University of Piraeus
Department of Informatics

Doctoral Dissertation

Optimized Heuristic Implementations in AI Multi-Agent Systems

By Christos **Drosos**

Supervisor: Professor, Dr. Themistoklis Panayotopoulos

Submitted in partial satisfaction of the requirements
for the degree of **Doctor of Philosophy**

Piraeus, 2016

University of Piraeus
Department of Informatics

Doctoral Dissertation

"Optimized Heuristic Implementations in AI Multi-Agent Systems"

by **Christos Drosos**

Advisory Committee:	Themistoklis Panayotopoulos, Professor, University of Piraeus
	Evangelos Fountas, Professor, University of Piraeus
	George Tsihrintzis, Professor, University of Piraeus

Approved by the seven (7) member committee in …/…/2016

.....	.....	.....
Themistoklis Panayotopoulos,	**Evangelos Fountas,**	**George Tsihrintzis,**
Professor,	Professor,	Professor,
University of Piraeus	University of Piraeus	University of Piraeus

.....	.....
Vrizidis Lazarus,	**Dimitrios Tseles,**
Head - Professor,	Deputy Head - Professor
Technological Institution of Piraeus	Technological Institution of Piraeus

.....	.....
Gregory Hondrokoukis,	**Michalis Papoutsidakis**,
Professor,	Assistant Professor,
University of Piraeus	Technological Institution of Piraeus

Piraeus, 2016

“Intelligence and interaction are deeply and inevitably coupled,

and multi-agent systems reflect this insight.”

- Gerhard Weiss -

ACKNOWLEDGEMENTS

I dedicate this Doctoral Dissertation

To my Family

To my Professors and

To all my friends

SUMMARY

A **central vision** of the *combinatorial optimization* is to deliver a number of independent lines of operational research by implementing algorithms and hyper-graph theory on a series of random theoretical computer science problems. A main motivation is that thousands of real-life problems can be formulated as abstract combinatorial optimization problems. Albeit, **multi-agent systems** can also be used to solve problems that are difficult or impossible for monolithic systems to solve and methodic, functional, procedural, algorithmic and reinforcement forms of approaches are needed. Thus, in practice, extensive research and development efforts are required when applying combinatorial optimization in intelligent multi-agent systems, in order to achieve cooperation, interoperability and sustainability in heterogeneous and complex existing or future industrial, aerospace, robotic systems or/and any other cyber-physical system.

Driven by the above mentioned problems, this *Doctoral Dissertation* **presents** *meta-heuristic approaches* in the area of AI algorithms and specifically the **particle swarm** and **ant colony optimization** to solve *combinatorial optimization problems*. More specifically present thesis' applications/implementations **focus** on enhancing the:

I. Combinational optimization of the intelligent multi-agent systems;
II. Capabilities of the modern and intelligent multi-agent systems;
III. Sustainability of the global cyber-physical systems.

The *methodological approach* followed in this Dissertation for reaching the **first goal** may be described within the following achievements:

- Reformulate the problem of dartboard game within the spectrum of metaheuristic algorithms
- Use/Apply the Ant Colony System and Max-Min Ant System (MMAS) algorithm as a metaheuristic strategy that guide the search process
- Reinforces local search in neighborhood of the best solution found in each iteration.

In addition, **heuristic** is an adjective for experience-based techniques that help in problem solving, learning and discovery and heuristic methods are used to rapidly come to an 'optimal solution'. In more precise terms, heuristics stand for strategies using readily accessible information to control problem solving in many practical scenarios, slowing convergence and facilitating exploration.

However, the *Ant Colony Optimization* technique has emerged recently as a new meta-heuristic for hard combinatorial Optimization Problems. Implementing a randomized construction heuristic extension of *Ant Colony System algorithm in the Vehicle Scheduling Problem* has become the **second goal** of this dissertation which is met by the following:

- Definition the objective function for VSP
- Make probabilistic decisions as a function of artificial pheromone trails
- Estimate the effect of varying of levels of initial pheromone quantities on the objective function using an extension of Ant Colony System (ACS) algorithm

The **final goal** of this thesis is to enhance the previously referred goals/contribution regarding the sustainable development of an application paradigm. All parts included in currently presented thesis must meet the needs of the present forms of computing without compromising the ability of future generations to meet their own needs. This last objective is met at high-level by presenting the application of a ship dispatching real-life problem by:

- Proposing a solution for a ship dispatching problem with the usage of Particle Swarm Optimization algorithm
- Analyze the operational dimension of the problem by introducing a VPR formulation
- Providing the paradigm of a set of 13 ports of the Aegean Sea (including a depot port) that was taken into consideration

Summarizing the thesis outline is consisted of six (6) chapters as described below:

- **Chp.1:** Artificial Intelligence and Agents
- **Chp.2:** Multi-Agent Systems, Taxonomy and Architectures
- **Chp.3:** Graph Searching Methods and Swarm Intelligence
- **Chp.4:** PSO Algorithm Application: Solving a Ship Dispatching Problem
- **Chp.5:** Ant Colony Optimization Application: An extension to the VSP
- **Chp.6:** Comparison of the Meta-heuristic ACS and MMAS Algorithms: An Optimized Dartboard Design Application
- **Chp.7:** Conclusions and Future Directions

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

1. Artificial Intelligence and Agents

AI is a history of fantasies, possibilities, demonstrations, and promise. Ever since Homer wrote of mechanical "tripods" waiting on the gods at dinner, imagined mechanical assistants have been a part of our culture. However, only in the last half century we have been able to build experimental machines that test hypotheses about the mechanisms of thought and intelligent behavior and thereby demonstrate mechanisms that formerly existed only as theoretical possibilities.

-Bruce Buchanan (2005)-

1.1 Introduction

Artificial intelligence (aka AI), is the field that studies and research the analysis of computational agents[1] acting intelligently (what an agent does) within an environment [1]. More specifically, an agent acts intelligently if:

- what it does is appropriate for its given goals;
- it is flexible to modify environments and goals;
- it learns from experience gained;
- it makes optimum choices given its perceptual and computational limitations.

In addition, a computational agent is an agent whose decisions about its actions can be explained in terms of computation. An agent typically cannot observe the state of the world directly; it has only a finite memory and it does not have unlimited time to act. Thus, a decision can be segmented into primitive/primary operation although there are some agents that are arguably not computational (i.e. wind, rain eroding a landscape). The main scientific/research goal of AI is to comprehend in-depth the principles that make intelligent behavior possible in natural or artificial systems by:

- the analysis of natural and artificial agents;
- formulate and test hypotheses schemes of constructing intelligent agents;
- designing, building, and experimenting with environments, agents and tasks.

As part of science, most researchers build empirical systems to test their hypotheses or to explore the remaining space of possibilities. Therefore the engineering goal of AI is to design artifacts; agents that act intelligently and become useful in many application fields and research domains.

[1] Agents include worms, dogs, thermostats, airplanes, robots, humans, companies, and countries.

Nevertheless, it is arguable that in AI you cannot have fake intelligence as it is only the external behavior that defines intelligence. Therefore if and when AI is achieved the real intelligence created is artificially. This idea of AI defined by external behavior was the motivation for a test designed by Turing (1950) (aka Turing test) [2]. The Turing test consists of an imitation game where an interrogator can ask a witness, via a text interface, any question [3]. If the interrogator cannot distinguish the witness from a human, the witness must be intelligent. Table 1 shows the example of a possible Turing Test. An agent that is not really intelligent could not fake intelligence for arbitrary topics

TABLE 1: A possible dialog for the Turing test

Interrogator	In the first line of your sonnet which reads "Shall I compare thee to a summer's day," would not "a spring day" do as well or better?
Witness	It wouldn't scan.
Interrogator	How about "a winter's day," That would scan all right.
Witness	Yes, but nobody wants to be compared to a winter's day.
Interrogator	Would you say Mr. Pickwick reminded you of Christmas?
Witness	In a way.
Interrogator	Yet Christmas is a winter's day, and I do not think Mr. Pickwick would mind the comparison.
Witness	I don't think you're serious. By a winter's day one means a typical winter's day, rather than a special one like Christmas.

The obvious naturally intelligent agent is the human being. One class of intelligent agents that may be more intelligent than humans is the class of organizations. Ant colonies are a prototypical example of organizations. Each individual ant may not be very intelligent, but an ant colony can act more intelligently than any individual ant. The colony can discover food and exploit it very effectively as well as adapt to changing circumstances. Similarly, companies can develop, manufacture, and distribute products where the sum of the skills required is much more than any individual could master. Modern computers, from low-level hardware to high-level software, are more complicated than any human can understand, yet they are manufactured daily by organizations of humans [4]. Human society viewed as an agent is arguably the most intelligent agent known and it is instructive to consider where human intelligence comes from as there are three (3) main sources:

- Biology: Humans have evolved into adaptable animals that can survive in various habitats.
- Culture: Culture provides not only language, but also useful tools, useful concepts, and the wisdom that is passed from parents and teachers to children.
- Life-long learning: Humans learn throughout their life and accumulate knowledge and skills.

These sources interact in complex ways [5]. Biological evolution has provided stages of growth that allow for different learning at different stages of life. Culture interacts strongly with learning. A major part of lifelong learning is what people are taught by parents and teachers. Language, which is part of culture, provides distinctions in the world that should be noticed for learning.

1.2 Defining Agents and Environments

An agent is something that acts in an environment [5][8]. Purposive agents have preferences. They prefer some states of the world to other states, and they act to try to achieve the states they prefer most. The non-purposive agents are grouped together and called nature. If an agent does not have preferences, by definition it does not care what world state it ends up in, and so it does not matter what it does. Agents interact with the environment via a body and an agent system is made up of an agent and its environment. The agent receives stimuli from the environment and carries out actions in the environment. An agent is made up of a body and a controller. The controller receives percepts from the body and sends commands to the body. A body includes sensors that convert stimuli into percepts and actuators that convert commands into actions. Agents are situated in time: they receive sensory data in time and do actions in time. The action that an agent does at a particular time is a function of its inputs. Discrete time has the property that, for all times, except perhaps a last time, there is always a next time. Dense time does not have a "next time." Initially, we assume that time is discrete and goes on forever. A percept trace, or percept stream, is the function that specifies what is observed at each time. A command trace specifies the command for each time point. A percept trace for an agent is thus the sequence of all past, present, and future percepts received by the controller.

A command trace is the sequence of all past, present, and future commands issued by the controller. The commands can be a function of the history of percepts. This gives rise to the concept of a transduction, a function that maps percept traces into command traces. Although a causal transduction is a function of an agent's history, it cannot be directly implemented because an agent does not have direct access to its entire history. It has access only to its current percepts and what it has remembered. The belief state of an agent at time is all of the information the agent has remembered from the previous times. An agent has access only to its history that it has encoded in its belief state. Thus, the belief state encapsulates all of the information about its history that the agent can use for current and future commands. At any time, an agent has access to its belief state and its percepts.

AI is about practical reasoning: a coupling of perception, reasoning, and acting comprise an agent acting in an environment [6][7]. An agent's environment may well include other agents. An agent together with its environment is called a world. Thus, an agent could be, for example, a coupling of a computational engine with physical sensors and actuators, called a robot, where the environment is a physical setting. It could be the coupling of an advice with a human who provides perceptual information and carries out the task. An agent could be a program that acts in a purely computational environment (i.e. a software agent).

FIGURE 1: An agent interacting with an environment

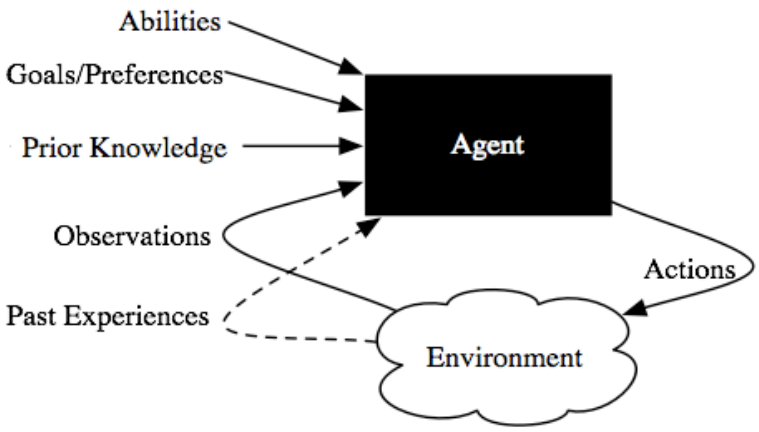


Figure 1 shows the inputs and outputs of an agent but an agent at any time depends on its

- prior knowledge about the agent and the environment;
- history of interaction with the environment, which is composed of
- observations of the current environment and
- past experiences of previous actions and observations, or other data, from which it can learn;
- goals that it must try to achieve or preferences over states of the world;
- abilities, which are the primitive actions it is capable of carrying out.

Two deterministic agents with the same prior knowledge, history, abilities, and goals should do the same thing. Changing any one of these can result in different actions. Each agent has some internal state that can encode beliefs about its environment and itself. It may have goals to achieve, ways to act in the environment to achieve those goals, and various means to modify its beliefs by reasoning, perception, and learning. There are a number of ways an agent's controller can be used:

- An embedded agent is one that is run in the real world, where the actions are carried out in a real domain and where the sensing comes from a domain.

- A simulated agent is one that is run with a simulated body and environment; that is, where a program takes in the commands and returns appropriate percepts. This is often used to debug a controller before it is deployed.
- A agent system model is where there are models of the controller (which may or may not be the actual code), the body, and the environment that can answer questions about how the agent will behave. Such a model can be used to prove properties of agents before they are built, or it can be used to answer hypothetical questions about an agent that may be difficult or dangerous to answer with the real agent.

Each of these is appropriate for different purposes.

- Embedded mode is how the agent must run to be useful.
- A simulated agent is useful to test and debug the controller when many design options must be explored and building the body is expensive or when the environment is dangerous or inaccessible. It also allows us to test the agent under unusual combinations of conditions that may be difficult to arrange in the actual world.
- How good the simulation is depends on how good the model of the environment is. Models always have to abstract some aspect of the world. Appropriate abstraction is important for simulations to be able to tell us whether the agent will work in a real environment.
- A model of the agent, a model of the set of possible environments, and a specification of correct behavior allow us to prove theorems about how the agent will work in such environments. For example, we may want to prove that a robot running a particular controller will always get within a certain distance of the target, that it will never get stuck in mazes, or that it will never crash. Of course, whether what is proved turns out to be true depends on how accurate the models are.
- Given a model of the agent and the environment, some aspects of the agent can be left unspecified and can be adjusted to produce the desired or optimal behavior. This is the general idea behind optimization and planning.

1.3 Knowledge Representation and Hierarchical Control

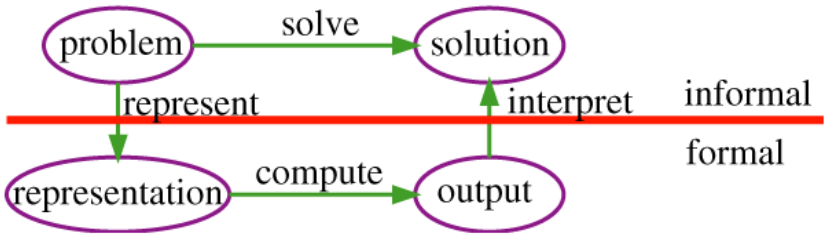
Typically, a problem to solve or a task to carry out is what it constitutes a solution and it is not simple [9][10]. The general framework for solving problems by computer is given in Figure 2 [11]. To solve a problem, the designer of a system must:

- determine what constitutes a solution;
- represent the problem in a language with which a computer can reason;
- compute an output and interpret the output as a solution to the problem.

Knowledge is the information about a domain that is used as part of designing a program to solve the problem of the domain of interest. A representation scheme is the form of how the knowledge is represented, used in an agent while specifying the form of the knowledge base; the representation of all of the knowledge that is stored by an agent [12]. A good representation scheme should be:

- rich enough to express the knowledge needed to solve the problem;
- as close to the problem as possible; it should be compact, natural, and maintainable.
- amenable to efficient computation;
- able to be acquired from people, data and past experiences.

FIGURE 2: The role of representations in solving problems



Many different representation schemes are designed with expressiveness in mind, and then inference and learning are added on. Computers and human minds are examples of physical symbol systems. As a symbol we define the meaningful pattern that can be manipulated as a unit by a symbol system. Much of AI rests on the physical symbol system hypothesis) [13][14][15]: A physical symbol system has the necessary and sufficient means for general intelligent action. This way an agent can use physical symbol systems to model the world. A model of a world is a representation of the specifics of what is true in the world or of the dynamic of the world. The world does not have to be modeled at the most detailed level to be useful. All models are abstractions; they represent only part of the world and leave out many of the details. An agent can have a very simplistic model of the world, or it can have a very detailed model of the world. The level of abstraction provides a partial ordering of abstraction. A lower-level abstraction includes more details than a higher-level abstraction. An agent can have multiple, even contradictory, models of the world. The models are judged not by whether they are correct, but by whether they are useful [16][17].

Choosing an appropriate level of abstraction is difficult [18][19] because:

- a high-level description is easier for a human to specify and understand;
- a low-level description can be more accurate and more predictive;
- the lower the level, the more difficult it is to reason with;
- you may not know the information needed for a low-level description.

Both biological systems, and computers, can be described at multiple levels of abstraction (levels for both biological and computational entities):

- The knowledge level is a level of abstraction that considers what an agent knows and believes and what its goals are. The knowledge level considers what an agent knows, but not how it reasons. At this level, you do not specify how the solution will be computed or even which of the many possible strategies available to the agent will be used.
- The symbol level is a level of description of an agent in terms of the reasoning it does. To implement the knowledge level, an agent manipulates symbols to produce answers. Many cognitive science experiments are designed to determine what symbol manipulation occurs during reasoning.

Given an informal description of a problem, before even considering a computer, a knowledge base designer should determine what would constitute a solution. This question arises not only in AI but in any software design. Much of software engineering involves refining the specification of the problem. Much work in AI is motivated by commonsense reasoning; we want the computer to be able to make commonsense conclusions about the unstated assumptions. Given a well-defined problem, the next issue is whether it matters if the answer returned is incorrect or incomplete [20]. There are four common classes of solutions (these categories are not exclusive):

1. Optimal solution: An optimal solution to a problem is one that is the best solution according to some measure of solution quality. This measure is typically specified as an ordinal, where only the order matters. However, in some situations, such as when combining multiple criteria or when reasoning under uncertainty, you need acardinal measure, where the relative magnitudes also matter. One general cardinal measure of desirability, known as utility, is used in decision theory.

2. Satisficing solution: Often an agent does not need the best solution to a problem but just needs some solution. A satisficing solution is one that is good enough according to some description of which solutions are adequate.

3. Approximately optimal solution: One of the advantages of a cardinal measure of success is that it allows for approximations. An approximately optimal solution is one whose measure of quality is close to the best that could theoretically be obtained. Typically agents do not need optimal solutions to problems; they only must get close enough. For some problems, it is much easier computationally to get an approximately optimal

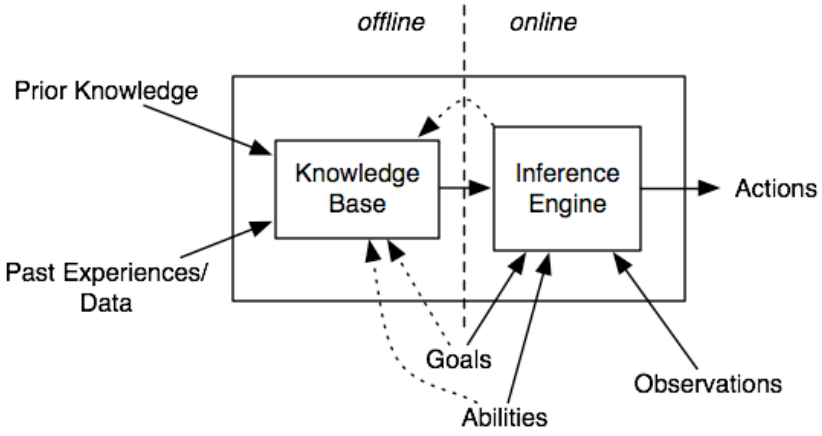
solution than to get an optimal solution. However, for other problems, it is (asymptotically) just as difficult to guarantee finding an approximately optimal solution as it is to guarantee finding an optimal solution. Some approximation algorithms guarantee that a solution is within some range of optimal, but for some algorithms no guarantees are available.

4. Probable solution: A probable solution is one that, even though it may not actually be a solution to the problem, is likely to be a solution. This is one way to approximate, in a precise manner, a satisficing solution. Often you want to distinguish the false-positive error rate (the proportion of the answers given by the computer that are not correct) from the false-negative error rate (which is the proportion of those answers not given by the computer that are indeed correct). Some applications are much more tolerant of one of these errors than of the other.

1.4 Reasoning (symbols), Perception and Acting

As already mentioned an agent has belief states that are maintained through time. The belief state can be complex even for a single layer, thus, experience in studying and building intelligent agents requires some internal representation of its belief state. Knowledge is the information about a domain and includes general information (knowledge) that can be applied to particular situations [22][23]. In AI systems, knowledge is typically but not necessarily true however this distinction often becomes blurry when a module of agent treat information as true while another module of an agent may be able to revise that information. A knowledge base is built offline and is used online to produce actions. This decomposition of an agent (Figure 3) is orthogonal to the layered view of an agent; an intelligent agent requires both hierarchical organization and knowledge bases.

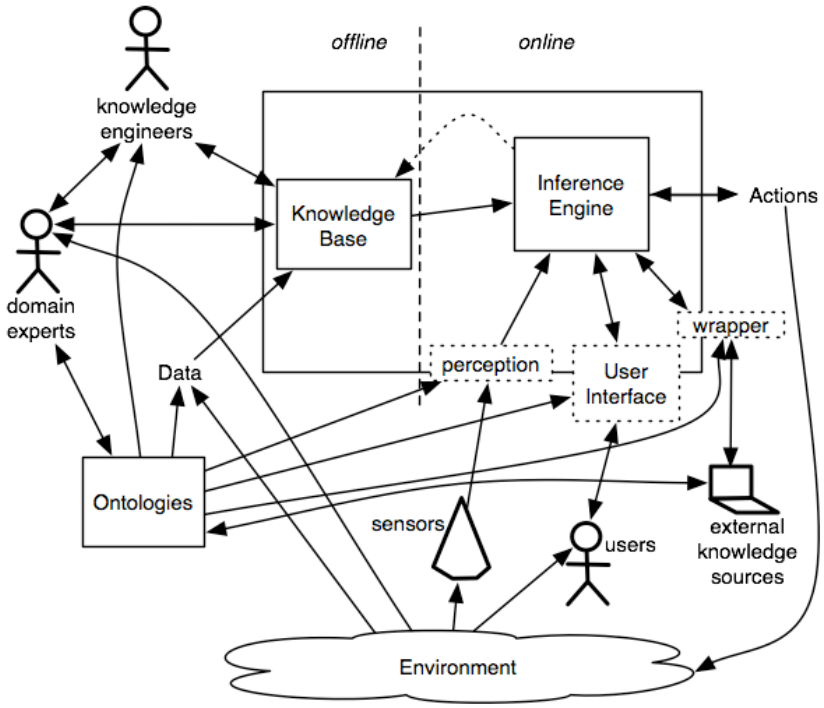
FIGURE 3: Offline and online decomposition of an agent



The knowledge base is its long-term memory, where it keeps the knowledge that is needed to act in the future. This knowledge comes from prior knowledge and is combined with what is learned from data and past experiences.

The belief state is the short-term memory of the agent, which maintains the model of current environment needed between time steps. Then, there is feedback from the inference engine to the knowledge base, because observing and acting in the world provide more data from which to learn. The goals and abilities are given offline, online, or both, depending on the agent. The online computation can be made more efficient if the knowledge base is tuned for the particular goals and abilities. However, this is often not possible when the goals and abilities are only available at runtime. Figure 4 shows more detail of the interface between the agents and the world.

FIGURE 4: Internals of an agent, showing roles



The manipulation of symbols to produce action is called reasoning. One way that AI representations differ from computer programs in traditional languages is that an AI representation typically specifies what needs to be computed, not how it is to be computed. Much AI reasoning involves searching through the space of possibilities to determine how to complete a task [24]. In deciding what an agent will do, there are three aspects of computation that must be distinguished:

- Design time reasoning is the reasoning that is carried out to design the agent. It is carried out by the designer of the agent, not the agent itself.
- Offline computation is the computation done by the agent before it has to act. It can include compilation and learning. Offline, the agent takes background knowledge and data and compiles them into a usable form called a knowledge base. Background knowledge can be given either at design time or offline.
- Online computation is the computation done by the agent between observing the environment and acting in the environment. A piece of information obtained online is called an observation. An agent typically must use both its knowledge base and its observations to determine what to do.

It is important to distinguish between the knowledge in the mind of the designer and the knowledge in the mind of the agent. Two broad strategies have been pursued in building agents:

- The first is to simplify environments and build complex reasoning systems for these simple environments. This is also important for building practical systems because many environments can be engineered to make them simpler for agents.
- The second strategy is to build simple agents in natural environments. This is inspired by seeing how insects (ants) can survive in complex environments even though they have very limited reasoning abilities.

One of the advantages of simplifying environments is that it may enable us to prove properties of agents or to optimize agents for particular situations. Proving properties or optimization typically requires a model of the agent and its environment [25].

1.5 Dimensions of Acting Complexity

Agents acting in environments range in complexity with multiple goals acting in competitive environments. A number of dimensions of complexity exist in the design of intelligent agents. These dimensions may be considered separately but must be combined to build an intelligent agent and define a design space of AI [26][27].

1. Modularity is the extent to which a system can be decomposed into interacting modules that can be understood separately. Modularity is important for reducing complexity. It is apparent in the structure of the brain, serves as a foundation of computer science, and is an important part of any large organization. In the modularity dimension, an agent's structure is one of the following:
   - flat: there is no organizational structure;

- modular: the system is decomposed into interacting modules that can be understood on their own;
- hierarchical: the system is modular, and the modules themselves are decomposed into interacting modules, each of which are hierarchical systems, and this recursion grounds out into simple components.

2. The representation scheme dimension concerns how the world is described. The different ways the world could be to affect what an agent should do are called states. We can factor the state of the world into the agent's internal state (its belief state) and the environment state. A state may be described in terms of features, where a feature has a value in each state. A proposition is a Boolean feature, which means that its value is either true or false. Thirty propositions can encode 2^{30}= 1,073,741,824 states. It may be easier to specify and reason with the thirty propositions than with more than a billion states. Thus, instead of dealing with features or propositions, it is often more convenient to have relational descriptions in terms of individuals and relations among them.

3. The Planning Horizon dimension is how far ahead in time the agent plans. In other words: How far the agent "looks into the future" when deciding what to do is called the planning horizon. In the planning horizon dimension, an agent is one of the following:
   - a non-planning agent is an agent that does not consider the future when it decides what to do or when time is not involved;
   - a finite horizon planner is an agent that looks for a fixed finite number of time steps ahead;
   - an indefinite horizon planner is an agent that looks ahead some finite, but not predetermined, number of steps ahead;
   - an infinite horizon planner is an agent that plans on going on forever. This is often called a process.

4. Uncertainty: An agent could assume there is no uncertainty, or it could take uncertainty in the domain into consideration. Uncertainty is divided into two dimensions: one for uncertainty from sensing and one for uncertainty about the effect of actions.
   - The sensing uncertainty dimension concerns whether the agent can determine the state from the observations:
     - fully observable is when the agent knows the state of the world from the observations;
     - partially observable is when the agent does not directly observe the state of the world. This occurs when many possible states can result in the same observations or when observations are noisy.

- The effect uncertainty dimension is that the dynamics can be:
    - deterministic, when the state resulting from an action is determined by an action and the prior state;
    - or stochastic, when there is only a probability distribution over the resulting states.

5. The preference dimension is whether the agent has:
    - goals, either achievement goals to be achieved in some final state or maintenance goals that must be maintained in all visited states;
    - complex preferences involve trade-offs among the desirability of various outcomes, perhaps at different times. An ordinal preference is where only the ordering of the preferences is important. Acardinal preference is where the magnitude of the values matters.

6. An agent reasoning about what it should do in an environment where it is the only agent is difficult enough. Many domains contain multiple agents and ignoring other agents' strategic reasoning is not always the best way for an agent to reason. Taking the point of view of a single agent, the number of agents dimension considers whether the agent does:
    - single agent reasoning, where the agent assumes that any other agents are just part of the environment. This is a reasonable assumption if there are no other agents or if the other agents are not going to change what they do based on the agent's action;
    - multiple agent reasoning, where the agent takes the reasoning of other agents into account. This happens when there are other intelligent agents whose goals or preferences depend, in part, on what the agent does or if the agent must communicate with other agents.

- The learning dimension determines whether knowledge is given or knowledge is learned (from data or past experience). Learning typically means finding the best model that fits the data. Sometimes this is as simple as tuning a fixed set of parameters, but it can also mean choosing the best representation out of a class of representations. Learning is a huge field in itself but does not stand in isolation from the rest of AI.

7. Sometimes an agent can decide on its best action quickly enough for it to act. The computational limits dimension determines whether an agent has:
    - perfect rationality, where an agent reasons about the best action without taking into account its limited computational resources;
    - bounded rationality, where an agent decides on the best action that it can find given its computational limitations.

1.6 Conclusions

AI is a very young discipline. Other disciplines as diverse as philosophy, neurobiology, evolutionary biology, psychology, economics, political science, sociology, anthropology, control engineering, and many more have been studying intelligence much longer. The science of AI could be described as synthetic-psychology, experimental-philosophy, or computational-epistemology[2] . AI can be seen as a way to study the old problem of the nature of knowledge and intelligence, but with a more powerful experimental tool than was previously available. Instead of being able to observe only the external behavior of intelligent systems, as philosophy, psychology, economics, and sociology have traditionally been able to do, AI researchers experiment with executable models of intelligent behavior. Most important, such models are open to inspection, redesign, and experiment in a complete and rigorous way. Moreover, AI theories can be empirically grounded in implementation we are often surprised when simple agents exhibit complex behavior. AI is intimately linked with the discipline of computer science. Although there are many non-computer scientists who are doing AI research, much, if not most, AI research is done within computer science departments. This is appropriate because the study of computation is central to AI. It is essential to understand algorithms, data structures, and combinatorial complexity to build intelligent machines and real-life systems. It is also surprising how much of computer science systems started as a spinoff from AI. Finally, AI is the umbrella of cognitive science. Cognitive science links various disciplines that study cognition and reasoning, from psychology to linguistics to anthropology to neuroscience. AI distinguishes itself within cognitive science by providing tools to build intelligence rather than just studying the external behavior of intelligent agents or dissecting the inner workings of intelligent systems.

1.7 References

[1] Haugeland, J. (Ed.) (1997). *Mind Design II: Philosohpy, Psycholgy, Artificial Intelligence*. MIT Press, Cambridge, MA, revised and enlarged edition.

[2] Turing, A. (1950). Computing machinery and intelligence. *Mind*, 59: 433-460.

[3] Cohen, P.R. (2005). If not Turing's test, then what? *AI Magazine*, 26(4): 61-67.

[4] Nilsson, N.J. (2009). *The Quest for Artificial Intelligence: A History of Ideas and Achievements*. Cambridge University Press, Cambridge, England.

[5] Chrisley, R. and Begeer, S. (2000). *Artificial intelligence: Critical Concepts in Cognitive Science*. Routledge, London and New York.

[6] Haugeland, J. (1985). *Artificial Intelligence: The Very Idea*. MIT Press, Cambridge, MA.

[2] epistemology is the study of knowledge

[7] Brooks, R. (1990). Elephants don't play chess. *Robotics and Autonomous Systems*, 6: 3-15.http://people.csail.mit.edu/brooks/papers/elephants.pdf.

[8] Nilsson, N. (2007). The physical symbol system hypothesis: Status and prospects. In e.a. M. Lungarella (Ed.), *50 Years of AI, Festschrift*, volume 4850 of *LNAI*, pp. 9-17. Springer.http://ai.stanford.edu/pers/pssh.pdf.

[9] Horvitz, E.J. (1989). Reasoning about beliefs and actions under computational resource constraints. In L. Kanal, T. Levitt, and J. Lemmer (Eds.), *Uncertainty in Artificial Intelligence 3*, pp. 301-324. Elsevier, New York.

[10] Boddy, M. and Dean, T.L. (1994). Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67(2): 245-285.

[11] Dechter, R. (1996). Bucket elimination: A unifying framework for probabilistic inference. In E. Horvitz and F. Jensen (Eds.), *Proc. Twelfth Conf. on Uncertainty in Artificial Intelligence (UAI-96)*, pp. 211-219. Portland, OR.

[12] Wilkins, D.E. (1988). *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann, San Mateo, CA.

[13] Russell, S. (1997). Rationality and intelligence. *Artificial Intelligence*, 94: 57-77.

[14] Simon, H.A. (1995). Artificial intelligence: an empirical science. *Artificial Intelligence*, 77(1): 95-127.

[15] Simon, H. (1996). *The Sciences of the Artificial*. MIT Press, Cambridge, MA, third edition.

[16] Bowen, K.A. (1985). Meta-level programming and knowledge representation. New Generation Computing, 3(4): 359-383.

[17] Brachman, R.J. and Levesque, H.J. (Eds.) (1985). Readings in Knowledge Representation. Morgan Kaufmann, San Mateo, CA.

[18] Kirsh, D. (1991a). Foundations of AI: the big issues. *Artificial Intelligence*, 47: 3-30.

[19] Kirsh, D. (1991b). Today the earwig, tomorrow man? *Artificial Intelligence*, 47: 161-184.

[20] Lenat, D.B. and Feigenbaum, E.A. (1991). On the thresholds of knowledge. *Artificial Intelligence*, 47: 185-250.

[21] Schank, R.C. (1990). What is AI, anyway? In D. Partridge and Y. Wilks (Eds.), *The Foundations of Artificial Intelligence*, pp. 3-13. Cambridge University Press, Cambridge, England.

[22] Brachman, R. and Levesque, H. (2004). Knowledge Representation and Reasoning. Morgan Kaufmann.

[23] Brooks, R.A. (1991). Intelligence without representation. *Artificial Intelligence*

[24] Aamodt, A. and Plaza, E. (1994). Case-based reasoning: foundational issues, methodological variations, and system approaches. AI Communications, 7(1): 39-59.

[25] Bowling, M. and Veloso, M. (2002). Multiagent learning using a variable learning rate. Artificial Intelligence, 136(2): 215-250.

[26] Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Series in Artificial Intelligence. Prentice-Hall, Englewood Cliffs, NJ, third edition. http://aima.cs.berkeley.edu/.

[27] Webber, B.L. and Nilsson, N.J. (Eds.) (1981). *Readings in Artificial Intelligence*. Morgan Kaufmann, San Mateo, CA.

2. Multi-Agent Systems, Taxonomy and Architectures

EXTENDING the realm of the social modern world to include autonomous systems has always been of paramount importance. However, examining the implications of multiple autonomous "agents" interacting in real world scenarios/schemes, what should an agent do when there are other agents, with their own values, who are also reasoning about what to do? We consider the problems given a mechanism that specifies how the world works, and of designing a mechanism that has useful properties.

2.1 Introduction

Multiagent Systems (MAS) is the emerging subfield of AI that aims to provide both principles for construction of complex systems involving multiple agents and mechanisms for coordination of independent agents' behaviors. While there is no generally accepted definition of "agents" in AI [68], for the purposes of this Thesis, we consider an agent to be an entity with goals, actions, and domain knowledge, situated in an environment and the way it acts is called "behavior". Although the ability to consider coordinating behaviors of autonomous agents is a new one, the field is advancing quickly by building upon pre-existing work in the field of Distributed Artificial Intelligence (DAI). The main topics considered in Distributed Problem Solving (DPS) are information management issues such as task decomposition and solution synthesis. For example, a constraint satisfaction problem can often be decomposed into several not entirely independent sub-problems that can be solved on different processors. Then these solutions can be synthesized into a solution of the original problem. MAS allows the sub-problems of a constraint satisfaction problem to be subcontracted to different problem solving *agents* with their own interests and goals. Because of the inherent complexity of MAS, there is much interest in using machine learning techniques to help deal with problem complexity[3] [1][2]. Two obvious questions about any type of technology are:

- What kind of advantages does MAS offer over the alternatives?
- In what circumstances MAS are useful?

Like any useful approach, there are some situations for which it is particularly appropriate, and others for which it is not. Thus, it would not be right to claim that MAS should be used when designing all complex systems. The goal of this section is to underscore the need for MAS while giving characteristics of typical domains that can benefit from it [3].

[3] When several different systems exist that could illustrate the same or similar MAS techniques, the systems presented here are biased towards those that use machine learning (ML) approaches.

2.2 Multi-Agent Systems (MAS):

The most important reason to use MAS when designing a system is that some domains require it. In particular, if there are different people or organizations with different (possibly conflicting) goals and proprietary information, then a multi-agent system is needed to handle their interactions. Even if each organization wants to model its internal affairs with a single system, the organizations will not give authority to any single person to build a system that represents them all: the different organizations will need their own systems that reflect their capabilities and priorities.

Paradigm: Consider a manufacturing scenario in which company X produces tires, but subcontracts the production of lug-nuts to company Y. In order to build a single system to automate (certain aspects of) the production process, the internals of both companies X and Y must be modeled. However, neither company is likely to want to relinquish information and/or control to a system designer representing the other company. Perhaps with just two companies involved, an agreement could be reached, but with several companies involved, MAS is necessary. The only feasible solution is to allow the various companies to create their own agents that accurately represent their goals and interests. They must then be combined into a multi-agent system with the aid of some of the techniques described in this article [20].

Even in domains that could conceivably use systems that are not distributed, there are several possible reasons to use MAS. Having multiple agents could speed up a system's operation by providing a method for parallel computation. Furthermore, the parallelism of MAS can help deal with limitations imposed by time-bounded reasoning requirements. While parallelism is achieved by assigning different tasks or abilities to different agents, robustness is a benefit of multi-agent systems that have redundant agents. If control and responsibilities are sufficiently shared among different agents, the system can tolerate failures by one or more of the agents. Another benefit of multi-agent systems is their scalability. Since they are inherently modular, it should be easier to add new agents to a multi-agent system than it is to add new capabilities to a monolithic system. Systems whose capabilities and parameters are likely to need to change over time or across agents can also benefit from this advantage of MAS. From a programmer's perspective the modularity of multi-agent systems can lead to simpler programming. Rather than tackling the whole task with a centralized agent, programmers can identify subtasks and assign control of those subtasks to different agents. Thus, when the choice is between using a multi-agent system or a single-agent system, MAS is often the simpler option [4].

2.2.1 MAS Taxonomy

Several taxonomies have been presented[4] for MAS [16]:

1. Agent granularity (coarse vs. fine);
2. Heterogeneity of agent knowledge (redundant vs. specialized);
3. Methods of distributing control (benevolent vs. competitive, team vs. hierarchical, static vs. shifting roles);
4. Communication possibilities (blackboard vs. messages, low-level vs. high-level, content).

Along dimensions 1 and 4, multi-agent systems have coarse agent granularity and high-level communication. Along the other dimensions, they can vary across the whole ranges. In addition, Parunak has presented a taxonomy of MAS from an application perspective[5] [5]:

1. System function;
2. Agent architecture (degree of heterogeneity, reactive vs. deliberative);
3. System architecture (communication, protocols, human involvement).

Other overviews of DAI and/or MAS include [6][7]. The three combinations of heterogeneity and communication considered in this article (homogeneous non-communicating agents; heterogeneous non-communicating agents; and heterogeneous communicating agents) are presented in order of increasing complexity and power. The multi-agent scenarios along with the issues that arise therein are summarized in Table 2.

TABLE 2: Issues arising in the various scenarios as reflected in the literature.

Homogeneous Non-Communication Agents	Heterogeneous Non-Communicating Agents
▪ Reactive vs. Deliberative Agents ▪ Local or Global Perspectives ▪ Modeling of other Agents' states	▪ Benevolence vs. Competitiveness ▪ Evolving Agents ▪ Modeling of other Agents' goals, actions ▪ Resource Management

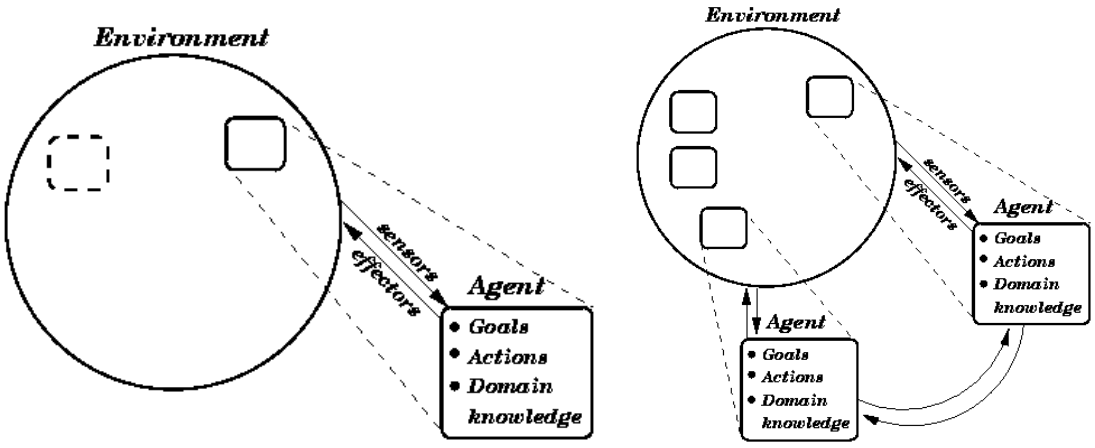
Heterogeneous Communicating Agents
▪ Mutual understanding ▪ Planning Communicative acts ▪ Benevolence vs. Competitiveness ▪ Resource Management ▪ Commitment/De-commitment

[4] For the related field of Distributed Artificial Intelligence (DAI)
[5] The dimensions are divided into agent and system characteristics

2.2.2 From Single-2-Multi-Agent Systems

Although it might seem that single-agent systems should be simpler than multi-agent systems, when dealing with a fixed, complex task, the opposite is often the case (see Figure 5). Thus centralized, single-agent systems belong at the end of the progression from simple to complex multi-agent systems. The agent in a single-agent system models itself the environment, the interactions and of course the agent itself is part of the environment, because agents are considered to have extra-environmental components as well. They are independent entities with their own goals, actions, and domain knowledge (AI) (see Figure 6).

FIGURE 5: A general single-agent framework. The agent models itself, the environment, and their interactions. If other agents exist, they are considered part of the environment.
FIGURE 6: The fully general multi-agent scenario. Agents model each other's goals and actions; they may also interact directly (communicate)



On the other side, multi-agent systems differ from single-agent systems in several ways, and agents model a plethora of goals and multiple actions without excluding a direct interaction among agents (communication)[6]. From an individual agent's perspective, multiagent systems differ from single-agent systems most significantly in that the environment's dynamics can be determined by other agents. In addition to the uncertainty that may be inherent in the domain, other agents intentionally affect the environment in unpredictable ways. Thus, all multiagent systems can be viewed as having dynamic environments. Figure 6 illustrates the view that each agent is both part of the environment and modeled as a separate entity. There may be any number of agents, with different degrees of heterogeneity and with or without the ability to communicate directly.

[6] This interaction could be viewed as environmental stimuli

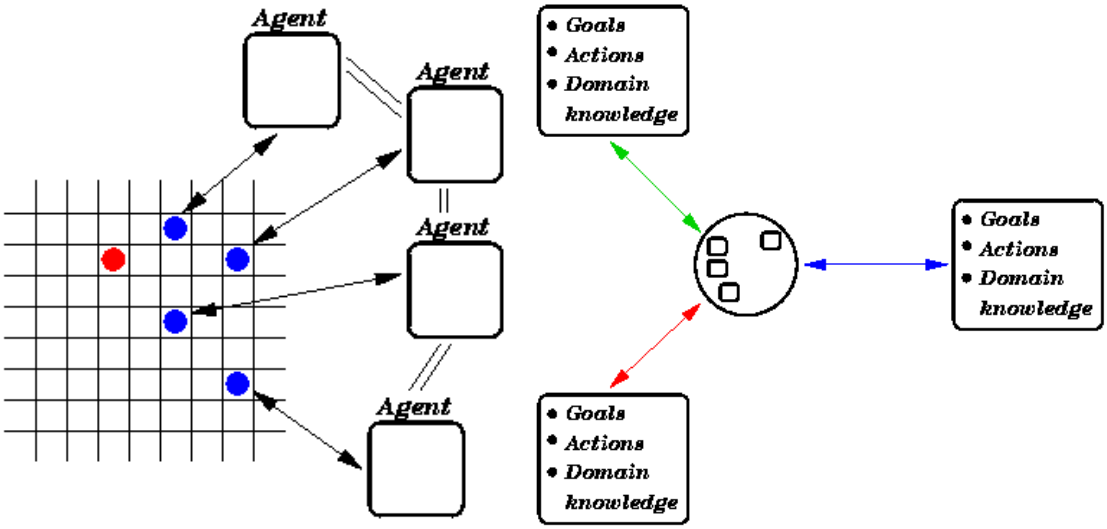
2.3 Homogeneous Non-Communicating (HoNC) MAS

The simplest multi-agent scenario involves homogeneous non-communicating (HONC) agents. In this scenario, all of the agents have the same internal structure including goals, domain knowledge, and possible actions. They also have the same procedure for selecting among their actions. The only differences among agents are their sensory inputs and the actual actions they take: they are situated differently in the world.

2.3.1 HoNC Multi-Agent Goal

In the homogeneous non-communicating version of the pursuit domain, although the agents have identical capabilities and decision procedures, they may have limited information about each other's internal state and sensory inputs. Thus they may not be able to predict each other's actions. The pursuit domain with homogeneous agents is illustrated in Figure 7.

FIGURE 7: The pursuit domain with homogeneous agents. There is one identical agent per predator. Agents may have (the same amount of) limited information about other agents' internal states.
FIGURE 8: MAS with homogeneous agents. Only the sensor input and effector output of agents differ, as represented by the different arrow styles. The agents' goals, actions, and/or domain knowledge are all identical as indicated by the identical fonts.



Within this framework, Stephens and Merx propose a simple heuristic behavior for each agent that is based on local information [8]. They define capture positions as the four positions adjacent to the prey. They then propose a “local” strategy whereby each predator agent determines the capture position to which it is closest and moves towards that position. The predators cannot see each other, so they cannot aim at different capture positions. Of

course a problem with this heuristic is that two or more predators may move towards the same capture position, blocking each other as they approach[7].

Since the predators are identical, they can easily predict each other's actions given knowledge of each other's sensory input. Vidal and Durfee analyze such a situation using the Recursive Modeling Method (RMM) [9]. Levy and Rosenschein use a game theoretical approach to the pursuit domain [10]. Korf also takes the approach that each agent should try to greedily maximize its own local utility [11] (optimization). As additional support for the theory that much coordination and cooperation in both natural and man-made systems can be viewed as an emergent property of the interaction of agents maximizing their particular utility functions in the presence of environmental constraints.

However, whether or not altruism occurs, there is certainly some use for benevolent agents in MAS and Haynes and Sen show that Korf's heuristics do not work for certain instantiations of the domain [12]. The general multi-agent scenario with homogeneous agents is illustrated in Figure 8. There are several different agents with identical structure (sensors, effectors, domain knowledge, and decision functions), but they have different sensor input and effector output; a necessary condition for MAS[8].

2.3.2 Open Issues, Techniques and Perspectives

Even in this simplest of multi-agent scenarios, there are several issues with which to deal and the techniques provided below are representative ways to deal with these issues.

- **Reactive vs. Deliberative agents:** When designing any agent-based system, it is important to determine how sophisticated the agents' reasoning will be. Reactive agents simply retrieve pre-set behaviors similar to reflexes without maintaining any internal state-Balch and Arkin [13]. On the other hand, deliberative agents behave more like they are thinking, by searching through a space of behaviors, maintaining internal state, and predicting the effects of actions - Levy and Rosenschein [10]. In other cases, the red thin line between reactive and deliberative agents can be somewhat blurry, an agent with no internal state is certainly reactive, and one which bases its actions on the predicted actions of other agents is deliberative - Rao and Georgeff's [14].

[7] This strategy serves as a basis for comparison with two other control strategies-"distributed" and "central"
[8] if the agents all act as a unit, then they are essentially a single agent

- **Local or global perspective:** Another issue to consider when building a multi-agent system is how much sensor information should be available to the agents (global perspective of the environment vs. limitations to local views) [15]. Thus, a better performance by agents with less knowledge is occasionally summarized by the cliche "Ignorance is Bliss" [16].

- **Modeling multiple-agents' state:** Recursive Modeling Method (RMM) models the internal state of another agent in order to predict its actions. Even though the agents know each other's goals and structure (they are homogeneous), they may not know each other's future actions. The missing pieces of information are the internal states (for deliberative agents) and sensory inputs of the other agents. In the more complex multi-agent scenarios presented in the next sub-sections of the present chapter, agents may have to model not only the internal states of other agents, but also their goals, actions, and abilities. Schmidhuber advocates a form of multi-agent reinforcement learning (RL) with which agents do not model each other as agents [17]. Instead they consider each other as parts of the environment and affect each other's policies only as sensed objects.

- **Beyond Advantages:** Inspired by the concept of stigmergy, an agent may try to learn to take actions that will not directly help it in its current situation, but that may allow other similar agents to be more effective in the future. Typical RL situations with delayed reward encourage agents to learn to achieve their goals directly by propagating local reinforcement back to past states and actions [18].

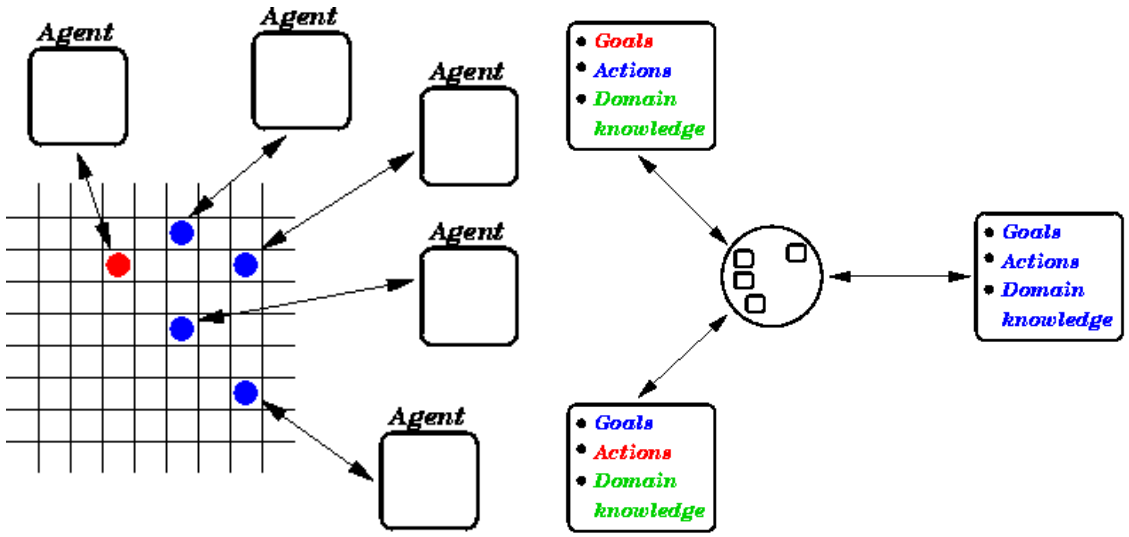
2.4 Heterogeneous Non-Communicating (HeNC) MAS

Adding the possibility of agents acting in a heterogeneous non-communicating (HeNC) multi-agent domain is of great potential power at the dimension of complexity and the aspect whether agents can be benevolent or competitive.

2.4.1 HeNC Multi-Agent Goal

In the HeNC multi-agent scenario the predators are controlled by separate agents and their goals, actions and domain knowledge may differ. In addition, the prey, which inherently has goals different from those of the predators, can now be modeled as an agent (Figure 9) - Haynes et al. [19]

FIGURE 9: The pursuit domain with heterogeneous agents. Goals and actions may differ among agents. Now the prey may also be modeled as an agent.
FIGURE 10: The general heterogeneous MAS scenario. Now agents' goals, actions, and/or domain knowledge may differ as indicated by the different fonts. The assumption of no direct interaction remains.



The general multi-agent scenario with heterogeneous non-communicating agents is depicted in Figure 10. The agents are situated differently in the environment which causes them to have different sensory inputs and necessitates their taking different actions. However in this scenario, the agents have much more significant differences. They may have different goals, actions, and/or domain knowledge. This condition of heterogeneity among agents adds a great deal of power for the system designer.

2.4.2 Open Issues, Techniques and Perspectives

Even without communication, numerous issues that were not present in the homogeneous agent scenario arise in this scenario. These issues and existing techniques along with further learning opportunities are summarized below.

- **Benevolence vs. competitiveness:** When designing a multi-agent system it is important to consider the plethora of different agents that will be benevolent or competitive. Even if they have different goals, the agents can be benevolent to achieve their respective goals [20] or the agents may be selfish and only consider their own goals. In some extreme scenarios, the agents may be involved in a zero-sum situation and actively oppose other agents' goals in order to achieve their own. Korf used greedy agents that minimized their own distance to the target/goarl [11], and similarly, Levy and

Rosenschein used Game Theory to study how agents can cooperate despite maximizing their own utilities [10]. Ridley provides a detailed chronicle and explanation of apparent altruism in nature (usually explainable as kin selection) and cooperation in human societies [64]. Whether or not altruism exists, in some situations it may be in an agent's interest to cooperate with other agents as Mor and Rosenschein illustrate this possibility in the context of the prisoner's dilemma [21], where two agents try to act so as to maximize their own individual rewards. However, when the agents are actually competitive (such as in zero-sum games), cooperation is no longer sensible as Littman introduced the variant of Q-learning called Minimax-Q which was designed to work on Markov games as opposed to Markov Decision Processes [22]. Thus, the issue of benevolence (willingness to cooperate) vs. competitiveness comes up repeatedly in many systems of our modern society.

- **Stable vs. evolving agents**: Another important characteristic to consider when designing multi-agent systems is whether the agents are stable or evolving/competitive. Of course evolving agents can be useful in dynamic environments. But particularly when using competitive agents, allowing them to evolve can lead to complications. Such systems –dynamic- that use competitive evolving agents use a technique called competitive co-evolution. Systems that evolve benevolent agents are said to use cooperative co-evolution -Haynes and Sen [23] and Grefenstette and Daley [24]. One problem to contend with in competitive rather than cooperative co-evolution is the possibility of an escalating "arms race" with no end. Competing agents might continually adapt to each other in more and more specialized ways, never stabilizing at a good behavior. Of course in a dynamic environment, it may not be feasible or even desirable to evolve a stable behavior. Another issue in competitive co-evolution is the credit/blame assignment problem. When performance of an agent improves, it is not necessarily clear whether the improvement is due to an improvement in that agent's behavior or a negative change in the opponent's behavior. Similarly, if an agent's performance gets worse, the blame or credit could belong to that agent or to the opponent. One way to deal with the credit/blame problem is to fix one agent while evolving the other and then switch. Of course this method encourages the arms race more than ever. Nevertheless, Rosin and Belew use this technique, along with an interesting method for maintaining diversity in genetic populations, to evolve agents that can play TicTacToe, Nim, and a simple version of Go [25].

- **Resource management:** Heterogeneous agents may have interdependent actions due to limited resources needed by several of the agents. Example domains include network traffic problems in which several different agents must send information through the

same network; and load balancing in which several computer processes or users have a limited amount of computing power to share among them. Designers of multiagent systems with limited resources must decide how the agents will share the resources - Braess' paradox [26]. Braess' paradox is the phenomenon of adding more resources to a network but getting worse performance.

- **Beyond Advantages:** When several different agents are evolving at the same time, changes in an agent's behavior or due to the behavior of other agents can be observed. Yet if agents are to evolve effectively, they must have a reasonable idea of whether a given change in behavior is beneficial or detrimental. Methods of objective fitness measurement are also needed for testing various evolution techniques. In competitive (especially zero-sum) situations, it is difficult to provide adequate performance measurements over time. In a dynamic environment, these flexible agents are more effective if they can switch roles dynamically. Dynamic role assumption is a particularly good opportunity for ML researchers in MAS.

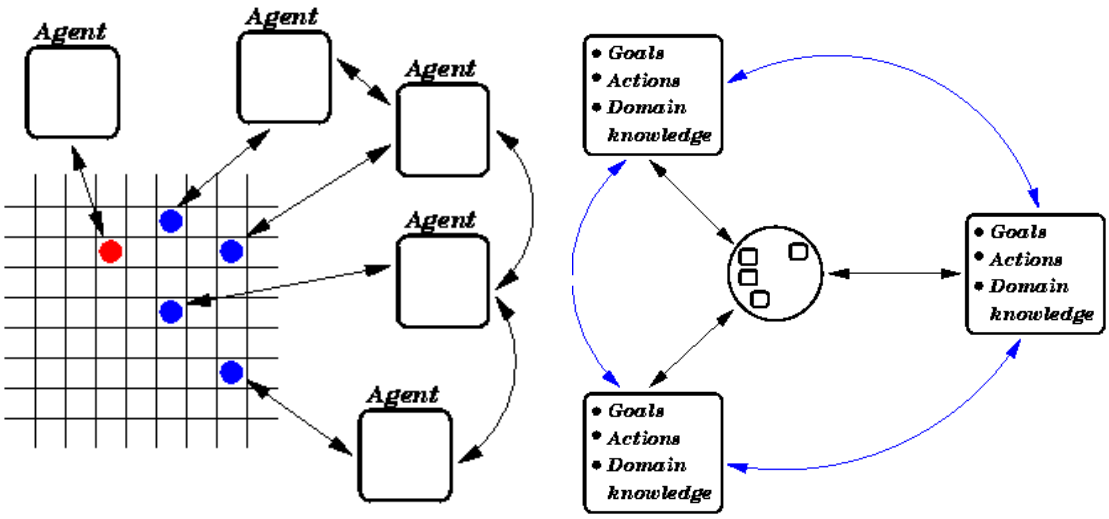
2.5 Heterogeneous Communicating (HEC) MAS

Even though heterogeneous multi-agent systems can be very complex and powerful, the full power of MAS is unleashed when adding the ability for agents to communicate with one another. By sending their sensor inputs to and receiving their commands from one agent, all the other agents can surrender control to that single agent. Thus communicating heterogeneous agents can span the full range of complexity in extra-environmental agent systems.

2.5.1 HEC Multi-Agent Goal

Communication creates new possibilities for agent behavior as they exchange information in order to help them capture the target more effectively as illustrated in Figure 11. The continuum of complexity leading into the multi-agent scenario as it appears in Figure 12. In this scenario, we allow the agents to be heterogeneous to any degree from homogeneity to full heterogeneity. The key addition is the ability for agents to transmit information directly to each other. Tan [27] uses communicating agents in the pursuit domain to conduct some interesting multi-agent Q-learning experiments. Benda et al. [28] also consider the full range of communication possibilities of a central strategy. They consider the possible organizations of the four agents when any pair can either exchange data, exchange data and goals, or have one control the other.

FIGURE 11: The pursuit domain with communicating agents. Agents can still be fully heterogeneous but now the predators can communicate with one another.
FIGURE 12: The general communicating MAS scenario. Agents can be heterogeneous to any degree. Information can be transmitted directly among agents as indicated by the arrows between agents. Communication can either be broadcast or transmitted point-to-point.



The tradeoff between lower communication costs and better decisions is described and communication costs might come in the form of limited bandwidth or consumption of reasoning time. Osawa suggests that the agents should move through four phase: autonomy, communication, negotiation, and control [29]. When the agents stop making sufficient progress toward the target using one strategy, they should move to the next most expensive strategy.

2.5.2 Open Issues, Techniques and Perspectives

The issue of benevolence vs. competitiveness as already discussed in the previous sub-sections, becomes more complicated in this context. The summaries of issues are described as found on Table 7.

- **Mutual Understanding:** In all communicating multi-agent systems[9] there must be some set language and protocol for the agents to use when interacting. Thus, MAS designers must carefully consider what features in a communication protocol are needed in a given domain.

[9] particularly in domains that include agents built by different designers

- **Planning communicative acts:** With the addition of communication as a capability available to agents, when an agent transmits information to another agent, it has an effect just like any other action would have. Thus within a planning framework, one can define pre-conditions and effects for communicative acts. The theory of communication as action is called speech acts [30][31][32][33].

- **Benevolence vs. competitiveness:** Several studies involving competitive agents were described in the heterogeneous non-communicating scenario. In this scenario, there are many more examples of competitive agents. As opposed to the process of shaping, in which the system designer develops simple behaviors and slowly builds them into more complex ones, populations appropriately seeded for competitive co-evolution can reduce the amount of designer effort [34][35].

- **Commitment/De-commitment:** Committing to another agent involves agreeing to pursue a given goal, possibly in a given manner; regardless of how much it serves one's own interests. Commitments can make systems run much more smoothly by providing a way for agents to ``trust'' each other. The theory of commitment and decommitment (when the commitment terminates) has consequently drawn considerable attention [36].

- **Beyond Advantages:** Agents can also coordinate schedules while a Generalized Partial Global Planning (GPGP) allows several heterogeneous agents to post constraints, or commitments to do a task by some time, to each other's local schedulers and thus coordinate without the aid of any centralized agent [37]. Once again, there are many possible ways in the current scenario to enhance MAS. A very interesting alternative would be to allow the agents to learn for themselves how and what to communicate and to interpret it. A possible result would be more efficient communications in domains with low bandwidth or large time delays.

2.6 Conclusions

This chapter is presenting an overall description of the field of Multi-Agent Systems (MAS) serving as an introduction for people unfamiliar with the field to system engineers. In this chapter a series of three increasingly complex and powerful scenarios are presented. The first scenario involves systems with homogeneous non-communicating agents, the second scenario involves heterogeneous non-communicating agents and the final involves, the general MAS scenario involves communicating agents with any degree of heterogeneity.

2.7 References

[1] Gerhard Weiß and Sandip Sen, editors. Adaptation and Learning in Multiagent Systems. Springer Verlag, Berlin, 1996.

[2] AAAI. Adaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium, Menlo Park,CA, March 1996. AAAI Press. AAAI Technical Report SS-96-01. Sandip Sen-Chair.

[3] Alan H. Bond and Les Gasser. An analysis of problems and research in DAI. In Alan H. Bond and Les Gasser, editors, Readings in Distributed Artificial Intelligence, pages 3-35. Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[4] Keith S. Decker. Task environment centered simulation. In M. Prietula, K. Carley, and L. Gasser, editors, Simulating Organizations: Computational Models of Institutions and Groups. AAAI Press/MIT Press, 1996.

[5] H. Van Dyke Parunak. Applications of distributed artificial intelligence in industry. In G. M. P. O'Hare and N. R. Jennings, editors,Foundations of Distributed Artificial Intelligence, pages 139-164. Wiley Interscience, 1996.

[6] Edmund H. Durfee. What your computer really needs to know, you learned in kindergarten. In Proceedings of the Tenth National Conference on Artificial Intelligence, Philadelphia, PA, 1992. Morgan Kaufman. Invited Talk.

[7] Victor R. Lesser. Multiagent systems: An emerging subdiscipline of AI. ACM Computing Surveys, 27(3):340-342, September 1995.

[8] Larry M. Stephens and Matthias B. Merx. The effect of agent control strategy on the performance of a dai pursuit problem. InProceedings of the 10th International Workshop on Distributed Artificial Intelligence, Bandera, Texas, October 1990.

[9] Jose M. Vidal and Edmund H. Durfee. Recursive agent modeling using limited rationality. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), pages 376-383, Menlo Park, California, June 1995. AAAI Press.

[10] Ran Levy and Jeffrey S. Rosenschein. A game theoretic approach to the pursuit problem. In Working Papers of the 11th International Workshop on Distributed Artificial Intelligence, pages 195-213, February 1992.

[11] Richard E. Korf. A simple solution to pursuit games. In Working Papers of the 11th International Workshop on Distributed Artificial Intelligence, pages 183-194, February 1992.

[12] Thomas Haynes, Kit Lau, and Sandip Sen. Learning cases to compliment rules for conflict resolution in multiagent systems. InAdaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium, pages 51-56, Menlo Park,CA, March 1996. AAAI Press. AAAI Technical Report SS-96-01.

[13] Tucker Balch and Ronald C. Arkin. Motor schema-based formation control for multiagent robot teams. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), pages 10-16, Menlo Park, California, June 1995. AAAI Press.

[14] Anand S. Rao and Michael P. Georgeff. Bdi agents: From theory to practice. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), pages 312-319, Menlo Park, California, June 1995. AAAI Press.

[15] Shounak Roychowdhury, Neeraj Arora, and Sandip Sen. Effects of local information on group behavior. In Adaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium, pages 78-83, Menlo Park,CA, March 1996. AAAI Press. AAAI Technical Report SS-96-01

[16] Edmund H. Durfee. Blissful ignorance: Knowing just enough to coordinate well. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), pages 406-413, Menlo Park, California, June 1995. AAAI Press

[17] Jurgen Schmidhuber. A general method for multi-agent reinforcement learning in unrestricted environments. In Adaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium, pages 84-87, Menlo Park,CA, March 1996. AAAI Press. AAAI Technical Report SS-96-01.

[18] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. Journal of Artificial Intelligence Research, 4:237-285, May 1996.

[19] Thomas Haynes, Roger Wainwright, Sandip Sen, and Dale Schoenefeld. Strongly typed genetic programming in evolving cooperation strategies. In Stephanie Forrest, editor, Proceedings of the Sixth International Conference on Genetic Algorithms, pages 271-278, San Mateo, CA, July 1995. Morgan Kaufman

[20] Claudia Goldman and Jeffrey Rosenschein. Emergent coordination through the use of cooperative state-changing rules. InProceedings of the Twelfth National Conference on Artificial Intelligence, pages 408-413, Philadelphia, PA, 1994. Morgan Kaufman

[21] Yishay Mor and Jeffrey Rosenschein. Time and the prisoner's dilemma. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), pages 276-282, Menlo Park, California, June 1995. AAAI Press.

[22] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In Proceedings of the Eleventh International Conference on Machine Learning, pages 157-163, San Mateo, CA, 1994. Morgan Kaufman.

[23] Thomas Haynes and Sandip Sen. Evolving behavioral strategies in predators and prey. In Gerhard Weißand Sandip Sen, editors,Adaptation and Learning in Multiagent Systems, pages 113-126. Springer Verlag, Berlin, 1996.

[24] John Grefenstette and Robert Daley. Methods for competitive and cooperative co-evolution. In Adaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium, pages 45-50, Menlo Park,CA, March 1996. AAAI Press. AAAI Technical Report SS-96-01.

[25] Shounak Roychowdhury, Neeraj Arora, and Sandip Sen. Effects of local information on group behavior. In Adaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium, pages 78-83, Menlo Park,CA, March 1996. AAAI Press. AAAI Technical Report SS-96-01.

[26] Natalie S. Glance and Tad Hogg. Dilemmas in computational societies. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), pages 117-124, Menlo Park, California, June 1995. AAAI Press.

[27] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In Proceedings of the Tenth International Conference on Machine Learning, pages 330-337, 1993.

[28] M. Benda, V. Jagannathan, and R. Dodhiawala. On optimal cooperation of knowledge sources - an empirical investigation. Technical Report BCS-G2010-28, Boeing Advanced Technology Center, Boeing Computing Services, Seattle, Washington, July 1986.

[29] Ei-Ichi Osawa. A metalevel coordination strategy for reactive cooperative planning. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), pages 297-303, Menlo Park, California, June 1995. AAAI Press.

[30] Philip R. Cohen and Hector J. Levesque. Communicative actions for artificial agents. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), pages 65-72, Menlo Park, California, June 1995. AAAI Press.

[31] Andreas Lux and Donald Steiner. Understanding cooperation: an agent's perspective. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), pages 261-268, Menlo Park, California, June 1995. AAAI Press.

[32] Maja J. Mataric. Interaction and intelligent behavior. MIT EECS PhD Thesis AITR-1495, MIT AI Lab, August 1994.

[33] Maja J. Mataric. Learning to behave socially. In Third International Conference on Simulation of Adaptive Behavior, 1994.

[34] L. Bull, T.C. Fogarty, and M. Snaith. Evolution in multi-agent systems: Evolving communicating classifier systems for gait in a quadrupedal robot. In Stephanie Forrest, editor, Proceedings of the Sixth International Conference on Genetic Algorithms, pages 382-388, San Mateo,CA, July 1995. Morgan Kaufman.

[35] Mitchell A. Potter, Kenneth A. De Jong, and John J. Grefenstette. A coevolutionary approach to learning sequential decision rules. In Stephanie Forrest, editor, Proceedings of the Sixth International Conference on Genetic Algorithms, pages 366-372, San Mateo,CA, July 1995. Morgan Kaufman.

[36] Cristiano Castelfranchi. Commitments: From individual intentions to groups and organizations. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), pages 41-48, Menlo Park, California, June 1995. AAAI Press.

[37] Keith S. Decker and Victor R. Lesser. Designing a family of coordination algorithms. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), pages 73-80, Menlo Park, California

3. Graph Searching Methods and Swarm Intelligence

THE NOTION of search is computation inside the agent and it is not only different from searching in the world but it is also different from searching the web, which involves searching for information [1]. The idea of search in this chapter is straightforward meaning an internal representation for a path to a goal. More specifically, the agent constructs a set of potential partial solutions to a problem that can be checked to see if they truly are solutions or if they could lead to solutions. Search proceeds by repeatedly selecting a partial solution, stopping if it is a path to a goal, and otherwise extending it by one more arc in all possible ways [2]. Therefore, search underlies much of artificial intelligence and strategies for providing the best solution to a specific problem.

3.1 Introduction

The formulation of intelligent actions [3] is in terms of state space that contains all of the necessary information to predict the effects of an action and then determine the goal state if

- the agent has perfect knowledge of the state space and can observe it;
- the agent has a set of actions that have known deterministic effects;
- the agent can recognize a goal state;
- the agent will perform the sequence of actions to get from current state to goal state.

In addition, a state-space problem is consisted of:

- a set of states and a distinguished set of states (aka start states);
- a set of actions available to the agent in each state and an action function;
- a set of goal states (Boolean function) and the criterion that specifies the quality of an acceptable solution (optimal solution and minimal total cost).

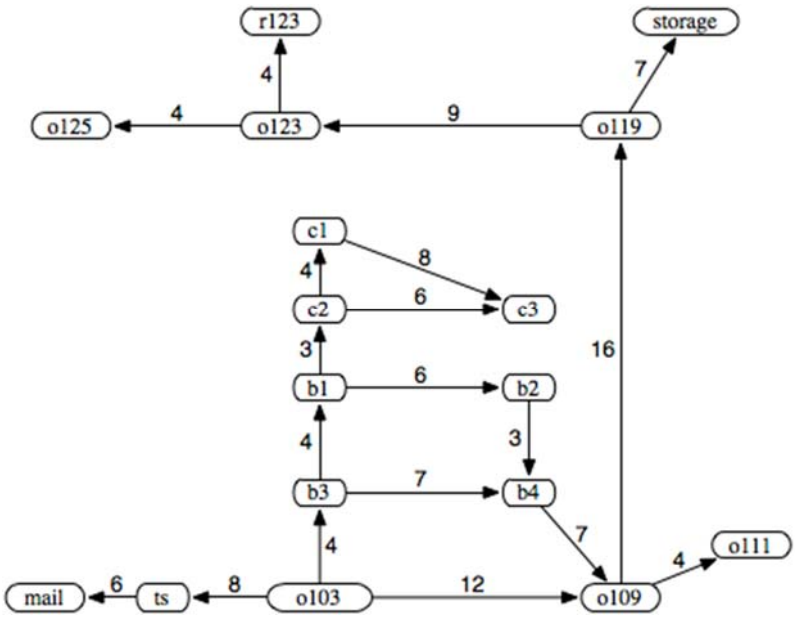
3.1.1 Generic Graph Searching Mechanism

Therefore, many problem-solving tasks can be transformed into the problem of finding a path in a graph [3]. In this sub-section, a search mechanism in terms of paths in directed graphs, to solve a problem is presented [4]. Searching in graphs provides an appropriate level of abstraction to study simple or complex problems of a particular domain. More particularly, a (directed) graph is a set of nodes and a set of directed arcs between nodes. The idea is to find a path along these arcs from a start node to a goal node thus the abstraction is necessary to represent a problem as a graph. A directed graph is consisted of:

- a set N of nodes and
- a set A of ordered pairs of nodes called arcs.

A node can be anything as there can be infinitely many nodes and arcs. The arc $\langle n_1,n_2\rangle$ is an outgoing arc from n_1 and an incoming arc to n_2. A node n_2 is a neighbor[10] of n_1 if there is an arc from n_1 to n_2; that is, if $\langle n_1,n_2\rangle \in A$. A path from node s to node g is a sequence of nodes $\langle n_0, n_1,\ldots, n_k\rangle$ such that $s=n_0$, $g=n_k$, and $\langle n_{i-1},n_i\rangle \in A$; that is, there is an arc from n_{i-1} to n_i for each i. Sometimes it is useful to view a path as the sequence of arcs, $\langle n_o,n_1\rangle, \langle n_1,n_2\rangle,\ldots, \langle n_{k-1},n_k\rangle$, or a sequence of labels of these arcs. A cycle is a nonempty path such that the end node is the same as the start node - that is, a cycle is a path $\langle n_0, n_1,\ldots, n_k\rangle$ such that $n_0=n_k$ and $k\neq 0$. A directed graph without any cycles is called a directed acyclic graph (DAG). A tree is a DAG where there is one node with no incoming arcs and every other node has exactly one incoming arc. Consider the problem of the delivery ship finding a path from location *o103* to location *r123* in the domain where the interesting locations are named. Figure 3.2 shows the resulting graph where the nodes represent locations and the arcs represent possible single steps between locations.

FIGURE 13: A graph with arc costs for the delivery ship domain



In figure 13, each arc is shown with the associated cost of getting from one location to the next. In this graph, the nodes are $N=\{mail,ts,o103,b3,o109,\ldots\}$ and the arcs are $A=\{\langle ts,mail\rangle, \langle o103,ts\rangle, \langle o103,b3\rangle, \langle o103,o109\rangle, \ldots\}$. Node *o125* has no neighbors. Node *ts* has one neighbor, namely *mail*. Node *o103* has three neighbors, namely *ts*, *b3*, and *o109*. There are three paths from *o103* to *r123*:

[10] Note that being a neighbor does not imply symmetry; just because n2 is a neighbor of n1 does not mean that n1 is necessarily a neighbor of n2. Arcs may be labeled, for example, with the action that will take the agent from one state to another.

⟨o103, o109, o119, o123, r123⟩

⟨o103, b3, b4, o109, o119, o123, r123⟩

⟨o103, b3, b1, b2, b4, o109, o119, o123, r123⟩

If *o103* were a start node and *r123* were a goal node, each of these three paths would be a solution to the graph-searching problem.

3.1.2 Generic Graph Searching Algorithm

Nevertheless, in many problems the search graph is not given explicitly; it is dynamically constructed as needed. All that is required for the search algorithms is a way to generate the neighbors of a node and to determine if a node is a goal node. The forward branching factor of a node is the number of arcs leaving the node. The backward branching factor of a node is the number of arcs entering the node. These factors provide measures of the complexity of graphs. When having time and space complexity of the search algorithms, we assume that the branching factors are bounded from above by a constant. The intuitive idea behind the generic search algorithm (figure 14), given a graph, a set of start nodes, and a set of goal nodes, is to incrementally explore paths from the start nodes by maintaining a frontier (or fringe) of paths from the start node that have been explored. The frontier contains all of the paths that could form initial segments of paths from a start node to a goal node.

FIGURE 14: Generic graph searching algorithm

```
1: Procedure Search(G,S,goal)
2:         Inputs
3:                 G: graph with nodes N and arcs A
4:                 S: set of start nodes
5:                 goal: Boolean function of states
6:         Output
7:                 path from a member of S to a node for which goal is true
8:                 or ⊥ if there are no solution paths
9:         Local
10:                Frontier: set of paths
11:        Frontier ←{⟨s⟩: s∈S}
12:        while (Frontier ≠{})
13:                select and remove ⟨s0,...,sk⟩ from Frontier
14:                if ( goal(sk)) then
15:                        return ⟨s0,...,sk⟩
16:                Frontier ←Frontier ∪{⟨s0,...,sk,s⟩: ⟨sk,s⟩∈A}
17:        return ⊥
```

Initially, the frontier is the set of empty paths from start nodes. At each step, the algorithm advances the frontier by removing a path $\langle s_0,\ldots,s_k\rangle$ from the frontier. If $goal(s_k)$ is true (i.e., s_k is a goal node), it has found a solution and returns the path that was found, namely $\langle s_0,\ldots,s_k\rangle$. Otherwise, the path is extended by one more arc by finding the neighbors of s_k. For every neighbors of s_k, the path $\langle s_0,\ldots,s_k,s\rangle$ is added to the frontier. This step is known as expanding the node s_k [5][6][7][8].

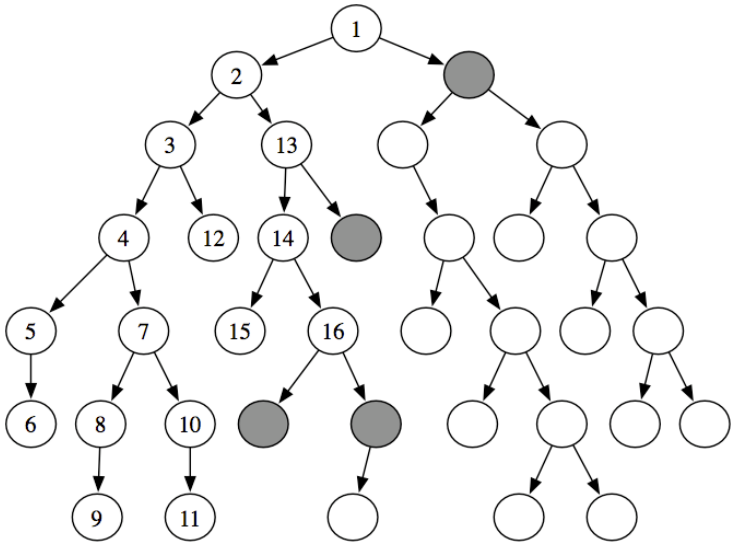
3.2 Search-Strategy Paradigms

A problem determines the graph and the goal but not which path to select from the frontier. The search strategy specifies which paths are selected from the frontier by modifying how the selection of paths in the frontier is implemented.

3.2.1 Depth-First Search Method

The first strategy is depth-first search (figure 15) [9]. The elements are added to the stack one at a time and taken off the frontier at any time is the last element that was added. Implementing the frontier as a stack results in paths in a depth-first manner means searching one path to its completion before trying an alternative path. This method involves backtracking; the algorithm selects a first alternative at each node, and it backtracks to the next alternative when it has pursued all of the paths from the first selection. Some paths may be infinite when the graph has cycles or infinitely many nodes, in which case a depth-first search may never stop.

FIGURE 15: The order nodes are expanded in depth-first search

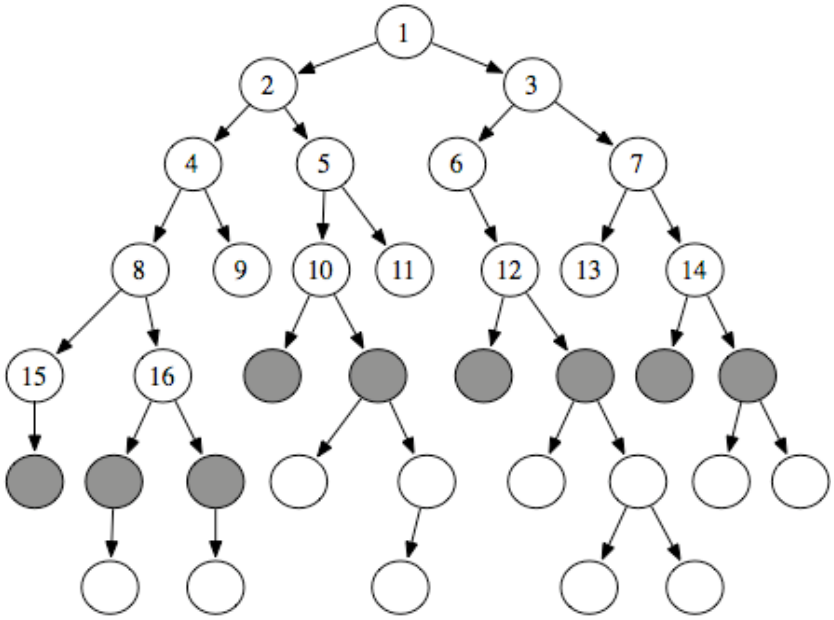


Because depth-first search is sensitive to the order in which the neighbors are added to the frontier, the ordering can be done either statically (so that the order of the neighbors is fixed) or dynamically (where the ordering of the neighbors depends on the goal). Nevertheless, the depth-first search algorithm does not specify the order in which the neighbors are added to the stack that represents the frontier, thus, the efficiency of the algorithm is extremely sensitive to the selected ordering. Depth-first search mechanism is an appropriate strategy when space is restricted/limited, many solutions with long path lengths exist or/and the order of the neighbors' node added to the stack can be fine-tuned so that solutions are found on the first try. It is not applied when infinite paths might exist or solutions exist at shallow depth. In addition, the depth-first search mechanism is the basis for a number of other algorithms, such as iterative deepening.

3.2.2 Breadth-First Search Method

In breadth-first search (figure 16) [10], frontiers are implemented as FIFO (first-in, first-out) queues, however it is not used quite often because of its space complexity. This approach implies that the paths from the start node are generated in order of the number of arcs in the path. One of the paths with the fewest arcs is selected at each stage. Breadth-first search is used if space is not a problem, searching to find the solution containing the fewest arcs and in cases where few solutions exist (at least one has a short path length) or infinite paths may exist. It is not recommended when all solutions have a long path length or there is some heuristic knowledge available.

FIGURE 16: The order in which nodes are expanded in breadth-first search



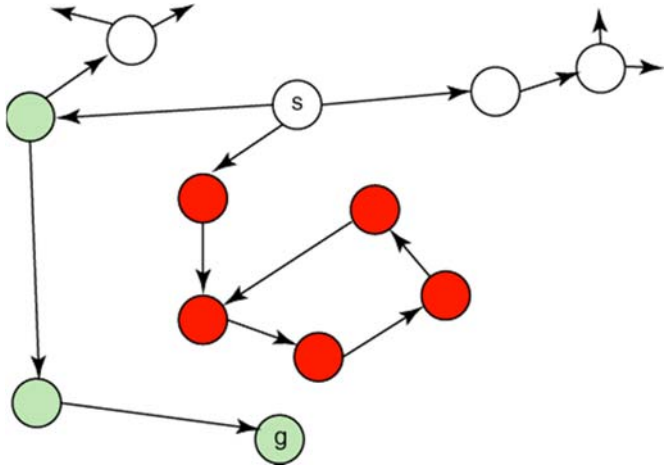
3.2.3 Lowest-Cost-First Search Method

When a non-unit cost is associated with arcs, in order to find the solution, we search for the solution that minimizes the total cost of the path [11][12]. Nevertheless, the previously referred search algorithms are not guaranteed to find the minimum-cost paths. The simplest search method that is guaranteed to find a minimum cost path is similar to breadth-first search; however, instead of expanding a path with the fewest number of arcs, it selects a path with the minimum cost. This is implemented by treating the frontier as a priority queue ordered by the *cost* function. If the costs of the arcs are bounded below by a positive constant and the branching factor is finite, the lowest-cost-first search is guaranteed to find an optimal solution if exists. The bounded arc cost is used to guarantee the lowest-cost search will find an optimal solution. Without such a bound there can be infinite paths with a finite cost. For example, there could be nodes $n_0,n_1,n_2,\ldots$ with an arc $\langle n_{i-1},n_i\rangle$ for each $i>0$ with cost $1/2^i$. Infinitely many paths of the form $\langle n_0,n_1,n_2,\ldots,n_k\rangle$ exist, all of which have a cost of less than 1. If there is an arc from n_0 to a goal node with a cost greater than or equal to 1, it will never be selected. This is the basis of Zeno's paradoxes that Aristotle wrote about more than 2,300 years ago. Like breadth-first search, lowest-cost-first search is typically exponential in both space and time.

3.3 Heuristic Search Strategies

A number of refinements can be made to the preceding strategies (i.e. cycle checking, multiple-path pruning, iterative deepening, branch and bounds and bidirectional, island-driven search). Nevertheless, dynamic programming can be used for path finding and for constructing heuristic functions [13][14]. All of the search methods in the preceding sub-section did not take into account the goal [15][16][17]. One form of heuristic information about which nodes seem the most promising is a heuristic function $h(n)$, which takes a node and returns a non-negative real number that is an estimate of the path cost from a node to a goal node. The heuristic function is a way to inform the search about the direction to a goal. A standard method to derive a heuristic function is to solve a simpler problem and to use the actual cost in the simplified problem as the heuristic function of the original problem [18][19][20]. A simple use of a heuristic function is to order the neighbors that are added to the stack representing the frontier in depth-first search. This search chooses the locally best path, but it explores all paths from the selected path before it selects another path. Although it is often used, it suffers from the problems of depth-fist search. Another way to use a heuristic function is to always select a path on the frontier with the lowest heuristic value (best-first search). It usually does not work very well; it can follow paths that look promising because they are close to the goal, but the costs of the paths may keep increasing.

FIGURE 17: A graph that is bad for best-first search



Consider the graph shown in figure 17, where the cost of an arc is its length. The aim is to find the shortest path from *s* to *g*. Suppose the Euclidean distance to the goal *g* is used as the heuristic function. A heuristic depth-first search will select the node below *s* and will never terminate. Similarly, because all of the nodes below *s* look good, a best-first search will cycle between them, never trying an alternate route from *s*. Table 3 gives a summary of the searching strategies presented so far.

TABLE 3: Summary of search strategies

Strategy	Selection from Frontier	Halts?[11]	Space
Depth-first	Last node added	No	Linear
Breadth-first	First node added	Yes	Exponential
Best-first	Globally minimal *h(p)*	No	Exponential
Lowest-cost-first	Minimal *cost(p)*	Yes	Exponential
Other (i.e. A)*	Minimal *cost(p)+h(p)*	Yes	Exponential

[11] "Halts?" means "Is the method guaranteed to halt if there is a path to a goal on a (possibly infinite) graph with a finite number of neighbors for each node and where the arc costs have a positive lower bound?" Those search strategies where the answer is "Yes" have worst-case time complexity which increases exponentially with the size of the path length. Those algorithms that are not guaranteed to halt have infinite worst-case time complexity. Space refers to the space complexity, which is either "Linear" in the path length or "Exponential" in the path length.

3.4 Combinatorial Optimization

A large number of well-known numerical combinatorial programming, linear programming (LP), and nonlinear programming (NLP) based algorithms are applied to solve a variety of optimization problems. In small and simple models, these algorithms were always successful in determining the global optimum. But in reality, many optimization problems are complex and complicated to solve using algorithms based on LP and NLP methods. Combinatorial optimization can be defined as the mathematical study of finding an optimal arrangement, grouping, ordering, or selection of discrete objects usually finite in number [24]. A combinatory optimization problem can be either simple of complex [25]. We call the problem simple if we can develop an efficient algorithm to solve for optimality in a polynomial time. If an efficient algorithm does not exist to solve for optimality in a polynomial time, we call the problem complex. An optimal algorithm to compute optimality complex problems requires a large number of computational steps which grows exponentially with the problem size. The computational drawback of such algorithms for complex problems requires the development of metaheuristic algorithms to obtain a (near) optimal solution [26].

Combinatorial optimization is such a topic that consists of finding an optimal object from a finite set of objects, in cases where exhaustive search is not feasible [27]. It is a subset of mathematical optimization that is related to operations research, algorithm theory, and computational complexity theory. Combinatorial optimization operates on the domain of those optimization problems, in which the set of feasible solutions is discrete or can be reduced to discrete, and in which the goal is to find the best solution [28]. Known paradigms involving combinatorial optimization are the traveling salesman problem ("TSP") and the minimum spanning tree problem ("MST"). Moreover, a number of applications in several fields, including artificial intelligence, machine learning, mathematics, auction theory, and software engineering [30].

3.5 Swarm Intelligence

A single ant or bee isn't smart, but their colonies are. A colony can solve problems unthinkable for individual ants, such as finding the shortest path to the best food source, allocating workers to different tasks, or defending a territory from neighbors. As individuals, ants might be tiny dummies, but as colonies they respond quickly and effectively to their environment. They do it with something called swarm intelligence. The collective abilities of such animals, none of which grasps the big picture, but each of which contributes to the group's success, seem miraculous even to the biologists who know them best. Yet during the past few decades, researchers have come up with intriguing insights. One key to an ant

colony, for example, is that no one's in charge. No generals command ant warriors. No managers boss ant workers. The queen plays no role except to lay eggs. Even with half a million ants, a colony functions just fine with no management at all—at least none that we would recognize. It relies instead upon countless interactions between individual ants, each of which is following simple rules of thumb. Scientists describe such a system as self-organizing and swarm intelligence is providing insights that help humans to understand better and manage such complex systems and manage complex systems. In other words, swarm intelligence (SI) is the collective behavior of decentralized, self-organized systems, natural or artificial as the concept is employed within the field of artificial intelligence[12]. SI systems consist typically of a population of simple agents interacting locally with one another and with their environment. The inspiration often comes from nature, especially biological systems. The agents follow very simple rules, and although there is no centralized control structure dictating how individual agents should behave, local, and to a certain degree random, interactions between such agents lead to the emergence of "intelligent" global behavior, unknown to the individual agents. Examples in natural systems of SI include ant colonies, bird-flocking, animal herding, bacterial growth, fish schooling and microbial intelligence. The most important ones are described as found in the sub-sections below [27][28][29].

3.5.1 Particle Swarm Optimization

Particle swarm optimization (PSO) is an optimization algorithm for dealing with problems in which a best solution can be represented as a point or surface in an n-dimensional space. Hypotheses are plotted in this space and seeded with an initial velocity, as well as a communication channel between the particles. Particles then move through the solution space, and are evaluated according to some criterion after each time instance. Over time, particles are accelerated towards those particles within their communication grouping which have better fitness values. The main advantage of such an approach over other global minimization strategies is that the particle swarm technique is impressively resilient to the problem of local minima [31][32][33][34].

3.5.2 Ant Colony Optimization

Ant colony optimization (ACO) is a meta-heuristic algorithm modeled on the actions of an ant colony. ACO is a probabilistic technique useful in problems that deal with finding better paths through graphs. Artificial 'ants'—simulation agents—locate optimal solutions by

[12] The expression was introduced by Gerardo Beni and Jing Wang in 1989, in the context of cellular robotic systems

moving through a parameter space representing all possible solutions. Natural ants lay down pheromones directing each other to resources while exploring their environment. The simulated 'ants' similarly record their positions and the quality of their solutions, so that in later simulation iterations more ants locate better solutions [35][36][37]. An extended analysis of the metaheuristic algorithm of Ant Colony Optimization algorithm is described as found on the next section of this chapter.

3.5.3 Artificial bee colony algorithm

Artificial bee colony algorithm (ABC) is a meta-heuristic algorithm simulates the foraging behavior of honey bees. The ABC algorithm has three phases: employed bee, onlooker bee and scout bee. In the employed bee and the onlooker bee phases, bees exploit the sources by local searches in the neighborhood of the solutions selected based on deterministic selection in the employed bee phase and the probabilistic in the onlooker bee phase. In the scout bee phase which is an analogy of abandoning exhausted food sources in the foraging process, solutions that are not beneficial anymore for search progress are abandoned, and new solutions are inserted instead of them to explore new regions in the search space. The algorithm has a well-balanced exploration and exploitation ability.

3.5.4 Artificial immune systems

Artificial immune systems (AIS) concerns the usage of abstract structure and function of the immune system to computational systems, and investigating the application of these systems towards solving computational problems from mathematics, engineering, and information technology. AIS is a sub-field of the biologically inspired computing, and natural computation, with interests in Machine Learning of the Artificial Intelligence research field.

3.5.5 Bat algorithm

Bat Algorithm (BA) is a swarm-intelligence-based algorithm, inspired by the echolocation behavior of bats. BA is automatically balances exploration (long-range jumps around the global search space to avoid getting stuck around one local maxima) with exploitation (searching in more detail around known good solutions to find local maxima) by controlling loudness and pulse emission rates of simulated bats in the multi-dimensional search space.

3.6 Conclusion

Metaheuristics are strategies that guide a search process which explore the search space to find a (near-) optimal solution. A metaheuristic strategy is generally applied to problems classified as NP-Complete but could also be applied to other combinatorial optimization problems. Metaheuristics are among the best known methods for a good enough and cheap (i.e., minimal computer time) solution for NP-Hard or NPComplete problems. As already mentioned before, typical examples where metaheuristics are used are the traveling salesman problem (TSP), scheduling problems, assignment problems, and vehicle routing problems (VRP); all types of problems falls under combinatory optimization problems. A metaheuristic algorithm is defined as: "An iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions." [38][39[40]] Metaheuristics are not problem-specific and may make use of domain specific knowledge in the form of heuristics. Application paradigms of metaheuristic approaches are the genetic algorithms, simulated annealing, Tabu search, memetic algorithms, ant colony optimization, particle swarm optimization, etc. The following sub-sections provide a detailed overview of Particle Swarm and Ant Colony Optimization, which are the main topic of interest to this dissertation as presented in the next chapters.

3.6.1 Particle Swarm Optimization Overview

In computer science, particle swarm optimization (PSO) is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. It solves a problem by having a population of candidate solutions, here dubbed particles, and moving these particles around in the search-space according to simple mathematical formulae over the particle's position and velocity. Each particle's movement is influenced by its local best known position but, is also guided toward the best known positions in the search-space, which are updated as better positions are found by other particles. This is expected to move the swarm toward the best solutions.

PSO is originally attributed to for simulating social behavior as a stylized representation of the movement of organisms in a bird flock or fish school. The algorithm was simplified and it was observed to be performing optimization. PSO is a meta-heuristic as it makes few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions. However, meta-heuristics such as PSO do not guarantee an optimal solution is ever found. More specifically, PSO does not use the gradient of the problem being

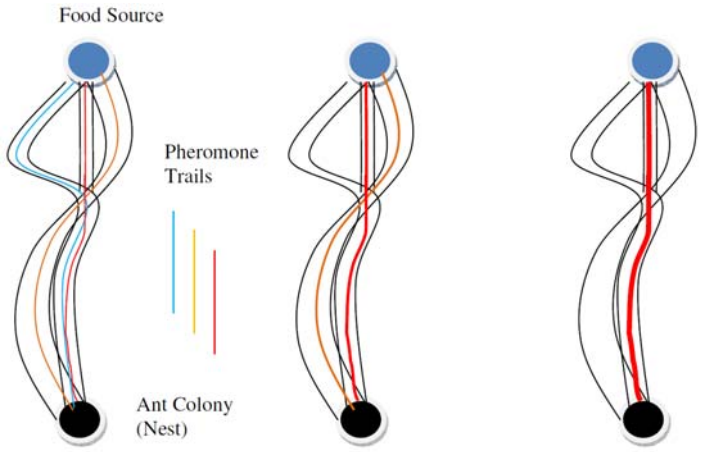
optimized, which means PSO does not require that the optimization problem be differentiable as is required by classic optimization methods such as gradient descent and quasi-newton methods.

A basic variant of the PSO algorithm works by having a population (called a swarm) of candidate solutions (called particles). These particles are moved around in the search-space according to a few simple formulae. The movements of the particles are guided by their own best known position in the search-space as well as the entire swarm's best known position. When improved positions are being discovered these will then come to guide the movements of the swarm. The process is repeated and by doing so it is hoped, but not guaranteed, that a satisfactory solution will eventually be discovered. The choice of PSO parameters can have a large impact on optimization performance. Selecting PSO parameters that yield good performance has therefore been the subject of much research. The PSO parameters can also be tuned by using another overlaying optimizer, a concept known as meta-optimization and they can be tuned for various optimization scenarios [31][32][34].

3.6.2 Ant Colony Optimization Overview

Ant Colony Optimization (ACO) is a metaheuristic approach proposed by Dorigo in 1992 to solve combinatory optimization problems. Inspired by the behavior of ants forming pheromone (e.g., a trace of a chemical substance that can be smelled by other) trails (figure 18) in search of food, ACO belongs to a class of algorithms which can be used to obtain good enough solutions in reasonable computational time for combinatory optimization problems [38][39][40]. Ants communicate with one another by depositing pheromones. Initially in search of food, ants wander randomly and upon finding a food source, return to their colony. On their way back to the colony, they deposit pheromones on the trail. Other ants then tend to follow this pheromone trail to the food source and on their way back may either take a new trail, which might be shorter or longer than the previous trail, or would come back along the previous laid pheromone trail. Also, on their way back, the other ants deposit pheromones on the trail. Pheromones have a tendency to evaporate with time. Hence, over a period of time, the shortest trail (path) from the food source to the colony would become more attractive and have a larger amount of pheromone deposited as compared with other trails. Initially, a single ant, called "blitz," goes from the colony to the food source via the blue pheromone trail. As time progresses, more and more ants either follow this blue trail or form their own shorter trail (red and orange trail). Eventually, the shortest trail (red) becomes more attractive and is taken by all the ants from the colony to the food source and the other trails evaporate in a period of time [35][36][37].

FIGURE 18: Ant Colony Optimization



The ACO replicates the foraging behavior of ants to construct a solution. The main elements in an ACO are ants which independently build solutions to the problem. For an ant k, the probability of it visiting a node j after visiting node i, depends on the two attributes namely:

- **Attractiveness:** It is the static heuristic value that never changes. In the case of VRP, it is calculated as inverse of arc length for shortest path problems and for other variants, it can depend on other parameters besides the arc length (e.g., in VRPTW it also depends on the current time and the time window limits of the customers to be visited (2004a).
- **Pheromone trails:** It is the dynamic component which changes with time. It is used to measure the desirability of insertion of an arc in the solution. In other words, if an ant finds a strong pheromone trail leading to a particular node, that direction will be more desirable than other directions.

3.7 References

[1] Nilsson, N.J. (1971). Problem-Solving Methods in Artificial Intelligence. McGraw-Hill, New York.

[2] Nilsson, N.J. (2009). The Quest for Artificial Intelligence: A History of Ideas and Achievements. Cambridge University Press, Cambridge, England.

[3] Nisan, N. (2007). Introduction to mechanisn design (for computer scientists). In N.Nisan et al. (Ed.),Algorithmic Game Theory, chapter 9, pp. 209-242. Cambridge University Press, Cambridge, England.

[4] Pearl, J. (1984). Heuristics. Addison-Wesley, Reading, MA.

[5] Pearl, J. (1988). Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Mateo, CA.

[6] Pearl, J. (2000). Causality: Models, Reasoning and Inference. Cambridge University Press.

[7] Hart, P.E., Nilsson, N.J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, 4(2): 100-107.

[8] Hart, T.P. and Edwards, D.J. (1961). The tree prune (TP) algorithm. Memo 30, MIT Artificial Intelligence Project, Cambridge MA.

[9] Korf, K.E. (1985). Depth-first iterative deepening: An optimal admissible tree search. Artificial Intelligence, 27(1): 97-109.

[10] Lawler, E.L. and Wood, D.E. (1966). Branch-and-bound methods: A survey. Operations Research, 14(4): 699-719.

[11] Dijkstra, E.W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1: 269-271. http://gdzdoc.sub.uni-goettingen.de/sub/digbib/loader?did=D196313.

[12] Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C. (2001). Introduction to Algorithms. MIT Press and McGraw-Hill, second edition.

[13] Minsky, M. (1986). The Society of Mind. Simon and Schuster, New York.

[14] Minsky, M. (1961). Steps towards artificial intelligence. Proceedings of the IEEE, 49: 8-30.http://web.media.mit.edu/ minsky/papers/steps.html.

[15] Bryce, D. and Kambhampati, S. (2007). A tutorial on planning graph based reachability heuristics. AI Magazine, 28(47-83): 1.

[16] Buchanan, B. and Shortliffe, E. (Eds.) (1984). Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. Addison-Wesley, Reading, MA.

[17] Felner, A., Korf, R.E., and Hanan, S. (2004). Additive pattern database heuristics. Journal of Artificial Intelligence Research (JAIR), 22: 279-318.

[18] Hart, P.E., Nilsson, N.J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, 4(2): 100-107.

[19] Tversky, A. and Kahneman, D. (1974). Judgment under uncertainty: Heuristics and biases. Science, 185: 1124-1131.

[20] Minton, S., Johnston, M.D., Philips, A.B., and Laird, P. (1992). Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. Artificial Intelligence, 58(1-3): 161-205.http://dx.doi.org/10.1016/0004-3702(92)90007-K.

[21] Pearl, J. (1984). Heuristics. Addison-Wesley, Reading, MA.

[22] Charikar, M., and Guha, S. [1999]: Improved combinatorial algorithms for the facility location and k-median problems. Proceedings of the 40th Annual IEEE Conference on Foundations of Computer Science (1999), 378-388

[23] Chudak, F.A., and Shmoys, D.B. [1998]: Improved approximation algorithms for uncapacitated facility location. In: Integer Programming and Combinatorial Optimization; Proceedings of the 6th International IPCO Conference; LNCS 1412 (R.E. Bixby, E.A. Boyd, R.Z. Rios-Mercado, eds.) Springer, Berlin 1998, pp. 180-194; to appear in SIAM Journal on Computing

[24] Levi, R., Shmoys, D.B., and Swamy, C. [2004]: LP-based approximation algorithms for capacitated facility location. In: Integer Programming and Combinatorial Optimization; Proceedings of the 10th International IPCO Conference; LNCS 3064 (G. Nemhauser, D.Bienstock, eds.), Springer, Berlin 2004, pp. 206-218

[25] Sviridenko, M. [2002]: An improved approximation algorithm for the metric uncapacitated facility location problem. In: Integer Programming and Combinatorial Optimization; Proceedings of the

10th International IPCO Conference; LNCS 2337 (W. Cook, A. Schulz, eds.), Springer, Berlin 2002, pp. 240-257

[26] Zhang, J., Chen, B., and Ye, Y. [2004]: Multi-exchange local search algorithm for the capacitated facility location problem. In: Integer Programming and Combinatorial Optimization; Proceedings of the 10th International IPCO Conference; LNCS 3064 (G. Nemhauser, D. Bienstock, eds.), Springer, Berlin 2004, pp. 219-233

[27] Kennedy, J., Eberhart, R. C., and Shi, Y., Swarm intelligence San Francisco: Morgan Kaufmann Publishers, 2001.

[28] Schoofs, L. and Naudts, B. Swarm intelligence on the binary constraint satisfaction problem. Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002), Honolulu, Hawaii USA. 2002

[29] El-Mounayri, H., Dugla, Z., and Deng, H. Predicting of surface roughness in end milling using swarm intelligence. Proceedings of the IEEE Swarm Intelligence Symposium 2003 (SIS 2003), Indianapolis, Indiana, USA. pp. 220-227, 2003

[30] Blum, C.; Roli, A. (2003). "Metaheuristics in combinatorial optimization: Overview and conceptual comparison" 35 (3). ACM Computing Surveys: 268–308.

[31] Hu, X., Eberhart, R. C., and Shi, Y. Particle swarm with extended memory for multiobjective optimization. Proceedings of the IEEE Swarm Intelligence Symposium 2003 (SIS 2003), Indianapolis, Indiana, USA. pp. 193-197, 2003

[32] Juang, C.-F., "A hybrid of genetic algorithm and particle swarm optimization for recurrent network design," IEEE Transactions on Systems, Man, and Cubernetics - Part B: Cybernetics, vol. accepted 2003.

[33] Hendtlass, T. and Randall, M. A survey of ant colony and particle swarm meta-heuristics and their application to discrete optimisation problems. Proceedings of The Inaugural Workshop on Artificial Life (AL 2001), pp. 15-25, 2001

[34] Coello Coello, C. A., Toscano Pulido, G., and Salazar Lechuga, M. An extension of particle swarm optimization that can handle multiple objectives. Workshop on Multiple Objective Metaheuristics, Paris, France. 2002

[35] M. Dorigo and T. Stützle, 2002. The ant colony optimization metaheuristic: Algorithms, applications and advances. Handbook of Metaheuristics, vol. 57, pp251-285

[36] M. Dorigo and T. Stützle, 2004. Ant Colony Optimization. MIT Press, Boston, MA.

[37] T. Stützle and H. H. Hoos, 2000. MAX-MIN Ant System. Future Generation Computer Systems, 16, 8, pages 889-914.

[38] I.H. Osman and J.P. Kelly, Metaheuristics. Theory and Applications, Kluwer, Boston, 1996.

[39] I.H. Osman and J.P. Kelly, Metaheuristics: An overview, in Metaheuristics: Theory and Applications, Kluwer, Boston, 1996. p.p. 1-21.

[40] Gudise, V. G. and Venayagamoorthy, G. K. Comparison of particle swarm optimization and back propagation as training algorithms for neural networks. Proc. IEEE Swarm Intelligence Symposium 2003 (SIS 2003), Indianapolis, Indiana, USA. pp. 110-117, 2003

4. PSO Algorithm Application: Solving a Ship Dispatching Problem

A solution for a ship dispatching problem with the usage of Particle Swarm Optimization algorithm is currently presented. The operational dimension of the problem is analyzed by introducing a Vehicle Routing Problem (VPR) formulation. In the present case study; a set of 13 ports of the Aegean Sea (including a depot port) is taken into paramount consideration.

4.1 Motivation and Contribution

The Vehicle Routine Problem (VPR) , which was first introduced by Dantzig and Ramser [1] , is a NP-hard combinatorial optimization problem . Recently, metaheuristic techniques, such as tabu search [2] , simulated annealing [3] and genetic algorithm [4] has been investigated. The last few years there has been a great interest in algorithms inspired by observation of natural phenomena. In particular in this paper, Particle Swarm Optimization (PSO) algorithm [5,6,7] , which was is investigated. The inspiration and the basic idea of PSO algorithm is the way a flock of birds behaves during the search for food. There is always a bird in the flock that can smell the food very well and better from the rest of the folk. Because the birds of a folk communicate with each other and transmit information, especially the good information, the birds will eventually flock to the place where food can be found. In currently presented study case, we apply PSO algorithm in the VPR that is given as a set of 13 ports (including a depot port) of the Aegean Sea in Greece [8].

4.2 The Vehicle Routing Problem (VRP)

The VPR [9] is a difficult Combinatorial Optimization Problem and can be described as follows: “A set of customers are to be serviced by a fleet of vehicles from a central depot. The locations of customers and the depot is given. Each vehicle has a limit capacity to carry goods and delivering or pick up goods from the customers. In the classic Vehicle Routing Problem we consider that all vehicles have the same capacity and goods are picked up from the customers.”

The VPR , from a mathematical point of view [8] , can be expressed via a complete weighted graph $G = (N, E)$, where $N = \{0,1,2 \dots, n\}$ is a set of nodes and $E = \{(i,j)|i,j \in N\}$ is a set of edges , and n_0 is the depot.

If we consider that the vehicle has unlimited capacity and is used to service a set of customers, then the problem reduces to a Travelling Salesman Problem (TSP). Thus, the VPR is the Θ-TSP problem. In Θ-TSP, the Θ-Salesman has to cover the given cities-nodes and each city-node must visited by exactly one salesman. In VPR, also the number of vehicles, Θ, is often considered as a minimization criterion in addition to total travelled distance. Generally, in the VPR three basic objectives can be distinguished:

- Minimize the number of vehicles used
- Minimize the total distance or time travelled
- Minimize a combination of number of vehicle used and total distance travelled.

4.3 Particle Swarm Optimization (PSO)

The Particle Swarm Optimization (PSO) algorithm [5,6,7] simulates the behavior of flock of birds . The algorithm uses a number of entities (particles) , where each entity represents a possible solution to the optimization problem. The basic characteristics of each particle are:

- x_i : the current position of particle i
- v_i : the current velocity of particle i
- y_i : the best position of particle i (personal best position)

The variable y_i is the best position in which the entity has been found, and therefore the best solution for objective function. The algorithm is initialized by a set of random particles (solutions) and then searches for the optimal by updating generations. In each iteration, each particle is updated by two best prices (if they achieved).
The first is particle's best solution achieved so far. This value is called pbest. The other "best" value is the "best" value that has been found so far by any particle of folk. After found the two best values in each particle updates its velocity(1) and position(2) by the following equations :

$$v_{ij}(t+1) = \omega_t v_{ij}(t) + c_1 r_{1j}(t)[p_{best,i}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[g_{best}(t) - x_{ij}(t)] \quad (1)$$

$$x_{ij}(t) = x_{ij}(t) + v_{ij}(t+1) \quad (2)$$

Where:

- j : the j^{th} dimension
- t : the number of iteration
- $v_{ij}(t)$: is the velocity of particle i (in dimension j if we have more than one dimensions) at time t.
- $x_{ij}(t)$: is the position of particle i (in dimension j if we have more than one dimensions) at time t.

- $p_{best,i}(t)$: is the particle's i best position.
- $g_{best}(t)$: is the global best position .
- c_1, c_2: are the acceleration coefficients (positive constants) which are used to determine the step size of each particle in each iteration.
- $r_{1j}(t), r_{2j}(t)$: are random numbers from uniform distribution (0,1) .
- ω_t: express the inertia

$$\omega_t = \omega_0 - \frac{\omega_0 - \omega_f}{T_{max}}\ t$$

Where:

- ω_0: is the initial value of inertia
- ω_f: is the final value of inertia
- T_{max}: is final number of iteration

4.4 PSO-2-VRP Application (encapsulation)

In the present sub-section the application of the Particle Swarm Optimization to the Vehicle Routing Problem is described in fully details.

4.4.1 Problem Formation

A subset of the Aegean Sea's islands is given in table 1. The numbers in cells of the Table 1 represent distance in nautical miles [10]. Every island has a specific demand for goods, provided by the available shipping fleet, which carries them from Piraeus port, designated as depot of every possible route to the islands. The goods are placed in small containers. For each route there is a corresponding traversal cost which is proportional to the distance between the islands the route connects.

Our goal is – under known supply and demand constraints – to minimize the total fuel costs and port dues, which expressed by the problem objective function:

$$min \sum_{ij} \left[n_{ij} L_{ij} C_F + n_{ij} C_{P_i} \right]$$

Where:

- n_{ij} : is the number of ships travelling from island i to island j.
- L_{ij}: is the distance of islands ij (miles)
- C_F : is the fuel's cost consumed per mile (in € per mile)

- C_{P_i}: is the fee of port i (in €)

For reasons of simplicity, the return cost of each ship after a successful route completion, is not included. The constraints of the optimization problem are [7]:

$$\sum_{j=1}^{i} w_{ij} - \sum_{k=1}^{k} w_{ik} + S_i - D_i = 0 :$$

Demand must be satisfied at all islands.

- w_{ij} : is the number of containers transported from island i to island.
- D_i : is the demand at the port.
- S_i : is the supply at the port i.

and,

- $n_{ij}Q - w_{ij} \geq 0, \forall\ (i,j)$
- $W_{ij} - (n_{ij} - 1)Q \geq 0, \forall (i,j)$
- $W_{ij} \geq n_{ij}, \forall, (i,j)$
- $n_{ij} \leq n_{max}$, $\forall\ (i,j)$
- $n_{ij} \geq 0$, $\forall\ (i,j)$: non negativity constraints
- $W_{ij} \geq 0, \forall (i,j)$: non negativity constraints
- n_{ij} , $integer$: integrality

4.4.2 Description of the Algorithm

Assumptions:

- Every particle (ship) start its route from a specified island (depot – Piraeus port) transporting cargo equals to their maximum carrying capacity.
- On every visit, the demand of the port for containers should be satisfied or all cargo should be unloaded.
- If a port's demand has been fulfilled, a ship can ignore it (but can use it as intermediate island/port, if the total cost is less)
- Every ship with no cargo left, returns to the depot (the return cost is not included, based on the problem definition).

The PSO Algorithm:

- Initialization of position x_i and velocity v_i for each particle (ship).
- Calculate initial cost.
- Define $pbest_i = x_i$ and calculate $gbest$.
- Set iteration count=1
- For every island j-1 to island j, given that :
  1. The costal route (j-1,j) exists,
  2. Demand at island j has not been fulfilled,
  3. Island j , is not revisited by ship i (if condition (2) is true then condition (3) is always true)
- Then ship i choose to move from island i to j based on its position
- For each ship
  - Calculate the velocity v_i
  - Calculate the position x_i
  - Calculate total cost
- Each particle/ship communicates with the rest in order to learn the next shortest route and compares it with his own one. If there is a better one, it replaces it.
- Update pbest and gbest.
- Continue iterations until gbest became pbest for all particles/ships.

4.5 Computational Experience

We now apply our problem formulation to the set of the 13 islands of the Aegean Sea, as shown in Table 4. Numbers on the cells of the Table 4 represents distance in nautical miles. The depot is the port of Piraeus.

TABLE 4: Set of 13 Islands of the Aegean Sea and distance in nautical miles from Piraeus

	Piraeus	Kea	Andros	Tinos	Mykonos	Naxos	Ios	Milos	Sifnos	Paros	Serifos	Kythnos	Syros
Piraeus	-	45	-	-	-	-	-	85	80	-	76	54	83
Kea	45	-	27	52	-	-	-	63	-	-	-	22	54
Andros	-	27	-	32	-	-	-	-	-	-	-	38	29
Tinos	-	52	32	-	11	-	-	-	-	26	-	-	13
Mykonos	-	-	-	11	-	22	-	-	-	26	-	-	18
Naxos	-	-	-	-	22	-	28	-	-	18	-	-	29
Ios	-	-	-	-	-	28	-	48	37	27	-	-	-
Milos	85	63	-	-	-	-	48	-	21	53	29	46	-
Sifnos	80	-	-	-	-	-	37	21	-	31	13	-	36
Paros	-	-	-	26	26	18	27	53	31	-	30	49	24
Serifos	63	-	-	-	-	-	-	29	13	30	-	29	38
Kythnos	54	22	38	-	-	-	-	46	-	49	29	-	41
Syros	83	54	29	13	18	26	-	-	36	24	38	41	-

The port demand and supply are shown in Table 5 [10].

TABLE 5a: Port Demand and Supply

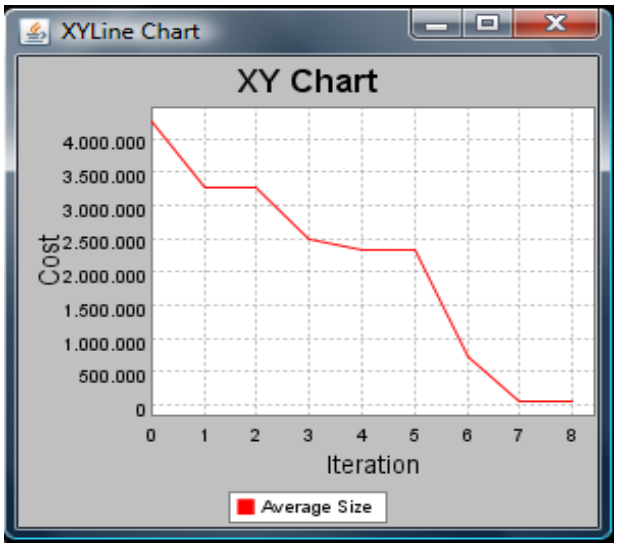
Port	Port Demand (in Containers)	Port supply (in Containers)
Piraeus	0	331368
Kea	8002	0
Andros	1684	0
Tinos	2992	0
Mykonos	6805	0
Naxos	15875	0
Ios	1847	0
Milos	26495	0
Sifnos	29477	0
Paros	48805	0
Serifos	23116	0
Kythnos	20251	0
Syros	5066	0
Total	190415	331368

TABLE 5b: Best Routes for minimal Cost, mumber of ships ∀ route number of Containers

Route	Containers	Ships
1-2	11533	39
1-4	26108	88
1-5	11871	40
1-6	15875	53
1-9	125028	417
2-3	1684	6
2-7	1847	7
4-11	23116	78
5-13	5066	17
9-8	26495	89
9-10	48805	163
9-12	20251	68

The final results (best routes for minimal cost, number of ships for each route and number of containers) are shown in Table 5b. Concluding, the graphical representation of the objective function optimization is shown in Figure 19.

FIGURE 19: Objective Function Optimization



4.6 Conclusions

The Particle Swarm Optimization (PSO) algorithm was applied in the vehicle routing problem. Particularly algorithm PSO was applied in a group of 13 islands of the Aegean Sea. Key Features of the PSO algorithm is that for the renewing of the particle's position, in each iteration of the algorithm, the particle uses two values. One value is the best position found during the execution of the algorithm (pbest), and the other is the best position found by any particle of the flock (gbest) when neighborhood is defined as the entire flock .PSO achieved to provide a solution in a small amount of time and with a minimum number of iterations.

4.7 References

[1] Dantzig, E.B , Ramser , J.H , The truck dispatching problem , Management Science , vol.6 , 1959 , pp:80-91

[2] Gendreau, M., Hertz, A., Laporte G., A tabu search heuristic for vehicle routing problem , Management Science , vol. 40, 1994 , pp:1276-1290

[3] Osman, I.H., Potts ,C.N., Simulates annealing for permutation flow shop scheduling . Omega , vol. 17 (6) , 1989 , pp:551-557.

[4] Baker , B.M., Ayechew , M.A., A genetic algorithm for vehicle routing problem. Computers and Operations Research , vol.30 (5) , 2003 , pp:787:800 .

[5] R.C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In Proceedings of the sixth international symposium on micro machine and human science, volume 43. New York, NY, USA: IEEE, 1995.

[6] A.P. Engelbrecht. Computational intelligence: An introduction. Wiley, 2007.

[7] J. Kennedy, R.C. Eberhart, et al. Particle swarm optimization. In Proceedings of IEEE international conference on neural networks, volume 4, pages 1942-1948. Perth, Australia, 1995.

[8] Alexandris , N., Fountas , C. , Vlachos , A., The Ant Colony System : Optimization for the logistics of marine cargo in the Aegean , Journal of Statistics and management Systems , vol.8(1) , 2005 , pp: 1-11.

[9] Christofides , N., Mingozzi , A. , Toth , P., The vehicle Routing Problem , in combinatorial Optimization , P. Toth , N. Cristofides , R. Mingozzi , and C. Sandi (Eds) , Tohn Wiley , New York , pp:315-338, 1989

[10] Ministry of Commercial Marine , Personal Communication, Greece, Statistic Analysis, 2015

5. Ant Colony Optimization Application: An extension to the VSP

ANT COLONY Optimization (ACO) algorithms have been applied successfully in Combinatorial Optimization Problems. In this chapter, we will focus in the definition of objective function for VSP and estimate the effect of varying of levels of initial pheromone quantities on the objective function using an extension of Ant Colony System (ACS) algorithm. JEL classification codes: C61, L92.

5.1 Motivation and Contribution

The Ant Colony Optimization (ACO) technique has emerged recently as a new meta-heuristic for hard combinatorial Optimization Problems. ACO algorithms have been inspired by colonies of real ants (Dorigo et al. 1996). An interesting and significantly important behavior of ant colonies is their ability to find the shortest path between their nest and food source (Doribo et al. 1997). The path taken by individual ants from the nest in search of a food source, is essentially random. Each ant leaves information on which path it has traversed by depositing a chemical substance called pheromone (Withrow 2000) on the ground. It has been observed that the more ants use a particular path, the more pheromone is deposited on that path and the more it becomes attractive to other ants seeking food. The artificial ants in ACO implement a randomized construction heuristic which makes probabilistic decisions as a function of artificial pheromone trails and possibly available heuristic information based on the input data of the problem to be solved. ACO algorithms have applied in a wide range of problems, incending the Traveling Salesman Problem (TSP), (Nararro et al. 1999), the Vehicle Routing Problem (VRP), (C. Fountas et al 2005), Quadratic Assignment Problem (QAP), (Maniezzo et al. 1999), and Sequential Ordering Problem (Gambardella et al. 1997).

5.2 Problem Formulation

The Vehicle Scheduling Problem (VSP) is defined on a complete directed graph $G=(V,A)$, where $V=\{0,1,\ldots,n\}$ is the set of vertices and A set contains allowed connections (arcs) and is the subset of VXV. Vertices $1,\ldots,n$, represent customers with customer i are associated a nonnegative demand d_i and a nonnegative service duration t_i. Vertex 0 is the depot at which is baset fleet of m homogenous vehicles of capacity q. To each arc (i,j) is associated a traveling cost c_{ij} calculated with the following equation:

$$c_{ij} = \sqrt{\left(i_x - j_x\right)^2 + \left(i_y - j_y\right)^2}$$

where i_x is the coordinate x for the customer i; and i_y is the coordinate y for customer i.

The VSP consist be design of m vehicle routes on G such as:

- each customer is visited only once,
- the total demand of any route, does not exceed the vehicle capacity q,
- the length of any route does not exceed a pre-set maximal route length L,
- in some version m is fixed a priori in other is a decion variable,
- the total cost of all vehicle routes is minimized.

The VSP is more complex than the traveling salesman problem (TSP) as every route in VSP is as TSP. Generally, if we have k vehicles in a VSP, then in order to get a best set of routes for the k vehicles the k number of TSP have to be solved.

5.3 The Mathematical Problem

In ACO algorithms an ant represent a vehicle. We concentrate on the way of choosing the next node by every ant. This selection is related with the pheromone quantity that has been displayed on every arc and also with the traverse cost of this arc. There are two possible actions: firstly an ant may follow, either it will choose with an absolute way the arc with the best efficiency (highest pheromone quantity low cost) and secondary an ant will choose according to a probability function between the candidate nodes. In the first option an ant k at the current position of the node i choose to move the next node j applying the state transition rule given by the following equation (Bonabeau et al. 1999).

$$j = \begin{cases} \arg\max\left\{\left[r_{ij}(t)\right] \times \left[n_{ij}\right]^{\beta}\right\} & \text{if } q \pounds q_0 \left(\text{exploration}\right) \\ J & \text{otherwise (exploration)} \end{cases} \qquad (1)$$

where $r_{ij}(t)$ is the amount of pheromone on arc (i,j) at time t, n_{ij} is the inverse of the distance between nodes i and j, called visibility, ß is a parameter which controls the relative weight of the visibility, and J is a node drawn by using the probabilities:

$$P_{ij}^{k}(t)=\begin{cases}\dfrac{r_{ij}^{..}(t)\times\left[n_{ij}^{..}\right]^{\beta}}{\sum_{r\hat{I}J_i^k}\left[r_{ir}(t)\times\left[n_{ir}\right]^{\beta}\right]} & \text{if } j\hat{I}J_i^k \\ 0 & \text{otherwise}\end{cases} \qquad (2)$$

where J_i^k is the set of the nodes to which ant k can move when being located in node i. Every ant will choose one of the two ways depending on a second probability function, which in the current state is zero for the first case and equal to one for the second case. In this paper we follow the second equation for the selection of the next arc. The equation with which the pheromone is updated in every arc after its traverse is:

$$r_{ij}(t+1)=(1-\rho)r_{ij}(t)+r_o \qquad (3)$$

where $1-\rho$, $\rho\in(0,1]$, is the pheromone decay coefficient, and r_0 is the initial amount of pheromone on arc (i,j). When a cycle ends, the value of the objective function of the problem is calculated using the following equation:

$$F=\sum\lambda_{ij}^{..}\times\delta_{ij}^{..-1} \qquad (4)$$

where:

- $\lambda_{ij}^{..}$: shows the number of traverse that took place for the arc joining nodes i and j.
- $\delta_{ij}^{..-1}$: is the cost of traverse for the arc.

The value of this objective function finally, arises from adding the cost of every arc used, multiplied with the number of times that is been traversed. Building a path network with the lowest cost arises from minimizing equation 4. In the paper parameter ß (beginning value $\beta=1$), and pheromone quantity by determined from the programmer user, whereas parameter ρ is by default zero which means that in this point we don't take any account of the evaporation.

5.4 The Proposed Algorithm

The steps of the proposed algorithm which compute the objective function end the effects of the varying the initial pheromone level in optimizing the objective function in VSP are:

1. Introduction of the problem graph, using the following data:
    a. Number of nodes
    b. Introduction of nodes using a four digit number, where the first digit represents the number of the starting node, the second digit the number of the finishing node, the third the cost of this path, and the fourth the load demand of the finishing node

2. Initialization with:
    a. Input of an initial pheromone quantity in all nodes
    b. Definition of load that an ant can transfer
    c. Definition of ant number, such that the total of demand from all nodes is just covered

3. Repeat until a satisfactory solution is found
    a. (3.1.) Repeat for every ant i
        i. (3.1.1.) If the load of an ant is zero, then it returns to the start
        ii. (3.1.2.) Else repeat for node j
            1. (3.1.2.1.)For node j-1 to node j and while:
                - The arc (j-1,j), exists.
                - There is load demand from node j
                - The ant i has not visited again j node from the moment of the last start (if (ii) is true then (iii) is always true
        iii. (3.1.3.) End for 3.1.2
        iv. (3.1.4.) The ant i selects the node to visit based on the probabilities that have already been calculated
    b. 3.2. End for 3.1
    c. 3.3. Calculation of the value of the objective function
    d. 3.4. Recalculation of the pheromone quantity in every arc
4. End (algorithm)

5.5 Case Studies

A computer program has been constructed to demonstrate how the concepts of the previous described algorithm can be applied (into practice), test, validate and finally and optimize (fine-tuning corrections) the proposed algorithm. The results taken from this program are illustrated using the input data of the Table 6.

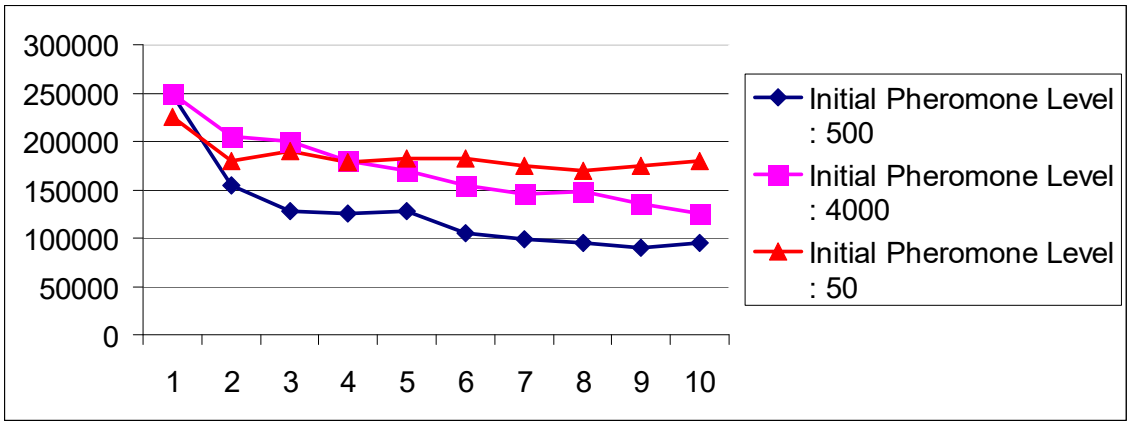
TABLE 6: The example of VSP (Input data form)

node	need
1	0
2	500
3	900
4	200
5	90
6	800
7	1000
8	50

The capacity of agent: 50

After several tests of the model, a graph is emerged (Figure 20, which has proved many interesting findings about the efficiency of the computer program according to our example and provided a mean to access the behavior of the process in a time horizon, which is measured in the example in cycles.

FIGURE 20: Initial pheromone level quantity Optimization in VSP



Observing figure 20, plenty of interesting facts come to light, about the initial pheromone quantity and the effect that these levels on the objective function for a VSP. Inputting a value of 4000(a high enough value) we noticed a relatively constant decrease of the objective function's value. Until the tenth cycle it was not detected that the objective function would

tend to stabilize or tend to some limit value. Inputting a very small value (50) as an initial pheromone quantity we saw that the value of the objective function after the end of the first cycle was lower than that of the other two cases, but a very small or rather no improvement was noticed after the end of the remaining cycles. The final value of the initial pheromone level was set at 500 and at that level we obtained the best results of the simulation process.

5.6 Conclusions

From the previous presented results we can derive to some conclusions that are not only related to the current study, but they are related in general with the Ant Colony Algorithm. It is obvious that the quality of the results is closely related with the initial values of the simulation process. A very important factor in every algorithm implementation of the ant colony algorithm is the correct definition of the initial pheromone quantity which is going to be used for examining whether some paths are favored against others. At this point is extremely difficult (if not impossible) to find a relatively better solution, so the lowest value that has been found at that moment, is assumed to be the best.

5.7 References

[1] Dorigo, M., Maniezzo V., Colorni, A., 1996, "The ant system – Optimization by a colony fo cooperating agents", IEE Transaction on Systems, Man, and Cybernetics 26, p.29-41.

[2] Dorigo, M., Gambardella, L. M., 1997, "Ant colonies for the travelling salesman problem" Biosystem 43(1), p.73-81.

[3] Withrow J., 2000, "Engineers Learn from Ants: The Application of chemical Ecology to Problem Solving" Department of Biagricultural Science & Pest Managent, Colorado State University.

[4] Nararro, G., Varela and Sinclair, M. C., 1999, "Ant Colony Optimization for Virtual – Wave length – Path Routing and Ware length Allocation" presented at Congress on Evolutionary Computation (CEC '99) Washington DC, USA.

[5] Fountas, C., Vlachos, A., 2005, "Ant Colonies Optimization (ACO) for the solution of the Vehicle Routing Problem – VRP" Journal of Information Optimization Sciences, vol.26, No 1, p.135-142.

[6] Maniezzo, V., Colorni, A., 1999, "The Ant system applied to the quadratic assignment problem" IEEE Transaction on Knowledge and Data Engineering, Vol.11, p.769-778.

[7] Costa, D., Hertz, A., 1997, "Ants can colour Graphs", J. op. Res. Soc., 48, p.295-305.

[8] Gambardella, L. M., Dorigo, M., 2000, "HAS-SOP: Hybrid Ant System for the Sequential Ordering Problem" Techical Reprot IDSIA 11-97, IDSIA, Lugano, Switzerland. To appear in INFORMS J. Comp.

[9] Bonabeau E., Dorigo M., Theraulaz G., 1999, "Swarm Intelligence, From Natural to Artificial Systems" Oxford University Press, p.46-49.

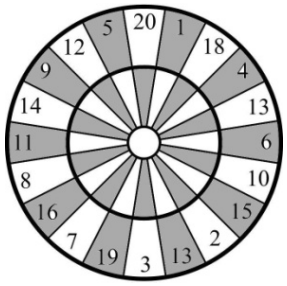
6. Comparison of the Metaheuristic ACS and MMAS Algorithms: An Optimized Dartboard Design Application

THE PROBLEM of optimally locating the numbers around a dartboard is a Combinatorial Optimization problem. In this paper, we're solving this problem using Ant Colony System and Max-Min Ant System (MMAS) algorithm. The algorithm reinforces local search in neighborhood of the best solution found in each iteration while implementing methods to slow convergence and facilitate exploration. Both algorithms have been proved to be very effective in finding optimum solution to hard combinatorial optimization problems.

6.1 Motivation and Contribution

The game of darts is one of the most popular games on the whole planet. It was invented during the 16th century but the method of score calculation belongs to Brian Gamlin (1896) as cited [1] .Then many variants were invented but the most important is the one where the players try to reduce the initial score value of 301 to 0 points [1]. The most common form of the dartboard is depicted in Fig. 21. The Dartboard design [2] is related to placing the numbers to sectors of a circular board [3,4,5,6].

FIGURE 21: Current Dashboard Design



Kohler [7] dealt with the designing of the dartboard and used Dynamic Programming in order to form a set of optimal strategies for some players' categories in which every player tries to reduce the initial score of 301 points. H. A. Eiselt and G. Laporte[8] studied the problem of optimally locating the numbers around a dartboard as a Traveling Salesman Problem (TSP) and as a Quadratic Assignment Problem (QAP) and proposed various dartboard designs. This paper examines the optimized placing of the numbers on the dartboard. For this purpose we use two algorithms from the family of Ant Colony Optimization algorithms (ACO) [9,10,11,12,13,14], Ant System algorithm[10] and Max-Min Ant System (MMAS) algorithm[9,15], and compare the results of each one.

6.2 Problem Formulation

Players have to aim at specific points and not at the whole dartboard. The points actually gained defined the standard deviation σ . Standard deviation is reverse proportional to the player's target accuracy [3]. The sector at which the player should aim at, gives 20 points Figure 1. Let π (k) be the number which is located at k position of the dartboard, beginning from any position arbitrarily and $\pi = (\pi(1), \pi(2), \ldots\ldots, \pi(20))$ be any permutation of the numbers 1,2,…,20. If a player aims at π(k) and gains $\pi(k+1)$ with probability p and π (k) with probability $1-2p$ the expected standard deviation from aimed points is [8]:

$$p[\pi(k-1) - \pi(k)] + p[\pi(k+1) - \pi(k)]$$

The aim is the maximization of the total expected standard deviation function, when the 20 numbers constitute targets to be gained with equal probabilities. There are two objective functions that have to be maximized for the total of permutations π (3).

$$Z_1 = \sum_{k=1}^{20} \{p[\pi(k-1) - \pi(k)]^2 + p[\pi(k+1) - \pi(k)]^2\}$$

And

$$Z_2 = \sum_{k=1}^{20} \{p|\pi(k-1) - \pi(k)| + p|\pi(k+1) - \pi(k)|\}$$

Since every term is calculated twice for every objective function while p is a constant, the objective functions are expresses as:

$$Z_1 = \sum_{k=1}^{20} [\pi(k+1) - \pi(k)]^2 \qquad (1)$$

$$Z_2 = \sum_{k=1}^{20} |\pi(k+1) - \pi(k)| \qquad (2)$$

6.3 AS and MMAS Algorithms Overview

Both algorithms were inspired by observation of the behavior of real ant colonies[11] and first were applied to the classic Travelling Salesman Problem (TSP) [15,16]. A set of agents , called artificial ants, cooperate to find good solutions to TSP using an indirect model of communication through pheromone trails, which they deposit on the edges of the TSP graph while constructing solutions. Most important part of both algorithms is the movement of the ants.

6.3.1 AS and MMAS Algorithms' Common Features

- n_{ij}: is the heuristic information which called visibility and express the desirability of edge (i,j) . It is defined as the inverse of the distance between i and j.
- τ_{ij}:is the pheromone trail and expresses the concentration of pheromone on edge (i,j) .
- Coefficients α, β are the parameters which control the relative influence of pheromone trail and heuristic information and they are defined by the user.

6.3.2 Ant-System (AS) Algorithm Analysis

Ant System algorithm [11,12] is the first member of the family of Ant Colony Optimization algorithms. The main characteristics AS algorithm are the following:

- Ant System algorithm uses a list (M_i^k) , known as tabu list, to store all the visited nodes. Initially the tabu list contains only the start node.
- $P_{ij}^k(t)$: is the stochastic state transition rule which expresses the probability that the kth ant choose the edge (i, j) is in iteration t.

$$P_{ij}^k(t) = \begin{cases} \dfrac{[\tau_{ij}(r)]^a [n_{ij}]^\beta}{\sum [\tau_{ij}(t)]^a [n_{ij}]^\beta} & if\ j \notin M_i^k \\ 0 \qquad if\ j \in M_i^k \end{cases}$$

- If coefficient $\alpha = 0$ the probability of selection are proportional to $[n_{ij}]^\beta$
- If coefficient $\beta = 0$ only pheromone amplification is at work

At the end of each iteration if each ant has generated a solution , the pheromone updating equation is given from:

$$\tau_{i,j}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=`1}^{m} \Delta\tau_{ij}(t)$$

where:

- ρ: is pheromone's evaporation rate coefficient , $\rho \in [0,1]$
- m: is the number of ants
- $\Delta\tau_{ij}^k(t)$: is the amount of pheromone the k^{th}deposited and is defined from:

$$\Delta\tau_{ij}^k(t) = \begin{cases} \dfrac{Q}{L_k(t)}, if\ (i,j) \in M_i^k \\ 0\ if\ (i,j) \notin M_i^k \end{cases}$$

where:

- Q: is a constant which representing the pheromone addition factor
- $L_k(t)$: tour length for the kth ant

6.3.3 Max-Min Ant System (MMAS) Algorithm

MMAS algorithm [9,15,16,18] is a direct improvement over AS algorithm [11,12,19]. The main modification by MMAS with respect to AS follow below:

1. Only one single ant is allowed to add pheromone in each iteration, in order to exploit the best solutions found. This ant may be the one which found the best solution in the current iteration (iteration-best ant) or the one which found the best solution from the beginning of the trial (global-best ant).
2. To avoid stagnation of the search , the range of the pheromone trails on each solution component is limited to an interval $[t_{min}, t_{max}]$.
3. The pheromone trails are initialized to the upper trail limit (τ_{max}), which causes a higher exploitation at the start of the algorithm.

The solutions in MMAS algorithm are constructed in exactly the same way as in AS algorithm. The decision for the next edge to be followed is given by the probability:

$$P_{ij(t)} = \frac{[\tau_{ij}(t)]^a [n_{ij}]^b}{\sum_{l \in N_i^*} [\tau_{il}(t)]^a [n_{il}]^b} \text{ , if } j \in N_i^*$$

where N_i^* : is the set of cities ant not visited yet. The pheromone update rule is given by the equation:

$$\tau_{i,j}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}^{best}$$

where $(1-\rho)$: is a parameter that models pheromone evaporation with $\rho \in [0,1)$.

$$\Delta\tau_{ij}^{best} = \frac{1}{f(s^{best})}$$

with $f(s^{best})$ is a solution cost of iteration-best.

When the algorithm converges, then the smoothing of pheromone trail mechanism can be activated , which increases pheromone levels depending on the differences $\tau_{ij}{}^* < \tau_{ij}$, before and after the smoothing scores rounded, so that the selection possibility of trails with low pheromone levels is minimized .

This can take place as following:

$$\tau_{ij}{}^{*} = \tau_{ij} + \delta\left(\tau_{max}(t) - \tau_{ij}(t)\right)$$

Where δ is a parameter set by the user and $0 \leq \delta \leq 1$. For = 1 , there is a reinitialization of pheromone levels, and for $\delta = 0$, this mechanism becomes inactive. (This mechanism is used mainly in executions with a large number of iterations).

6.4 AS and MMAS algorithm Implementation for Optimal location

Firstly we present the assumption for each algorithm followed by the steps used ton solver our problem.

6.4.1 Ant System Algorithm Assumptions

20 ants were used same as the numbers population that must be located on the dartboard. 100 iterations of the algorithm were executed. The value of the parameter Q is equal to 1.

6.4.2 Ant System Algorithm Step-Analysis

- **Step 1:** Define the pheromone giving it a large value in all arcs of the graph through which the problem is presented. The number of the ants (20) and the number of the iterations (100) are defined. Every ant is located on a different node and the cost for all the arcs is calculated (C_{ij}) .

- **Step 2:** Using the transition rule (3), the next number that will be located on the dartboard is evaluated for every ant. Step 2 is repeated until all ants fulfill their trip.

- **Step 3:** The cost of all trips is calculated. This is a measure expressing how much a failure will cost a player. The maximum cost is registered if it gives a better solution than the one that is already found.

- **Step 4:** Pheromone is updated at all trails used by the ants.

- **Step 5:** The process is repeated from Step 2 until 100 iterations are executed.

6.4.3 Max-Min Ant Algorithm Assumptions

20 ants were used for the implementation of the program, as many as the numbers that must be located on the dartboard. 100 iterations of the algorithm were executed. The upper limit of pheromone is calculated by:

$$\tau_{max}=\frac{1}{\rho f(s^{opt})}$$

where $f(s^{opt})$ is the solution cost in the corresponded iteration. The lower limit of pheromone is the limit of the sequence:

$$(\tau_{ij})_{\ni} = \sqrt[\ni]{(0{,}001)^{\ni} + \ni^{\ni+1}}\,\sin\frac{0{,}01}{\ni} \quad ,\forall \ni \in N^{*}$$

The parameter a, b, τ_0 and $\rho \in [0,1)$ is determined by the user.

6.4.4 Max-Min Ant Algorithm Step-Analysis

- **Step 1:** Define the pheromone giving it a large value in all arcs of the graph through which the problem is presented. The number of the ants (20) and the number of the iterations (100) are defined. Every ant is located on a different node and the cost for all the arcs is calculated (C_{ij}) .

- **Step 2:** Using the transition rule, the next number that will be located on the dartboard is evaluated for every ant. Step 2 is repeated until all ants fulfill their trip.

- **Step 3:** The cost of trips is calculated. This is a measure expressing how much a failure will cost a player. The maximum cost is registered if it gives a better solution than the one that is already found.

- **Step 4:** Renew pheromone using the update rule , and also $\tau_{max} - \tau_{min}$ levels.

- **Step 5:** The process is repeated from Step 2 until 100 iterations are executed.

- **Step 6:** (Optional Step). When the algorithm seems to converge, smoothing of pheromone trail can take place.

6.5 Case Studies - Optimization

Both algorithms (AS and MMAS) are implemented to solve the problem of locating the numbers on the dartboard in an optimal way, and compare their results. Each algorithm ran for 4851 reps to achieve all the possible combinations of parameters α,ß,ρ. The range of parameters that was selected, is for α=0…20 with step 1, for ß=0...20 with step 1 and for ρ=0...1 with step 0.1. We decide to include the extreme values of evaporation (ρ) 0 and 1, to observe the behavior and the results of both algorithms. Each algorithm's best costs for z_1,z_1 are shown in Table 7. For function z_2 both algorithms for most combination of α,ß,ρ gives constant best cost value equal to Cost=199. For function z_1 AS gives the best solution for ρ=0 , and it can consider as special case , since this means that the pheromone is accumulate at the trails without being evaporated. As a result, these trails reinforced continuously when the others progressively excluded by the colony. Each algorithm found his 9 best solutions in most cases for more than one combination of the parameters.

TABLE 7: AS and MMAS algorithm Best Cost Solutions

AS algorithm Best Cost Solutions		**MMAS algorithm Best Cost Solutions**
Cost for z_1	**Cost for z_2**	**Cost for z_1**
2623	199	2632
2622	199	2630
2621	199	2629
2618	199	2628
2617	199	2627
2616	199	2626
2614	199	2625
2613	199	2624
2612	199	2623

For the implementation of AS and MMAS algorithms, in Tables 8 & 9 shown the multitude of combinations which gives the nine best solutions.

TABLE 8: AS Multitude of Combinations

AS algorithm : Multitude of (α,β,ρ)'s combinations for the 9 best solutions	**Cost for z_1**
2	2623
2	2622
5	2621
1	2618
5	2617
13	2616
2	2614
3	2613
13	2612

TABLE 9: MMAS Multitude of Combinations

MMAS algorithm: Multitude of (α,β,ρ)'s combinations for the 9 best solutions	Cost for z1
2	2632
47	2630
1152	2629
57	2628
3	2627
73	2626
14	2625
65	2624
73	2623

The best combinations of parameters (α,ß,ρ), based on the number of iterations it took to find the nine best costs for z_1,z_2 ,is presented in Tables 10 & 11.

TABLE 10: Summary of search strategies

AS algorithm: Multiple combination of (α,β,ρ) values for the nine best costs	Cost for z_1	Cost for z_2
α=1,ß=9,ρ=0 || α=2,ß=18,ρ=0	2623	199
α=1,ß=9,ρ=0.3 || α=2,ß=18,ρ=0.3	2622	199
α=1,ß=7,ρ=0.1 || α=2,ß=14,ρ=0.1	2621	199
α=1,ß=20,ρ=0.2	2618	199
α=1,ß=12,ρ=0.1 || α=2,ß=14,ρ=0.3	2617	199
α=1,ß=18,ρ=0.2 || α=2,ß=20,ρ=0.4	2616	199
α=1,ß=15,ρ=0.2	2614	199
α=1,ß=8,ρ=0.1|| α=2,ß=16,ρ=0.1	2613	199
α=1,ß=17,ρ=0.1|| α=1,ß=18,ρ=0.4	2612	199

TABLE 11: Summary of search strategies

MMAS algorithm: Multiple combinations of (α,β,ρ) values for 9 best costs	Cost for z1	Cost for z2
α=7,ß=18,ρ=0.2|| α=7,ß=18,ρ=0.3	2632	199
α=4,ß=7,ρ=0.2|| α=3,ß=5,ρ=0.3	2630	199
α=2,ß=1,ρ=0.1|| α=4,ß=2,ρ=0.2	2629	199
α=3,ß=4,ρ=0.4||	2628	199
α=8,ß=15,ρ=0.1|| α=9,ß=17,ρ=0.1	2627	199
α=6,ß=11,ρ=0.1|| α=1,ß=2,ρ=0.3	2626	199
α=5,ß=8,ρ=0.2|| α=9,ß=19,ρ=0.3	2625	199
α=4,ß=5,ρ=0.2|| α=13,ß=17,ρ=0.9	2624	199
α=20,ß=1,ρ=0.1|| α=8,ß=15,ρ=0.2	2623	199

Finally in Tables 12 &13 are presented, the nine (9) best solutions for the numbers allocation on the dartboard.

TABLE 12: (9) Nine best solutions for AS

Best Combinations of (α,β,ρ)'s values for AS	**Cost for z_1**	**Cost for z_2**	**Solution**
α=1,β=9,ρ=0	2623	199	13,5,18,2,20,1,19,3,17,4,16,6,15,7,14,8,12,9,11,10
α=1,β=9,ρ=0.3	2622	199	12,7,18 ,1,20, 2, 19, 3, 17, 4,16 ,5, 15 ,6,14,8,13 ,9,11,10
α=1,β=7,ρ=0.1	2621	199	8,13,7,16,5,17,4,18,2,20,1,19,3,15,6,14,9,12,10,11
α=1,β=20,ρ=0.2	2618	199	12,8,16,3,18,1,20,2,19,5,17,4,15,6, 14,7,13,9,11,10
α=1,β=12,ρ=0.1	2617	199	13,7,17,2,20,1,19,3,18,4,16,5,15,6,14,9,12,8,11,10
α=2,β=20,ρ=0.4	2616	199	12,8,18,1,20,2,19,3,17,4,16,5,15, 6,14,7,13,9,11,10
α=1,β=15,ρ=0.2	2614	199	12,9,17,3,19,2,20,1,18,4,16,5,15,6,14,7,13,8,11,10
α=1,β=8,ρ=0.1	2613	199	13,4,20,1,19,2,18,3,17,5,16,6,15,7,14,8,12,9,11,10
α=1,β=17,ρ=0.1	2612	199	12,4,20,1,19,2,18,3,17,5,16,6,15,7,14,8,13,9,11,10

TABLE 13: (9) Nine best solutions for MMAS

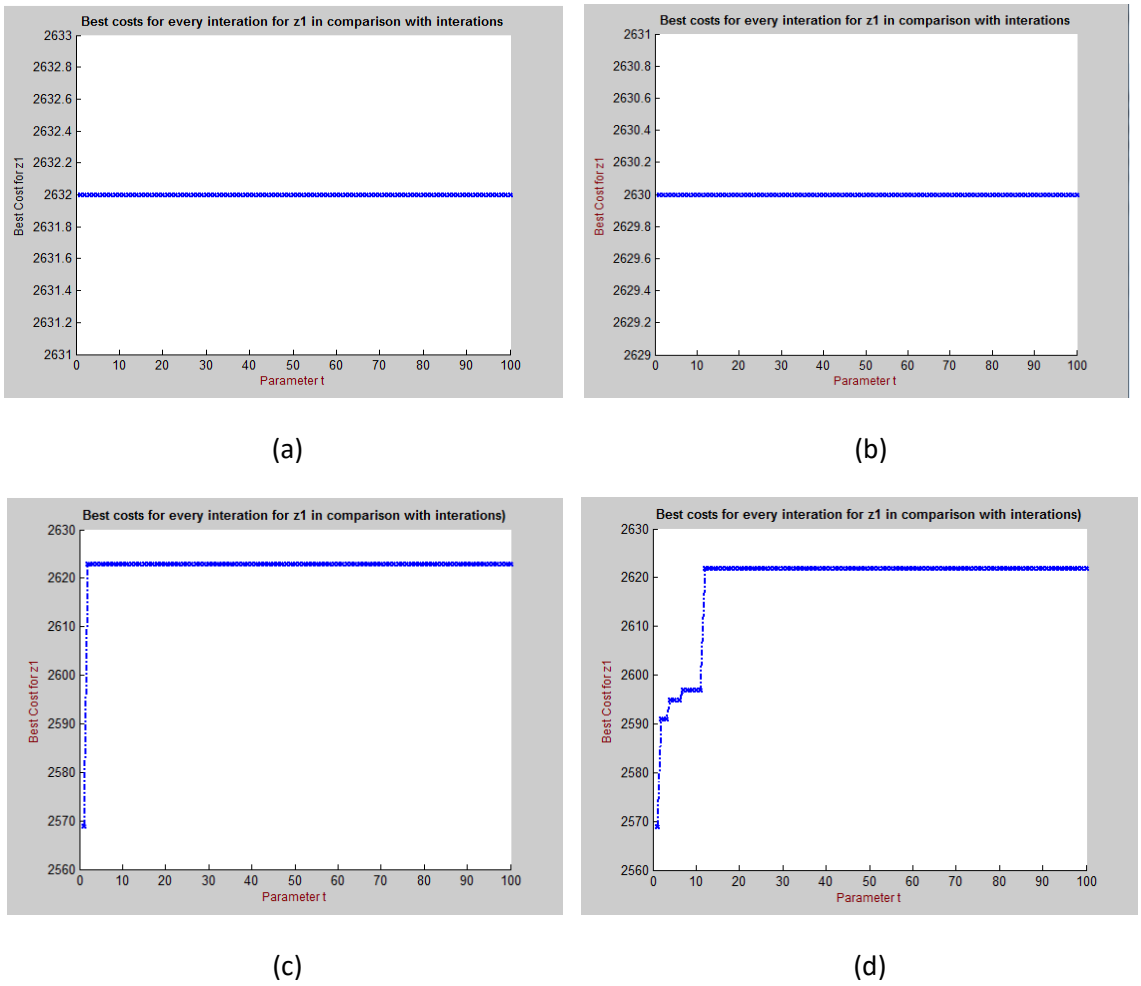
Best Combinations of (α,β,ρ)'s values for MMAS	**Cost for z_1**	**Cost for z_2**	**Solution**
α=7,β=18,ρ=0.2	2632	199	12,7,16,4,18,1,20,2,19,3,17,5,15,6,14,8,13,9,11,10
α=4,β=7,ρ=0.2	2630	199	12,8,14,6,18,1,20,2,19,3,17,4,16,5,15,7,13,9,11,10
α=2,β=1,ρ=0.1	2629	199	11,9,12,6,17,3,20,1,19,2,18,4,16,5,15,7,14,8,13,10
α=3,β=4,ρ=0.4	2628	199	12,8,14,6,15,4,20,1,19,2,18,3,17,5,16,7,13,9,11,10
α=8,β=15,ρ=0.1	2627	199	13,7,14,6,16,3,19,1,20,2,18,4,17,5,15,8,12,9,11,10
α=6,β=11,ρ=0.1	2626	199	12,8,14,4,19,1,20,2,18,3,17,6,16,5,15,7,13,9,11,10
α=5,β=8,ρ=0.2	2625	199	11,7,13,8,14,6,17,3,19,1,20,2,18,4,16,5,15,9,12,10
α=4,β=5,ρ=0.2	2624	199	12,8,14,4,19,2,20,1,18,3,17,6,16,5,15,7,13,9,11,10
α=8,β=15,ρ=0.2	2623	199	13,7,15,5,19,2,20,1,18,3,7,4, 16,6,14,8,12,9,11,10

6.6 Conclusions

The problem is referred to the Dartboard game where 20 numbers, ranging from 1 to 20 must be located in such an order around the dartboard that the players' failure is maximized. AS and MMAS algorithms were used to solve the problem. MMAS proved to be much faster and provide better solutions. Both algorithms tested in a desktop with 8 GB ram and an Intel I7-Core Processor. We timed both algorithms for the total of 4851 combinations of α, ß, ρ and AS needed sec when MMAS need. Both algorithms gives for the objective function z_2 best value, constant and equal cost=199, independently, of the values of the parameters α, b and ρ. For

the objective function $z_1$the optimum value MMAS provides a better best cost Cost=2632 when AS gives Cost=2623 (under special circumstances, when the evaporation coefficient is equal to zero).

FIGURE 22: AS & MMAS Evaluation Performance (Costs)



Also it was noticed that MMAS find the best cost for z_1 from the first iterations (figure22.a), when AS need much more(figure 22.b). In generally for most combinations' of the parameters (α,β,ρ) MMAS found the best costs almost immediately when AS needed much more iterations and in some occasion almost two third of them. This becomes more obviously when we compare the graphs of second best cost of each algorithm in comparison with the iterations (figure 22.c & 22.d). As we noticed MMAS achieve the best cost from the start, when AS. Finally it was noticed, that the best values of the cost were derived not exclusively from a particular combination of the parameters α, b and ρ, but for a multitude of their combinations.

6.7 References

[1] Encyclopedia Brittanica , Helen Hemingway Benton , 15th edition, Micropedia , Chicago , Vol. 3 , p.385. 1983.

[2] K. Selkirk, (1976) Re-designing the dartboard, Math. Gazette, vol. 60, pp. 171–178.

[3] D. Singmaster, (1980) Arranging a dartboard, IMA Bulletin, vol. 16, pp. 93–97.

[4] C.C. Chao, W.Q. Liang , (1992) Arranging n distinct numbers on a line or a cicle to reach extreme total variations , European J. Combin, 13, pp.325-334

[5] G.L. Cohen, E. Tonkes, (2001) Dartboard arrangements, Electron. J. Combin., vol. 8, R4.

[6] S.A Curtis, (2004) Darts and hoopla board design, Info. Proc. Letters, vol. 92, pp. 53–56.

[7] D. Kohler. (1982). Optimal strategies for the game of darts. Journal of the Operational Research Society, Vol 33 , pp. 871–884

[8] H.A. Eiselt , G. Laporte (1991) A combinatorial optimization problem arising in dartboard design, J.Oper. Res. Soc., vol. 42, pp.113–118.

[9] T. Stützle , H.H. Hoos , MAX-MIN Ant System , Future Generation Computer System , Future Generation Computer System , 16(8), (2000) , pp. 889-914.

[10] M. Dorigo, V. Maniezzo, A. Colorni, The ant system: optimization by a colony of cooperating agents, IEEE Trans. Systems Man Cybernet. B 26 (1996) 29–42.

[11] M. Dorigo, G. Di Caro, L.M. Gambardella, Ant algorithms for distributed discrete optimization, Artificial Life 5 (1999) 137–172.

[12] M. Dorigo, G. Di Caro, The ant colony optimization meta-heuristic, in: D. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, London, 1999, pp. 11–32.

[13] Kazyarov, A.A. and Kureichik, V.M. (2010) 'Ant colony optimization algorithms for solving transportation problems', Journal of Computer and Systems Sciences International, Vol. 49, No. 1, pp.33–40

[14] Ostfeld, A. (Ed.) (2011) Ant Colony Optimization-Methods and Applications, Published by InTech, Croatia.

[15] T. Stützle, H.H. Hoos, The MAX–MIN ant system and local search for the traveling salesman problem, Evolutionary Computation, IEEE International Conference, 1997

[16] M. Dorigo ,T. Stützle. Ant Colony Optimization. MIT Press,2004,25-38

[17] A. Vlachos Meta-Heuristics algorithms solving combinatorial problems , Ph.D. Thesis , University of Piraeus , Greece, 2006

[18] Vlachos, A. and Trichas, I. (2013) 'Max-min ant system algorithm for dartboard design', Int. J. Computational Intelligence Studies, Vol. 2, Nos. 3/4, pp.367–378.

[19] Argyri E and Vlachos A. ,'A metaheuristic algorithm is used for dartboard design ' , Int J. of Management Science and Engineering Management , Vol.4 (2009) ,Nos. 3, pp 198-204

[20] A.H. Borzabadi, H.H. Menhe , Ant Colony Optimazation for Optimal Control Problems, Journal of Information and Computing Science, No.4, Vol.4, pp.259-264, 2009.

7. Conclusions and Future Directions

The research goals addressed in this doctoral dissertation are related to the extensive research and development efforts are required when applying combinatorial optimization in intelligent multi-agent systems, in order to achieve cooperation, interoperability and sustainability in heterogeneous and complex existing or future designs of industrial, aerospace, robotic systems or/and other cyber-physical system.

In the first part of this Doctoral Dissertation, consisted of chapters 1, 2 and 3, the metaheuristic approaches in the area of AI algorithms and specifically the particle swarm and ant colony optimization to solve combinatorial optimization problems were presented and discussed.

In the second part, three implementations of metaheuristic algorithms are presented in details. In chapter 4, the problem of dartboard game within the spectrum of metaheuristic algorithms was described and the Ant Colony System and Max-Min Ant System algorithm as a metaheuristic strategy that guide the search process were applied in the effort to reinforce local search in neighborhood of the best solution found in each iteration.

In chapter 5, the Ant Colony Optimization technique is applied as a new metaheuristic for hard Combinatorial Optimization Problems. Thus, the implementation of a randomized construction heuristic extension of Ant Colony System algorithm in the Vehicle Scheduling Problem is proposed and studied. Moreover, the effect of varying levels of initial pheromone quantities are estimated on this study case.

In chapter 6, the application of a ship dispatching real-life problem is studied by implementing the Particle Swarm Optimization algorithm and analyze the operational dimension of the problem by introducing a VPR formulation based on a set of 13 ports of the Aegean Sea

All of the implementations, applications and contributions of present Thesis are studied within the spectrum of the Artificial Intelligence domain.

Nevertheless, any of the proposed enhanced interoperable and cooperative solutions provides a contribution towards the combinatorial problems that the industry and the research community might have highly acknowledged. As cyber-physical systems evolve, efficiency, quality, safety, security, trustworthiness and optimized solutions are major concerns for the industry, developers and researchers.

Since, MATALAB is one of the most common and effective tools to improve all aspects of the systems a wider deployment of multi-agent systems with optimum intelligence set them capable to provide cross border mobility would be of great challenge addressing reliability, time-sensitivity, quality and continuity. Thus, the sustainability of the entire ICT domain, including cyber-physical systems and SCADA, into the era of Internet of Things needs to be re-examined from the aspect of AI. Turing's original question "if machines can think" not only still remains, but also expands "if they can think; which types of machines should?"

In addition, albeit the concept of intelligent multi-agent systems is not new, their commercial success over the recent years played a major role in the aerospace domain. Over the next years, as future global aerospace transportation systems will expand beyond known capabilities, the need of AI in resource provisioning (pooling management) will be necessary.

Therefore, the transition of the ICT era to the "post-PC" era of an intelligent cyber-space and the cyber-physical microcosm, in order to be sustained fully connected and self-optimized need to identify both technological and non-technological issues related with the evolution of the AI. Related technological issues include:

- Optimization of scaled interoperability and elastic scalability, which is currently restricted due to the inefficiently implemented resource capabilities;
- Efficient handling of big data due to the diversity of data, leading to consistency and efficiency issues;
- Rapid designs and development simplicity in the solutions provided.

Non-technological issues include:

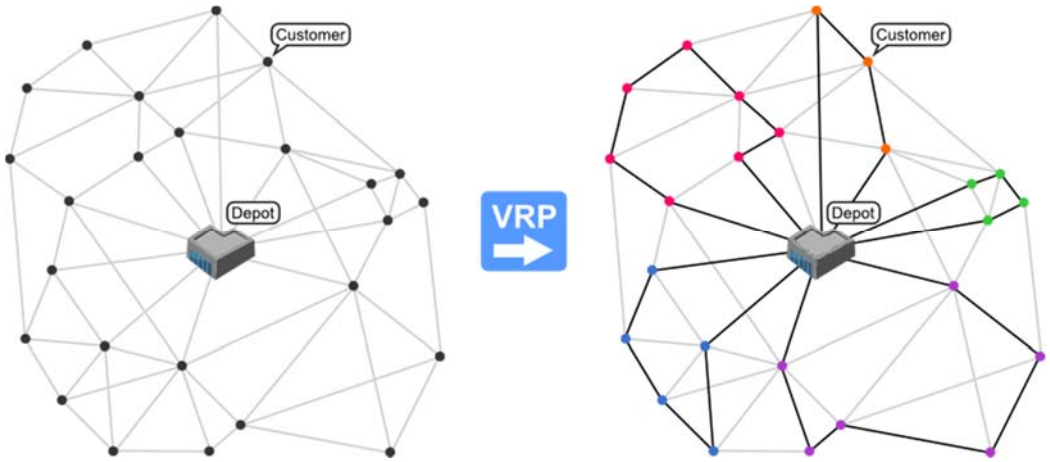
- economic aspects which cover knowledge about when, why and how to use AI in computing systems and technology;
- aspects related to green IT and "green capabilities" by reducing unnecessary power consumption
- worldwide harmonized regulations of protecting the environment

APPENDIX-I

Vehicle Routing Problem (VRP) – Generic Approach and Algorithmic Implementation

The vehicle routing problem (VRP) is a combinatorial optimization and integer programming problem seeking to service a number of customers with a fleet of vehicles in the fields of transportation, distribution, and logistics. The Vehicle Routing Problem (VRP) is a generic name given to a whole class of problems in which a set of routes for a fleet of vehicles based at one or several depots must be determined for a number of geographically dispersed cities or customers. The objective of the VRP is to deliver a set of customers with known demands on minimum-cost vehicle routes originating and terminating at a depot. Figure 23 illustrates a typical input for a VRP problem and one of its possible outputs:

FIGURE 23: An Instance of a VRP (left) and its solution (right)



Formulation:

The VRP is a combinatorial problem whose ground set is the edges of a graph ${G(V,E)}$. The notation used for this problem is as follows:

- ${V = \left\lbrace v_{0}, v_{1}, …, v_{n} \right\rbrace}$ is a vertex set, where:
  - Consider a depot to be located at ${v_0}$.
  - Let ${V' = V \backslash \left\lbrace v_{0} \right\rbrace}$ be used as the set of ${n}$ cities.
- ${A = \left\lbrace(v_{i},v_{j}) \mid v_{i},v_{j} \in V; i \neq j \right\rbrace}$ is an arc set.
- ${C}$ is a matrix of non-negative costs or distances ${c_{ij}}$ between customers ${v_{i}}$ and ${v_{j}}$.
- ${d}$ is a vector of the customer demands.
- ${R_{i}}$ is the route for vehicle ${i}$.
- ${m}$ is the number of vehicles (all identical). One route is assigned to each vehicle.

When ${c_{ij} = c_{ji}}$ for all ${(v_{i}, v_{j}) \in A}$ the problem is said to be symmetric and it is then common to replace ${A}$ with the edge set ${E = \left\lbrace(v_{i},v_{j}) | v_{i},v_{j} \in V; i \lt j \right\rbrace}$. With each vertex ${v_{i}}$ in ${V'}$ is associated a quantity ${q_{i}}$ of some goods to be delivered by a vehicle. The VRP thus consists of determining a set of ${m}$ vehicle routes of minimal total cost, starting and ending at a depot, such that every vertex in ${V'}$ is visited exactly once by one vehicle. For easy computation, it can be defined ${b(V) = \left\lceil \sum_{v_{i} \in V} d_{i}) / C \right\rceil}$, an obvious lower bound on the number of trucks needed to service the customers in set ${V}$. We will consider a service time ${\delta_{i}}$ (time needed to unload all goods), required by a vehicle to unload the quantity ${q_{i}}$ at ${v_{i}}$. It is required that the total duration of any vehicle route (travel plus service times) may not surpass a given bound ${D}$, so, in this context the cost ${c_{ij}}$ is taken to be the travel time between the cities.

The VRP defined above is NP-hard. A feasible solution is composed of:

- a partition ${R_{1}, … , R_{m}}$ of ${V}$;
- a permutation ${\sigma_{i}}$ of ${R_{i} \bigcup {0}}$ specifying the order of the customers on route ${i}$.

The **cost of a given route** (${R_{i} = \left\lbrace v_{0}, v_{1}, …, v_{m+1} \right\rbrace}$), where ${v_{i} \in V}$ and ${v_{0} = v_{m+1} = 0}$ (0 denotes the depot), is given by ${C(R_{i}) = \sum_{i=0}^{m} c_{i,i+1} + \sum_{i=1}^{m} \delta_{i}}$. A route ${R_{i}}$ is feasible if the vehicle stop exactly once in each customer and the total duration of the route does not exceed a prespecified bound ${D}$: ${C(R_{i}) \leq D}$. Finally, the cost of the problem solution ${S}$ is: ${F_{VRP} = \sum_{i=1}^{m} F(R_{i})}$.

Solution Methods for VRP:

Here, the most commonly used techniques for solving Vehicle Routing Problems are listed. Near all of them are heuristics and metaheuristics because no exact algorithm can be guaranteed to find optimal routes within reasonable computing time when the number of ports is large. This is due to the NP-Hardness of the problem.

Next we can find a classification of the solution techniques we have considered:
Metaheuristics → Ant Algorithms, Constraint Programming, Deterministic Annealing, Genetic Algorithms, Simulated Annealing, Tabu Search → Granular Tabu, The adaptative memory procedure, Kelly and Xu

APPENDIX-II

Particle Swarm Optimization (PSO) Algorithm - MATLAB Implementation Paradigm

Particle swarm optimization (PSO) is a technique for finding approximate solutions to difficult or impossible numeric optimization problems. Appendix A explains a number of PSO implementations and presents their complete programs in Matlab-Simulink.

PSO Outline:

- The particle swarm algorithm begins by creating the initial particles, and assigning them initial velocities.
- It evaluates the objective function at each particle location, and determines the best (lowest) function value and the best location.
- It chooses new velocities, based on the current velocity, the particles' individual best locations, and the best locations of their neighbors.
- It then iteratively updates the particle locations (the new location is the old one plus the velocity, modified to keep particles within bounds), velocities, and neighbors.
- Iterations proceed until the algorithm reaches a stopping criterion.

```
%% PSO Algorithm: Simulates the movements of a swarm to minimize the objective
function %%

%---------------------------------
% initialization algorithm parameters
%---------------------------------

iterations = 30;
inertia = 1.0;
correction_factor = 2.0;
swarm_size = 49;

% ---- initial swarm position -----
index = 1;
for i = 1 : 7
  for j = 1 : 7
    swarm(index, 1, 1) = i;
    swarm(index, 1, 2) = j;
    index = index + 1;
  end
end

swarm(:, 4, 1) = 1000;       % best value so far
swarm(:, 2, :) = 0;          % initial velocity

%% Iterations
```

```
for iter = 1 : iterations

   %-- evaluating position & quality ---
   for i = 1 : swarm_size
      swarm(i, 1, 1) = swarm(i, 1, 1) + swarm(i, 2, 1)/1.3;     %update x position
      swarm(i, 1, 2) = swarm(i, 1, 2) + swarm(i, 2, 2)/1.3;     %update y position
      x = swarm(i, 1, 1);
      y = swarm(i, 1, 2);

      val = (x - 15)^2 + (y - 20)^2;         % fitness evaluation (you may replace this objective
function with any function having a global minima)

      if val < swarm(i, 4, 1)               % if new position is better
         swarm(i, 3, 1) = swarm(i, 1, 1);    % update best x,
         swarm(i, 3, 2) = swarm(i, 1, 2);    % best y postions
         swarm(i, 4, 1) = val;               % and best value
      end
   end

   [temp, gbest] = min(swarm(:, 4, 1));       % global best position

   %--- updating velocity vectors
   for i = 1 : swarm_size
      swarm(i, 2, 1) = rand*inertia*swarm(i, 2, 1) + correction_factor*rand*(swarm(i, 3, 1) -
swarm(i, 1, 1)) + correction_factor*rand*(swarm(gbest, 3, 1) - swarm(i, 1, 1));   %x velocity
component
      swarm(i, 2, 2) = rand*inertia*swarm(i, 2, 2) + correction_factor*rand*(swarm(i, 3, 2) -
swarm(i, 1, 2)) + correction_factor*rand*(swarm(gbest, 3, 2) - swarm(i, 1, 2));   %y velocity
component
   end

   %% Plotting the swarm
   clf
   plot(swarm(:, 1, 1), swarm(:, 1, 2), 'x')   % drawing swarm movements
   axis([-2 30 -2 30]);
pause(.2)
end
```

APPENDIX-III

Ant Colony Optimization (ACO) Algorithm - MATLAB Implementation Paradigm

```
function[bestrute,setuptimes,machines] =
ACO(graph,machine,nodes,Q,a,b,p,t,mactime,jobtime)
%% (Ant System)
%---------------------------------
% initialization algorithm parameters
%---------------------------------

m = size(graph,2);
n = size(graph,1);
minimumspan=0;

% pheromone graph
pher = pheromone(graph,t);

simpulcount=0;
for i=1:n
   for j=1:m
      if(graph(i,j)~=0)
         v(i,j)=1/graph(i,j);
         simpulcount=simpulcount+1;
      end
   end
end

% preparation of the visit of ants
state=1;
while(state==1)
rute = zeros(m,simpulcount+2); %matrix route
for i= 1: n
  node=nodes(i,1); %the first thread
  rute(i,2)=nodes(i,1); %enter the first node of the matrix
  stat=1;
  temps =pher;
  temps(:,(nodes(i,1)+1))=0;
  while(stat<simpulcount)
     temp = temps(node+1,:);
     for j=2:size(temp,2)
        column=mod((j-1),size(v,2));
        if(column==0)
           column=3;
        end
        if(column>1) %filter operation
           allowed=0;
           for k=1:size(rute,2)
            if(nodes(ceil((j-1)/size(v,2)),column-1)==rute(i,k))
                 allowed=temp(1,j);
              end
```

```
            end
            temp(1,j)=allowed;
        end
        temp(1,j)= temp(1,j)*v(ceil((j-1)/size(v,2)),column);
    end
    sigma = sum(temp);
    prob = temp ./ sigma;
    nodemax = max(prob);
    for j=2:size(prob,2)
      if(prob(1,j)==nodemax)
    node=j-1; % Selected node with the greatest probability value
          stat=stat+1;
          rute(i,stat+1)=node; %store selected node
          temps(:,j)=0;
        end
    end
  end


% find routes with minimum span after the exchange node.
 bestrutes(i,:)=nodeexchange(graph,rute(i,:),nodes,machine,mactime,jobtime);
end
  if(min(bestrutes(:,(size(bestrutes,2))))== minimumspan)
     state=0;
  end
  minimumspan=min(bestrutes(:,(size(bestrutes,2))));
  for j=1:size(bestrutes,1)
    if(bestrutes(j,(size(bestrutes,2)))==minimumspan)
      bestrute=bestrutes(j,:);
    end
  end
  deltapher=Q/minimumspan;
  %update pheromone
  for j=1:size(pher,1)
    for k=1:size(pher,2)
      if(pher(j,k)~=0)
        pher(j,k)= (1-p)*t+deltapher;
      end
    end
  end
end

% calculate the setup time of each operation
setuptimes =setuptime(graph,bestrute,machine,mactime,jobtime);

% set the corresponding machine in a row
machines =setmachine(machine,bestrute);
```

APPENDIX-IV

Publications, Actions & Research Projects

Publications	▪ 2014: International Scientific Conference, eRA-9, Aigaleo Greece: “Solving a ship dispatching problem with PSO algorithm” Ch.Drosos, I.Trichas, A.Vlachos, G.Nikolaou, D.Tseles ▪ 2014: 3rd World Conference on Technology and Engineering Education, Aigaleo Greece: “An Integrated Course in Data Acquisition & Control” Ch.Drosos, A. Dagli - Kapoutsi, D. Piromalis, D. Tseles ▪ 2014: Fifth International Conference on Information, Intelligence, Systems and Applications IISA 2014, Chania Crete: “A comparative study of metaheuristic algorithms for Dartboard Design” Ioannis Trichas, Christos Drosos, Aristidis Vlachos. ▪ 2013: International Scientific Conference, eRA-7, Aigaleo Greece: “Ant Colony Optimization (ACO):The effect of Varying of Levels of Initial Pheromone Quantities on the Objective Function for a Vehicle Scheduling Problem (VSP)”, Ch.Drosos, Th. Panayiotopoulos, G.Trichas, Gr. Nikolaou ▪ 2012: International Scientific Conference, eRA-6, Aigaleo Greece: “Arduino electronics prototyping platform: the use as an Open source wireless data logger for Museum Infrastructures” Drosos Christos , Maniatis Nikolaos, Kanellopoulos Panagiotis, Prof Tseles Dimitris. ▪ 2011: International Scientific Conference, eRA-6, Aigaleo Greece: “Implementation of a dynamic site for agricultural unions” Automation Dpt., Technical Education Institute of Pireaus, Christos Drosos, Emanouil Sofianopoulos, Nikos Alafodimos, Dimitris Piromalis, Dr. Tseles Dimitris. ▪ 2008: International Scientific Conference, eRA-3, Aigina Greece: “An IEEE 802.15.4 Transceivers Characteristics Evaluation”, Automation Dpt., Technical Education Institute of Pireaus, "D.Piromalis, I. Zisos, A. Charitopoulos, A. Tzerachoglou, D. Tseles, N. Vasilakis, Ch. Drosos."
Research Activities	▪ Analysing and manufacturing leading circuits for linear motor. ▪ Development of Educational Packet in Telecommunicating Electronics. ▪ Design & implementation of wireless networks.
Research Projects	▪ Archimedes research program: “Integrated System of Neurofuzzy Systems, Grey Models and Genetic Algorithms for Metrological Parameters Estimation”. ▪ Archimedes research program "New materials of polymeric uterine, emphasized on applications of electrostatic charge".