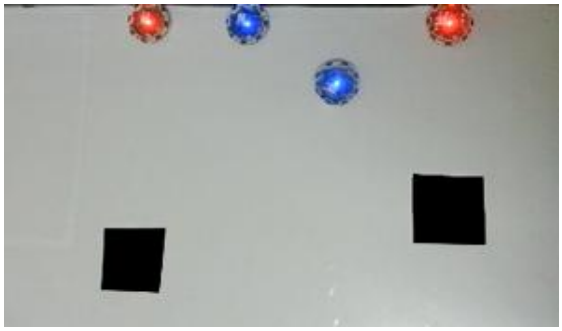


Figure 6.21: The second rescuer starts from the reference point (0,0)

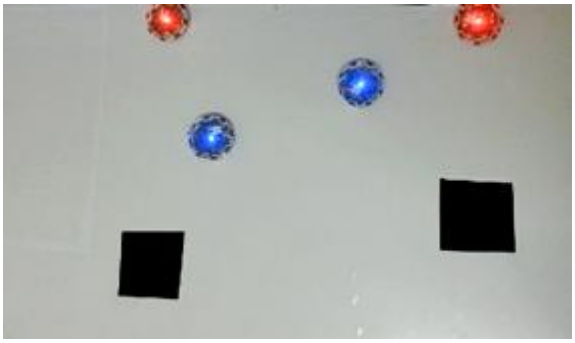


Figure 6.22: The two rescuer turns into sleep mode at random location

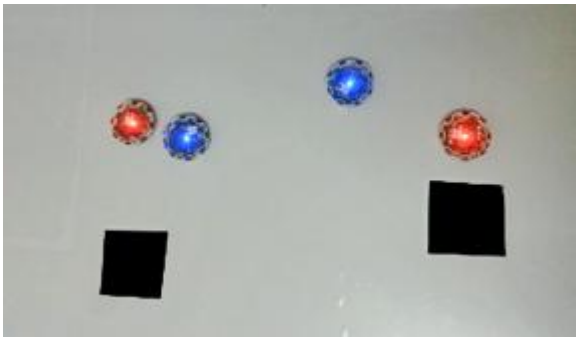


Figure 6.23: The two searching robots start searching for victims randomly

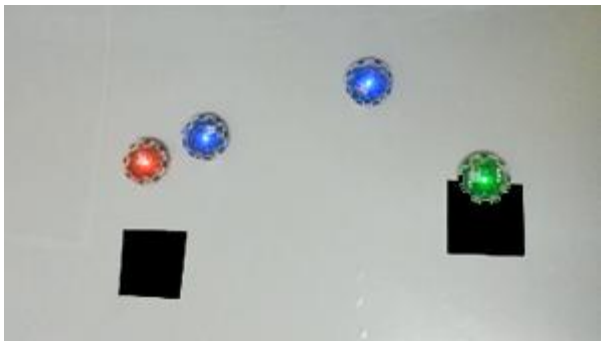


Figure 6.24: The right searching robot finds a victim

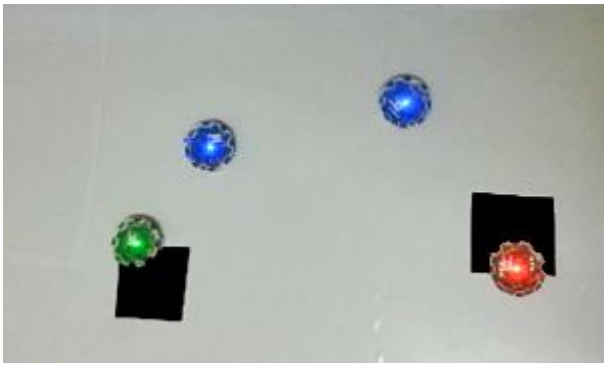


Figure 6.25: The left searching robot finds a victim

Figure 6.26: The searching robot continue searching for more victims

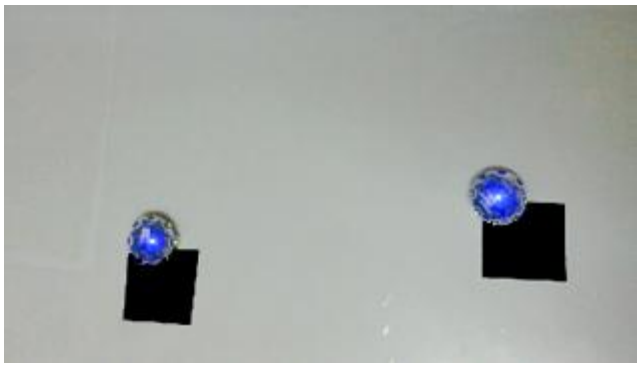


Figure 6.27: The tow rescuers find the victims detected by the searching team

Figure 6.28: The rescue robots transport the victims to the rescue zone

6.5.3.2.1 Second scenario results

In this scenario, we have 2 searching robots and 2 rescuers. The cost of allocating the right rescuer to the right victim is 8cm (correct allocation), while the cost of allocating the right rescuer to the left victim is 24cm (wrong allocation). In addition, the cost of allocating the left rescuer to the left victim is 12cm (correct allocation), while the cost of allocating the left rescuer to the right victim is 25cm (wrong allocation). Therefore, the correct allocation demands a total cost of (8+12=20cm). However, the wrong allocation requires an overall cost of around (24+25=49cm). By considering this small experiment, we can see that the cost of the wrong allocation is almost more than twice the cost of the correct allocation. Since, our experiment

works properly and the valid allocation achieved, it costs us only 20cm to allocate the robots to the victims. In addition, it is worth to mention that similar to the first scenario, the execution time for the algorithm was 0.18 sec given that we only have 4 robots in our system.

6.5.4 Communication overhead evaluation

In this research, each robot is equipped with a transceiver module, suitable for wireless communication with ultra-low power consumption in the communication range of Radio Frequency (RF). The chip is designed to operate in the ISM radio band (Industrial, Scientific and Medical band basis) of 2.4 GHz. Thus, the robots communicate with each other using RF via a base station connected to a computer via USB. Likewise, the transceiver module, embedded in the robot, communicates via SPI (Serial Peripheral Interface) with the microcontroller on the robot's board and transfers data to and from the PC via wireless communication, as shown in figure 6.29.

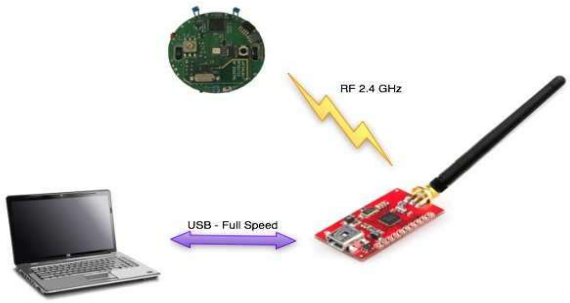


Figure 6.29.Communication between the robots and PC.

Each robot is identified by a unique address in the swarm. This address is stored in a specific memory address of the robot EEPROM. Every message coming from the base station has a destination address, which should coincide with one of the addresses of the robot in the swarm.

Up on reception of a message, the robot compares the destination address of the received message with its own address to assess whether it is intended by this message. If so, the message is saved and interpreted by the robot.

The master PC, which continuously polls the slave base station, controls the communication between the robots and the base station. The polling is performed on regular basis, which is a restriction on the maximum speed of communication. To overcome this limitation, an optimized protocol was implemented wherein the packet sent from the PC to the base station contains commands to four robots simultaneously. The base station is responsible of separating the received packet into four individual packets of 16 bytes each, before sending them to the indicated destination address. The same procedure is followed during the reception from the robots. In this case, the base station is responsible for receiving the packets of 4 robots, assembles them into a single message of 64 bytes, and sends this message to the computer. This procedure allows higher throughput communication, making it 4 times faster.

The packets generated by the robots to be transferred to the base station are composed of 16 bytes, as shown in figure 6.30(a). The first byte is used to identify the validity of the packet, the following two bytes represent the sender address of the robot, and the fourth byte represents the message type. Up to 255 different types of messages may exist during the communication process. The remaining 12 bytes contain the payload of the message, which are the actual data to be sent.

The structure of the message sent by the base station to a robot is shown in figure 6.30(b). It is similar to the structure of the message shown in figure 6.30(a), it also consists of 16 bytes. The first byte defines the validity, which validates the authenticity of the message sent by the base station when it is received by the robot. Byte 14 defines the type of message, used when there are more than one type of messages, and the last two bytes contain the destination address of the robot. The remaining 12 bytes contain the payload of the message to be received by the robot.

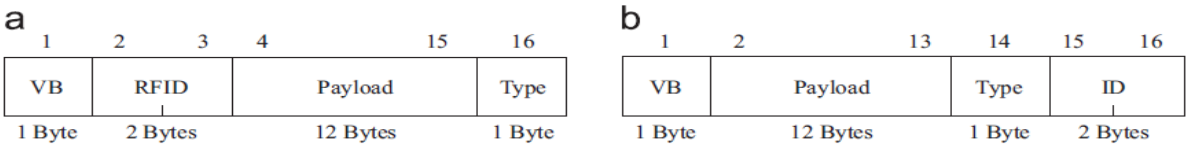


Figure 6.30. Overall structure of the message sent by robot ID to the RF base station. (a) From robot to RF base station. (b) From RF base station to robot.

Packet format - PC to radio to robot

The 16 bytes packet format is shown below in figure 6.31 (the number in the parenthesis expresses the bytes):

Command (1)	Red led (1)	Blue led (1)	Green led (1)	IR + Flags (1)	Right motor (1)	Left motor (1)	Small green leds (1)	Flags2 (1)	Remaining 5 bytes are unused

Figure 6.31: Structure of the message sent by the RF base station to robot ID

Packet format - robot to radio to PC

The robot sends back to the base-station information about all its sensors every time it receives a command; this is accomplished by using the "ack packet" feature of the radio module. Each "ack" is 16 bytes length as shown in figure 6.32 and is marked with an ID that is used to know which information the robot is currently transferring.

ID=3 (1)	Prox0 (2)	Prox1 (2)	Prox2 (2)	Prox3 (2)	Prox5 (2)	Prox6 (2)	Prox7 (2)	Flags (1)
ID=4 (1)	Prox4 (2)	Ground0 (2)	Ground1 (2)	Ground2 (2)	Ground3 (2)	AccX (2)	AccY (2)	TV remote (1)
ID=5 (1)	ProxAmbient0 (2)	ProxAmbient1 (2)	ProxAmbient2 (2)	ProxAmbient3 (2)	ProxAmbient5 (2)	ProxAmbient6 (2)	ProxAmbient7 (2)	Selector (1)
ID=6 (1)	ProxAmbient4 (2)	GroundAmbient0 (2)	GroundAmbient1 (2)	GroundAmbient2 (2)	GroundAmbient3 (2)	AccZ (2)	Battery (2)	Free (1)
ID=7 (1)	LeftSteps (4)	RightSteps (4)	theta (2)	xpos (2)	ypos (2)	Free (1)		

Figure 6.32: Structure of the message sent by robot ID to the RF base station.

Considering each packet received from the base station, the PC takes the four addresses and the allocated tasks' information and composes a new package of 16 bytes. The first byte of this

package is the validity flag of the package to be verified by the robot upon reception, the 12 following bytes represent the address and the allocated tasks' information received, the fourteenth byte is a sending signal enabler and the remaining two bytes contain the address of the target robot. This package is replicated four times using four distinct robot addresses to compose the message of 64 bytes to be sent to the base station to be transmitted to the destination addresses. Thus, the same information (robot ID and task) is transmitted to four robots at a time until all the robots of the swarm are encompassed.

The enabling signal generated and sent by the PC to robots of the swarm has the function of managing the process of packet transmission from the robots to the PC. This signal enables the transmission to a group of four robots to transmit at a time. Note that the transmission time for a group of 4 robots depends on the total number of robots that compose the swarm. This synchronization trick is an alternative to reduce the packet traffic, thus decreasing the chances of packages being lost due to limited buffer size at the base station.

In the developed algorithm, the payload of the message generated by the robot and sent to the base station consists solely of a byte that corresponds to the task currently allocated to this robot. In order to compose the necessary information to be sent to the remaining robots of the swarm, a robot includes its RFID. Thus, the information to be transferred to the robots has three bytes: two bytes to identify the robot and one byte to identify the allocated task. The packet composition during the communication between the robots is shown in figure 6.33.

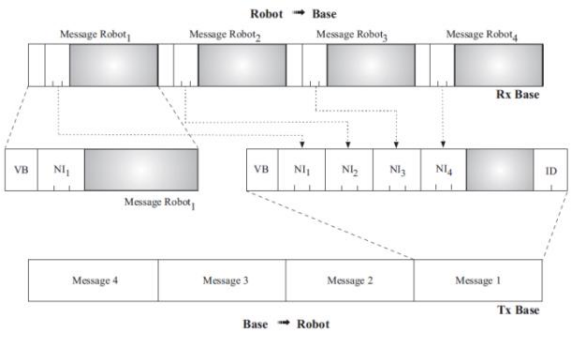


Figure 6.33: Packet format of the messages from and to the robots.

When the base station detects the existence of new messages in the buffer, the packet is formed of 64 bytes each, containing four messages. This packet is sent to a computer that interprets the message contents, identifying the payload of each of the composed messages. Then, the task informed in the payload of each of the four messages received and the two bytes identifying the robot are extracted to form a new message to be sent a way to the robots. Thus, the new generated message contains as a payload, the necessary information (NI) included in the four messages received by the base station. The content of the message thus composed is then replicated into four new messages, each one destined to a different target address. The four messages generated form a packet to be routed back to the base station by computer. Upon reception, the base station separates its content into four messages and sends them away to the identified robots.

6.5.4.1 Transmission and reception overhead

Since our swarm system depends on a wireless communication, we have to calculate the communication overheads in both the transmitter and receiver sides. In this project the transmitter is the Atmel 2560 microcontroller connected with the RF radio for communication (Nordic Semiconductor nRF24L01+), while the receiver is the base station which is connected to the pc through USB connection.

This system is developed using Atmel 2560 microcontroller to which a wireless transceiver called Nrf24L01 is connected. Atmel 2560 microcontroller is programmed via the AVR studio. The nrf24l01 is connected to the microcontroller and communicates with it via SPI pins, namely MOSI, MISO, SCK and 2 GPIOs for the ce and csn pins. Those are pins necessary for the correct operation of the nrf24l01. Libraries called SPI.h, nRF24L01.h, and mirf.h are used to handle the initializations and configurations of the registers in the nrf24l01 and the SPI related parameters in the system. In addition, SPI.c, nRF24L01.c, and mirf.c are dedicated to handle the implementation for these modules.

6.5.4.2 Real time analysis

This wireless system needs to be a hard real-time system, because the robots need to transmit messages containing their coordinates, all their internal states and sensors values. Then, each

robot receives the same message from the other robots. Since the robots continuously changing their positions and hence all their internal states change, so they have to send and receive messages in frequent basis.

Moreover, the robots must send their messages within a specific period. It has to be deterministic and meet all of its deadlines as unexpected behavior or delay of the transmission during the mission cannot be tolerated. The reason why the deadlines are hard is that failure to meet just one deadline would mean the delay or failure of the transmission of robot's information including the proximity IR and ground sensors, which may cause the robot to fall from a cliff, collide with obstacles, or even has wrong information about the other robots' coordinates. Therefore, missing deadlines cannot be tolerated because it may cause the failure of our entire system and hence, the hard real-time constraints are essential.

6.5.4.3 Transmission and reception instructions

Since the Atmel microcontroller runs at a clock = 8 MHz, therefore each instruction takes 1 / (8* 10^6), which is equal to 125 ns.

As mentioned above, the libraries used in this program are SPI.h, nRF24L01.h, and mirf.h to support the communication between the Atmel microcontroller and the nrf24l01. The SPI.h library sets the SPI clock by default to SYSCLK/4. This means that the SPI_CLOCK now runs at 8/4 = 2 MHz and hence is able to transmit one bit every 0.5 us. In our system, we send the message byte by byte. This means that one byte could be transmitted over SPI within a total time of 4 us. Table 6.10 and 6.11 show a list of the tasks in the system with the detailed instructions and timings of each one.

6.5.4.4 Transmission side analysis

In the transmission side, the distance from the Atmel microcontroller on the robot's board to the radio frequency module is not large, therefore the delay introduced from the wires connecting them is negligible.

The default and most reliable air data rate for the nrf24l01 is 2 Mbps. So, to transmit the 8 bits of the char, it would take 8/(2*10^6) = 4 us.

Table 6.10 shows the essential transmission side instructions and their timings to calculate the total communication overhead in the transmission side.

Table 6.10: Transmission communication overhead

Instruction	Time
Check for the data readiness	125 ns
16 commands for updating Ackpayload	2 us
Sending command (write payload)	125 ns
Delay due to transmission of a char between the microcontroller and the nrf24L01 via SPI.	4 us
Air data rate	4 us
Total	10.25 us

6.5.4.5 Receiver side analysis

The usb_receive(RX_buffer, 16) instruction in the receiver side checks whether the RF channel contains data ready to be read. If data is available, the nrf24l01, if in RX mode, can read it.

Table 6.11 shows the essential receiver side instructions and their timings to calculate the total communication overhead in the receiver side.

Table 6.11: Reception communication overhead

Instruction	Time
Check if there is data to be read	125 ns
Read the receiving data	125 ns
8 commands for updating sensor variables	1 us
Delay due to the reception of a char between the microcontroller and the nrf24L01 via SPI.	4 us
Air data rate	4 us
Total	9.25 us

Therefore, it can be concluded that the system's total worst communication overhead taken to transmit data from the robot and receive it in the base station = 9.25 us + 10.25 us = 19.5 us.

This time considers sending just one byte since the SPI sends byte by byte. However, the base station could receive up to 4 messages at the same time in which each message consists of 16 bytes. Thus, total communication time for sending 4 messages from 4 different robots and receiving them at the base station simultaneously = 16 * 4 * 19.5 us = 1248 us = 1.25 ms.

Since we have two ways communication, so same calculation should be applied for sending the data from the base station and receiving it via 4 different robots simultaneously. Therefore, The overall communication overhead = 2* 1.25 ms = 2.5 ms.

Finally, it is worth noting that the assumption that each instruction on the Atmel microcontroller takes 125 ns is not very accurate. This is because the instructions we write on the AVR studio represent more than one instruction in assembly, hence one instruction can be in fact split into perhaps 2 or three instructions which in turn each takes 125 ns, so the delay would be larger than the one which is calculated. However, since the microcontroller runs at a clock as fast as 8 MHz, the time taken to execute one instructions is quite small, which means that even if the actual time per instruction was four or five times more, the increase would still be in the order of nanoseconds so the overall system would still meet its deadline.

To sum up, the entire communication overheads including the transmission and reception on the robots and the base station happen within 2.5ms. Since, each message from a single robot is 16 bytes, thus the throughput is 46 bytes each 2.5 millisecond.

Chapter 7

Conclusion and Future Work

In this chapter, both the conclusion about the work that is done in this research and the future work that could be carried out to extend and enhance our research in the future are addressed.

7.1 Conclusion

In this thesis, a mathematical model for the dynamic task allocation problem is formulated for a swarm of homogeneous robots using linear integer programming. The task allocation decision process is carried out independently by each robot in the swarm using a clear and straightforward procedure to accomplish the whole tasks in a minimal time.

In addition, an efficient and applicable framework based on a probabilistic velocity representation is implemented to discretize the Particle Swarm Optimization technique in order to find solution to the dynamic task allocation problem, which is a discrete combinatorial problem. All the substantial efforts and improvements that have been discovered through years of working on developing the continuous version of the Particle Swarm Optimization technique are retained and discrete PSO just builds the new framework over these original efforts.

In addition, two case-studies with different problem dimensions are simulated to compare between a local searching technique (Simulated Annealing technique) and a global searching technique with parallel population based structure (Particle Swarm Optimization technique). The comparison is done in terms of both the quality of solution and the time complexity of the algorithm. Basically, we did this simulation to highlights the pros and cons of both PSO and SA and hence develop a more efficient, robust and scalable algorithm.

This simulation experiments prove that the Particle Swarm Optimization has a better convergence characteristics due to its parallel structure. However, it suffers from the issue of getting trapped into a local optimal solution and it loses the ability to reach the global optimal solution.

On the other hand, the results also show that the Simulated Annealing algorithm provides us with a better-quality solution due to the use of the metropolis mechanism, which determines whether to accept some worse solutions or not using a specific probability function. This mechanism enables the algorithm to jump out of any local optimal solution towards the global optimal solution and hence finds better quality solution. However, it is worth to mention that this algorithm suffers from bad convergence characteristics.

Thus, a novel flexible and simple hybrid algorithm combining both DPSO and SA optimization technique is proposed to find solutions to the cons of both DPSO and SA. Actually, this algorithm provides us with superior quality solutions with small and stable convergence time because it utilizes the probabilistic jumping property of the SA to escape from local minimum solutions, and to raise the particle's diversity. Thus, the DPSO/SA algorithm stretches the search scope of the swarm to new unexplored areas, and promotes its global searching ability.

In the novel hybrid DPSO/SA algorithm, every particle gains experience from both its own best solution and the global best solution. The inertial weigh and the acceleration constants c1 and c2 are tuned to control both the cognitive and social learning process via enhancing the performance of the proposed algorithm. Also, a restart approach is provided to solve the stagnation problem in the proposed algorithm.

Moreover, simulation scenarios are extended to a suite of 15 different swarm dimensions in order to confirm the efficiency of the proposed algorithm through comparing it with both SA and DPSO. In these extended simulation instances, the Hungarian algorithm which provides us with the exact solutions is used as a model to normalize the outcomes from all the three optimization techniques. Also, the algorithms' results are classified into 3 different categories, small swarm size, medium swarm size and large swarm size to observe the algorithms' behavior in different swarm sizes.

The simulation results provide an outstanding performance for the hybrid algorithm in almost all problem dimensions. It significantly outperforms both the traditional Particle Swarm Optimization technique and the Simulated Annealing algorithm due to its simplicity, stability, ruggedness, high quality solutions, raised computational efficiency and stable convergence

characteristics. It is also worth to mention that, the proposed algorithm performance is closely followed by the SA algorithm, which provides promising execution time results.

Moreover, it is pertinent to mention that the local searching approaches most probably provide a better-quality solution than the global meta-heuristic techniques in the combinatorial discrete problems. However, combining both the solution constructive techniques and the local searching techniques as a single unit is proven to be very effective and is found to be more successful compared with the traditional well-known local searching techniques.

Finally, we implement the proposed hybrid algorithm in a real system using 4 Eliza-3 robots. After a chain of experiments' runs, the system achieves a stable performance and performs all tasks efficiently.

Also, these experimental scenarios are mapped into a real application (human rescuing application) which could be applied in a semi-destroyed site after a disaster like an earthquake happened. In such situation, it is dangerous for human to get inside this site in an attempt to rescue victims. So, we studied a scenario in which we could deploy a team of autonomous robots to rescue those victims without the need for human operators' intervention.

Finally yet importantly, the communication overhead is evaluated in both transmission side and receiver side and the throughput of the system is calculated to be 46 bytes each 2.5 millisecond.

7.2 Future Work

The focus in this research is upon the problems, which include only single-robot tasks. Therefore, we could extend this work by taking into consideration two-robot tasks or more. Actually, tightly coupled tasks add some sort of complexity to the problem whenever it is taken into account. For example, it is more challenging to find a suitable allocation for tasks that need the cooperation of at least two robots in order to carry out the task concurrently. The resulting complexity is due to the environmental temporal and spatial constraints. Also, in this case, the task assignment complexity rises exponentially with growing the number of task's types. So, developing efficient approaches to deal with such complex problem could be a remarkable contribution in the future.

In addition, due to our limited budget, we implement our algorithm using only a set of homogeneous robots called Elisa-3 robots. So, these experiments could be extended in the future to include heterogeneous robots, which of course would increase the complexity of the problem. In this case, the mathematical model of the problem needs to be reformulated and the optimization technique should be modified to suite the new problem's formulation and constrains.

Also, robots with more sophisticated sensing capabilities like camera, laser and ultrasonic could be used in the future. For example, e-puck robot could be involved which owns microphone, camera, ultrasonic, optical sensor and accelerometer. All these different kind of sensors could be utilized to raise the accuracy of the odometry data through applying the concept of sensor diffusion.

Finally, robots' congestion could be taken into account in the future in order to help the robots to avoid jammed locations in the environment and hence increase the efficiency and robustness of the algorithm.

References

[1]. Agarwal. T., “History of Robotics, Types and Latest Applicatios List of Robots,” ElProCus - Electronic Projects for Engineering Students, 29-Mar-2016. [Online]. Available: https://www.elprocus.com/robots-types-applications/. [Accessed: 25-May-2017].

[2]. N.A.S.A. Administrator, “The ATHLETE Rover,” NASA, 29-Oct-2015. [Online]. Available : https : // www. nasa.gov/ multimedia/ imagegallery/ image_feature_748.html. [Accessed: 30-Jul-2017].

[3]. Jones, J., "Robots at the tipping point: the road to iRobot Roomba", *IEEE Robotics & Automation Magazine* 13 (1) (2006) 76-78.

[4]. Ranch. R., “Are You a Tesla Fan?,” Democratic Underground. [Online]. Available: https://www.democraticunderground.com/1002901 9840. [Accessed: 25-May-2017].

[5]. Atherton, K. D., "unmanned ground vehicles", *Popular Science*, 2017. [Online]. Available: http://www.popsci.com/tags/unmanned-ground-vehicles. [Accessed: 20- Aug-2017].

[6]. Kimball, D.G., “Arms Control Today,” U.S. Prepares to Arm Italian Drones | Arms Control Association. [Online]. Available: https://www.armscontrol.org/ACT/2015_12/News-Briefs/US-Prepares-to-Arm-Italian-Drones. [Accessed: 30-Jul-2017].

[7]. Navarro, I. and Matía, F., "An introduction to swarm robotics," ISRN Robotics (2013) 1–10.

[8]. Zhang, Y. and Liu, S., "Survey of multi-robot task allocation," CAAI Transactions on Intelligent Systems 3 (2) (2008) 115–120.

[9]. Stormont, D.P., "Autonomous rescue robot swarms for first responders", *Proceedings of the 2005 IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety*, Orlando, FL, USA, 2005, pp. 151-157.

[10]. Grady, R.O., Pinciroli, C., Grob, R., Christensen, A., Mondada, F., Bonani, M. and Darigo, M., "Swarm-bots to the rescue", *Advances in Artificial Life, Darwin Meets von Neumann*, Berlin Heidelberg, 2011, pp. 165-172.

[11]. Jennings, J.S., Whelan, G. and Evans, W.F., "Cooperative search and rescue with a team of mobile robots." *Advanced Robotics, 1997. ICAR'97. Proceedings., 8th International Conference on*. IEEE, Monterey, CA, USA, USA, 1997, pp. 193-200.

[12]. Keshmiri, S. and Payandeh, S., "Multi-robot, dynamic task allocation: a case study", Intelligent Service Robotics 6 (3) (2013) 137-154.

[13]. Brutschy, A., Pini, G., Pinciroli, C., Birattari, M. and Dorigo, M., "Self-organized task allocation to sequentially interdependent tasks in swarm robotics", Autonomous Agents and Multi-Agent Systems 28 (1) (2012) 101-125.

[14]. Liu, C. and Kroll, A., "Mimetic algorithms for optimal task allocation in multi-robot systems for inspection problems with cooperative tasks", Soft Computing 19 (3) (2014) 567-584.

[15]. Nedjah, N., Mendonça, R. and Mourelle, L., "PSO-based Distributed Algorithm for Dynamic Task Allocation in a Robotic Swarm", *Procedia Computer Science*, Vol. 51, 2015, pp. 326-335.

[16]. Mathias de Mendonça, R., Nedjah, N. and de Macedo Mourelle, L., "Efficient distributed algorithm of dynamic task assignment for swarm robotics", Neurocomputing 172 (2016) 345-355.

[17]. Liu, L., Michael, N. and Shell, D., "Communication constrained task allocation with optimized local task swaps", Autonomous Robots 39 (3) (2015) 429-444.

[18]. Liu, H., Zhang, P., Hu, B. and Moore, P., "A novel approach to task assignment in a cooperative multi-agent design system", Applied Intelligence 43 (1) (2015) 162-175.

[19]. Shieh, H., Kuo, C. and Chiang, C., "Modified particle swarm optimization algorithm with simulated annealing behavior and its numerical verification", Applied Mathematics and Computation 218 (8) (2011) 4365-4383.

[20]. Tang, F. and Parker, L.E., "Automated synthesis of multi-robot task solutions through software reconfiguration", *in IEEE International Conference, Robotics and Automation and ICRA*, Barcelona, Spain, 2005, pp. 1501-1508.

[21]. Guerrero, J. and Oliver, G., "Multi-robot task allocation strategies using auction-like mechanisms", in Artificial Intelligence Research and Development (2003) 111-122.

[22]. Zhang, K., Collins, E. and Shi, D., "Centralized and distributed task allocation in multi-robot teams via a stochastic clustering auction", ACM Transactions on Autonomous and Adaptive Systems 7 (2) (2012) 1-22.

[23]. Nagarajan, T. and Thondiyath, A., "An algorithm for cooperative task allocation in scalable, constrained multiple robot systems", Intelligent Service Robotics 7 (4) (2014) 221-233.

[24]. Krieger, M., Billeter, J. and Keller, L., "Ant-like task allocation and recruitment in cooperative robots", Nature 406 (2000) 992-995.

[25]. Momen, S. and Sharkey, A. J.C., "An ant-like task allocation model for a swarm of heterogeneous robots", in Swarm Intelligence Algorithms and Applications Symposium (2009) 31-38.

[26]. Tsalatsanis, A., Yalcin, A. and Valavanis, K.P., "Dynamic Task Allocation in Cooperative Robot Teams", International Journal of Advanced Robotic Systems (2009) 1.

[27]. Gigliotta, O., Mirolli, M. and Nolfi, S., "Communication basesd dynamic role allocation in a group of homogeneous robots", Natural Computing 13 (3) (2014) 391-402.

[28]. Zhang, D., Xie, G., Yu, J. and Wang, L., "Adaptive task assignment for multiple mobile robots via swarm intelligence approach", Robotics and Autonomous Systems 55 (7) (2007) 572-588.

[29]. Gerkey, B., "A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems", The International Journal of Robotics Research 23 (9) (2004) 939-954.

[30]. Campo, A. and Dorigo, M., "Efficient multi-foraging in swarm robotics", *in Advances in artificial*, vol 4648. Springer, Berlin, Heidelberg, 2007, pp. 696-705.

[31]. Dasgupta, P., "Multi-Robot Task Allocation for Performing Cooperative Foraging Tasks in an Initially Unknown Environment", in Innovations in Defence Support Systems -2, Springer, Berlin Heidelberg (2011) 5-20.

[32]. Guha, S. and Khuller, S., "Greedy Strikes Back: Improved Facility Location Algorithms," Journal of Algorithms 31 (1) (1999) 228–248.

[33]. Karunakaran, D., Mei, Y., Chen, G. and Zhang, M., “Evolving dispatching rules for dynamic Job shop scheduling with uncertain processing times”, IEEE Congress on Evolutionary Computation (CEC) (2017).

[34]. Hoffman, K.L., Padberg, M. and Rinaldi, G., "Traveling salesman problem", Encyclopedia of operations research and management science , Springer US (2013) 1573-1578.

[35]. Östergård, P.R., “A fast algorithm for the maximum clique problem,” Discrete Applied Mathematics 120 (1-3) (2002) 197–207.

[36]. Faroe, O., Pisinger, D. and Zachariasen, M., “Guided Local Search for the Three-Dimensional Bin-Packing Problem”, INFORMS Journal on Computing 15 (3) (2003) 267–283.

[37]. Talbi, E.G. and Bessière, P., “A parallel genetic algorithm for the graph partitioning problem,” *Proceedings of the 5th international conference on Supercomputing - ICS*, New York, NY, USA, 1991, pp. 312-320.

[38]. Benlic, U. and Hao, J., "Breakout local search for the quadratic assignment problem", Applied Mathematics and Computation 219 (9) (2013) 4800-4815.

[39]. Benlic, U. and Hao, J., "Memetic search for the quadratic assignment problem", Expert Systems with Applications 42 (1) (2015) 584-595.

[40]. Mangat, V., "Survey on particle swarm optimization based clustering analysis", Swarm and evolutionary computation, Springer Berlin Heidelberg 7269 (2012) 301-309.

[41]. Kulkarni, R. and Venayagamoorthy, G., "Particle Swarm Optimization in Wireless-Sensor Networks: A Brief Survey", IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 41 (2) (2011) 262-267.

[42]. AlRashidi, M.R. and El-Hawary, M.E., "A survey of particle swarm optimization applications in electric power systems", IEEE Transactions on Evolutionary Computation 13 (4) (2009) 913-918.

[43]. Hafiz, F. and Abdennour, A., "Particle Swarm Algorithm variants for the Quadratic Assignment Problems - A probabilistic learning approach", Expert Systems with Applications 44 (2016) 413-431.

[44]. Menhas, M., Wang, L., Fei, M. and Ma, C., "Coordinated controller tuning of a boiler turbine unit with new binary particle swarm optimization algorithm", International Journal of Automation and Computing 8 (2) (2011) 185-192.

[45]. Shen, M., Zhan, Z., Chen, W., Gong, Y., Zhang, J. and Li, Y., "Bi-Velocity Discrete Particle Swarm Optimization and Its Application to Multicast Routing Problem in Communication Networks", IEEE Transactions on Industrial Electronics 61 (12) (2014) 7141-7151.

[46]. Wang, L., Wang, X., Fu, J. and Zhen, L., "A Novel Probability Binary Particle Swarm Optimization Algorithm and Its Application", Journal of Software 3 (9) (2008).

[47]. Liu, B., Wang, L. and Jin, Y., "An Effective PSO-Based Memetic Algorithm for Flow Shop Scheduling", IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics) 37 (1) (2007) 18-27.

[48]. Hu, X., Eberhart, R.C. and Shi, Y., "Swarm intelligence for permutation optimization: a case study of n-queens problem", *In Swarm intelligence symposium, SIS'03, Proceedings of the 2003 IEEE*, Indianapolis, IN, USA, USA, 2003, pp. 243-246.

[49]. Correa, E.S., Freitas, A.A. and Johnson, C.G., "A new discrete particle swarm algorithm applied to attribute selection in a bioinformatics data set", *Proceedings of the 8th annual conference on Genetic and evolutionary computation, ACM*, New York, NY, USA, 2006, pp. 35-42.

[50]. Neethling, M. and Engelbrecht, A.P., "Determining RNA secondary structure using set-based particle swarm optimization", Evolutionary Computation, CEC, Congress on. IEEE (2006) 1670-1677.

[51]. Clerc, M., "Discrete particle swarm optimization, illustrated by the traveling salesman problem", New optimization techniques in engineering, Springer (2004) 219-239.

[52]. Langeveld, J. and Engelbrecht, A.P., "A generic set-based particle swarm optimization algorithm." *International conference on swarm intelligence, ICSI*, Cergy, France, 2011, pp. 1-10.

[53]. Langeveld, J. and Engelbrecht, A., "Set-based particle swarm optimization applied to the multidimensional knapsack problem", Swarm Intelligence 6 (4) (2012) 297-342.

[54]. Wang, D. and Liu, L., "Hybrid particle swarm optimization for solving resource-constrained FMS", Progress in Natural Science 18 (9) (2008) 1179-1183.

[55]. Wang, L. and Zheng, D., "An effective hybrid optimization strategy for job-shop scheduling problems", Computers & Operations Research 28 (6) (2001) 585-596.

[56]. Hu, X., "PSO Tutorial," Particle Swarm Optimization: Tutorial. [Online]. Available: http://www. swarmintelligence.org/tutorials.php. [Accessed: 14- Mar- 2017].

[57]. Gao, Y., Du, W. and Yan, G., "Selectively-informed particle swarm optimization", Scientific article (2015). [Online]. Available: https://www.nature.com/articles/srep09295. [Accessed: 21- Aug- 2017].

[58]. Geltman, K.E., "Simulated Annealing,". [Online]. Available: http://katrinaeg.com/ simulated-annealing.html. [Accessed: 14- Mar- 2017].

[59]. Carr. R., "Simulated Annealing," . [Online]. Available: http://mathworld.wolfram.com/ SimulatedAnnealing.html. [Accessed: 14- Mar- 2017].

[60]. Busetti, F., "Simulated annealing overview". [Online]. Available: http://www. aiinfinance. com/saweb.pdf. [Accessed: 21- Aug- 2017].

[61]. Deepa, S., Nagarajan, D. and Palanikumar, R., "Solving Assignment Problem using MATLAB", International Journal of Research Instinct 3 (2) (2016) 396-401.

[62]. Kuhn, H., "Variants of the hungarian method for assignment problems", Naval Research Logistics Quarterly 3 (4) (1956) 253-258.

[63]. Kuhn, H., "The Hungarian method for the assignment problem", Naval Research Logistics Quarterly 2 (1-2) (1955) 83-97.

[64]. Munkres, J., "Algorithms for the Assignment and Transportation Problems", Journal of the Society for Industrial and Applied Mathematics 5 (1) (1957) 32-38.

[65]. Newton, D., "Learning Big O Notation With O(n) Complexity - DZone Performance", dzone.com (2017). [Online]. Available: https://dzone.com/articles/learning-big-o-notation-with-on-complexity. [Accessed: 21-Aug- 2017].

[66]. Mills-Tettey, G.A., Stentz, A. and Dias, M.B., "The dynamic hungarian algorithm for the assignment problem with changing costs", (2007).

[67]. GCtronics, “Elisa-3,” [Online]. Available: http://www.gctronic. com/doc/index. php/Elisa-3. [Accessed: 11- Jan- 2017].

[68]. Ghadle, K.P. and Muley, Y.M., "Travelling salesman problem with MATLAB programming ", International Journal in Advance Applied Mathematics and Mechanics (2) (3) (2015) 258 - 266.

[69]. SINGH, S., "A Comparative Analysis of Assignment Problem", IOSR Journal of Engineering 2 (8) (2012) 01-15.

[70]. Eberhart, R. and Kennedy, J., "A new optimizer using particle swarm theory", *in Proceedings of the Sixth International Symposium on Micro Machine and Human Science* , Nagoya, Japan, 1995, pp. 39-43.

[71]. Kennedy, J. and Eberhart, R.C., "A discrete binary version of the particle swarm algorithm", Systems, Man, and Cybernetics, *Computational Cybernetics and Simulation, IEEE International Conference*, Vol. 5, Orlando, FL, USA, 1997, pp. 4104-4108.

[72]. Pang, W., Wang, K.P., Zhou, C.G. and Dong, L.J., "Fuzzy discrete particle swarm optimization for solving traveling salesman problem." *Computer and Information Technology, the Fourth International Conference on IEEE*, Wuhan, China, 2004, pp. 796-800.

[73]. Liu, H.A. and Clerc, M., "An hybrid fuzzy variable neighborhood particle swarm optimization algorithm for solving quadratic assignment problems", Journal of Universal Computer Science 13 (9) (2007) 1309-1331.

[74]. Liu, H.A. and Zhang, J., "A particle swarm approach to quadratic assignment problems", Soft computing in industrial applications (2007) 213-222.

[75]. Hasan, M., "Local search algorithms for the maximal planar layout problem", International Transactions in Operational Research 2 (1) (1995) 89-10.

[76]. Locatelli, M., "Convergence Properties of Simulated Annealing for Continuous Global Optimization", Journal of Applied Probability 33 (4) (1996) 1127.

[77]. Mitra, D., Romeo, F. and Sangiovanni-Vincentelli, A., "Convergence and Finite-Time Behavior of Simulated Annealing", Advances in Applied Probability 18 (3) (1986) 747-771.

[78]. Neshat, M., Sargolzaei, M., Nadjaran Toosi, A. and Masoumi, A., "Hepatitis Disease Diagnosis Using Hybrid Case Based Reasoning and Particle Swarm Optimization", Artificial Intelligence 2012 (2012) 1-6.

[79]. Brownlee. J., "Particle Swarm Optimization," [Online]. Available: http://www.cleveralgorithms.com/nature-inspired/swarm/pso.html. [Accessed: 21- Aug- 2017].

[80]. Meng, Q., Zhang, L. and Fan, Y., "A Hybrid Particle Swarm Optimization Algorithm for Solving Job Shop Scheduling Problems", *in Asian Simulation Conference*, Vol. 644, Springer Singapore, 2016, pp.71–78.

[81]. Loiola, E.M., de Abreu, N.M.M., Boaventura-Netto, P.O., Hahn, P. and Querido, T., "A survey for the quadratic assignment problem", European journal of operational research 176 (2) (2007) 657-690.