Cognitive Ant Colony Optimization: A New Framework in Swarm Intelligence

Indra Chandra Joseph Riadi

Ph.D. Thesis **2014**

Cognitive Ant Colony Optimization: A New Framework in Swarm Intelligence

Indra Chandra Joseph Riadi

School of Computing, Science and Engineering
College of Science and Technology
University of Salford, Manchester, UK

Submitted in Partial Fulfilment of the Requirements of the
Degree of Doctor of Philosophy, January 2014

Tables of Contents

List of Publications

Riadi, I.C.J & Nefti-Meziani, 2011, *Cognitive Ant Colony Optimization: A New Framework In Swarm Intelligence,* Proceeding of the 2nd Computing, Science and Engineering Postgraduate Research Doctoral School Conference, 2011

Riadi, I.C.J & Nefti-Meziani, Ab Wahab, M.N, *Ant Colony Optimization for Discrete Domain by using Prospect Theory,* is being prepared to be submitted.

Riadi, I.C.J & Nefti-Meziani, Ab Wahab, M.N, *Ant Colony Optimization for Continuous Domain by using Prospect Theory,* is being prepared to be submitted.

List of Figures

List of Tables

Acknowledgements

I would like to thank all those who have assisted me along the long route to completing this work.

Foremost, I would like to express my sincere gratitude to my supervisor Prof. Samia Nefti-Meziani for the continuous support of my PhD study and research, from initial advice and contacts in the early stages of conceptual inception and through on-going advice and encouragement to this day. Her guidance helped me in all the time of research and writing of this thesis.

I thank to all my other colleagues who have worked with me in the various aspects of this work; in particular, Ahmed Dulaimy, May Namman Bunny, Bassem Al-Alachkar, and Rosidah, to name just a few.

I would also like to thank my family for the support they provided me through my entire life and in particular, I must acknowledge my wife Ririn, and to our children; Anggraini and Setiadi for being so patient and understanding.

In conclusion, I recognize that this research would not have been possible without the scholarship of the Directorate General of Higher Education of Republic Indonesia, and the granted permission for further study of Politeknik Negeri Bandung. I express my gratitude to those agencies.

Declaration

The work contained within this thesis is my own and has not been done in collaboration, except where otherwise stated. No part of this thesis has been submitted to any other university in application for a higher degree.

Abstract

Ant Colony Optimization (ACO) algorithms which belong to metaheuristic algorithms and swarm intelligence algorithms have been the focus of much attention in the quest to solve optimization problems. These algorithms are inspired by colonies of ants foraging for food from their nest and have been considered state-of-art methods for solving both discrete and continuous optimization problems. One of the most important phases of ACO algorithms is the construction phase during which an ant builds a partial solution and develops a state transition strategy. There have been a number of studies on the state transition strategy. However, most of the research studies look at how to improve pheromone updates rather than at how the ant itself makes a decision to move from a current position to the next position.

The aim of this research is to develop a novel state transition strategy for Ant Colony Optimization algorithms that can improve the overall performance of the algorithms. The research has shown that the state transition strategy in ACO can be improved by introducing non-rational decision-making.

The new proposed algorithm is called Cognitive Ant Colony Optimization and uses a new concept of decision-making taken from cognitive behaviour theory. In this proposed algorithm, the ACO has been endowed with non-rational behaviour in order to improve the overall optimization behaviour of ants during the process. This new behaviour will use a non-rational model named prospect theory (Kahneman & Tversky, 1979) to select the transition movements of the ants in the colony in order to improve

the overall search capability and the convergence of the algorithm. The new Cognitive Ant Colony Optimization framework has been tested on the Travelling Salesman Problem (TSP), Water Distribution System and Continuous optimization problems. The results obtained show that our algorithm improved the performance of previous ACO techniques considerably.

Chapter 1 Introduction

1.1 Overview

Ant behaviour fascinates in many ways. How can such a small creature, with the brain the size of a pinhead, be intelligent? Human beings learn to navigate in a city by exploring new knowledge or exploiting existing knowledge. Ants can also navigate in nature to find food and bring it back to their nests. However, ants cannot perform such tasks as individuals; they must be part of ant groups. Ants and other animals, such as certain birds and fish, navigate in their environment by relying on information provided by the others. This collective behaviour leads to what is called “swarm intelligence”, which was first introduced in the context of cellular robotic systems by (Beni & Wang, 1993)

The behaviour of ants as a collective or a colony inspired Dorigo in 1992 to propose an algorithm simulating their behaviour in his PhD thesis. This algorithm is called the ant colony optimization algorithm (ACO), and its aim is to find an optimal solution of NP-hard (non-deterministic polynomial-time hard) problems. ACO algorithms were originally proposed for combinatorial or discrete problems; but recently, they have also been applied to solve continuous or mixed problems. ACO algorithms were first implemented in the travelling salesman problem (TSP) (Dorigo & Gambardella, 1997). Now, they have been applied to various problems, such as job assignment problems, scheduling problems, graph colouring, maximum clique problems, and vehicle routing problems. The latest applications include (for example) cell placement problems in

circuit design, communication network design or bio-informatics problems. Some researchers have also focused on applying ACO algorithms to multi-objective problems and to dynamic or stochastic problems (Blum, 2005).

ACO algorithms have also been applied to path planning for mobile robots, which is an important task in robot navigation (Brand, Masuda, Wehner, & Yu, 2010). They enhance robotic navigation systems in both static and dynamic environments. The need of autonomous mobile robots that move faster in a dynamic, complex — even unknown — environment is of paramount importance in robotics. Without path planning, there would be a need for human operators to specify the motion for mobile robots. Autonomous path planning is essential to increase the efficiency of robot operation.

The latest overview of past and on-going research of ACO in various engineering applications can be found in Geetha & Srikanth (2012). They presented the comprehensive study of the application of ACO in diverse fields such as mobile and wireless networks, sensor networks, grid computing, P2P Computing, Pervasive computing, Data mining, Software engineering, Database systems, Multicore Processing, Artificial intelligence, Image processing, Biomedical application and also other domains relevant to Electrical Engineering fields; hence, the use of ACO algorithms is one of the most encouraging optimization techniques.

1.2 Motivation

Optimization techniques are essential in our daily lives, in engineering and in industry. These techniques are applied in almost all fields of applications. We can optimize to minimize cost and energy consumption, or to maximize profit, output, performance and efficiency. Optimization is imperative in the real world because of limitations in

resources, time and money. In reality, optimization problems are more complex; the parameters as well as the constraints influence the problems' characteristics. Furthermore, optimal solutions are not always possible. We can only attain suboptimal — or even just feasible — solutions that are satisfying, robust and practically achievable in a reasonable time scale.

Uncertainty adds more complexity when searching for optimal solutions; for example, if the available materials have a certain degree of inhomogeneity that significantly affects the chosen design. Consequently, we seek not only optimal but also robust design in engineering and in industry. Moreover, most problems are nonlinear and often NP-hard, and need exponential time to find the optimum solution in terms of the problem size. The challenge is to find an effective method for search for the optimal solution, and this is not always achievable.

Many attempts have been made to improve the optimization techniques for dealing with NP-hard problems over the past few decades of computer science research. We can distinguish several high-level approaches, which are either problem-specific or metaheuristic. In the first approach, the algorithms are developed based on the problem's characteristics derived from theoretical analysis or engineering. The second approach, the algorithms are used to find an answer to a problem when there are few or no assumptions; no clue as to what the optimal solution looks like; very little heuristic information to be followed; and where brute force search is out of the question because the space of the candidate solutions is comparatively large.

Metaheuristics implement a form of stochastic optimization method, which randomly searches for the optimal solution. Such randomness makes the methods less sensitive to modelling errors, and may enable the search to escape a local minimum and eventually

to approach a global optimum. In fact, this randomization principle is a simple and effective way to obtain algorithms with good performance for all sorts of problems. Some examples of stochastic optimization are simulated annealing, swarm intelligence algorithms and evolutionary algorithms.

Swarm intelligence algorithms derive from the social behaviour of animals that consist of a group of non-intelligent simple agents with no central control structure and no self-organization to systematise their behaviour (Engelbrecht, 2005). The local interactions between the agents and their environment lead to the emergence of global collective behaviour that is new to the individual agents. Examples of swarm intelligence algorithms are particle swarm optimization (PSO), ant colony optimization (ACO), and bee colony algorithms (BCA).

ACO algorithms are useful in problems that need to find paths to goals. Natural ants lay down pheromones directing each other to resources while exploring their environment. Artificial ants locate optimal solutions by moving through a searching space representing all possible solutions using a state transition strategy. They measure the quality of their solutions, so that in later simulation iterations more ants can locate better solutions.

Currently, research about the state transition strategy of ACO algorithms is scarce. Deneubourg, Aron, Goss & Pasteels (1990) presented a simple random model to describe the dynamic of ants in the double bridge experiment, which was the origin of state transition strategy. Dorigo, Maniezzo & Colorni (1996) proposed the first real state transition strategy while introducing the ant system (AS) algorithm, which is called the random-proportional rule. Subsequently, Dorigo and Gambardella (1997) proposed the pseudo-random-proportional rule when they were introducing the ant

colony system (ACS) algorithm, which produced better results when compared to the AS algorithm. Until now, almost all the advanced ant colony algorithms have used the pseudo-random-proportional rule.

Pseudo-random-proportional transition strategy has two main behaviours: exploitation and exploration. *Exploitation* behaviour is the ability of the algorithm to search through the solution space — where good solutions have previously been found — by always choosing the candidate with maximum utility. Such behaviour is *rational* behaviour. *Exploratory* behaviour is the ability of the algorithm to broadly search by randomly choosing the candidate's solution, which is not always that with the maximum utility. This kind of behaviour is *non-rational* behaviour. Higher exploitation is reflected in the rapid convergence of the algorithm to a suboptimal solution. Higher exploration results in better solutions at a higher computational cost due to the slow convergence of the method. In order to have a good result it is important to balance between these two behaviours. This is not straightforward; it needs experience and many experiments.

Experiments in behavioural studies often find the prediction of utility maximization, which is adopted by Expected Utility Theory (EUT), to be violated (Camerer, 1989). Kahneman & Tversky (1979) showed that changing the ways in which options are framed could generate noticeable shifts in preference. Their experiments captured a pattern of risk attitudes that differed from utility maximization. The rational assumption of EUT has been demonstrated to be wrong by Allais in the Allais paradox (Allais, 1979). This paradox shows that the rational assumption systematically fails to reproduce the human preferences over a simple discrete choice problem. Fortunately, a mathematical model has been developed by Kahneman and Tversky (1979) to cope with the Allais paradox; it is called the prospect theory (PT). PT is a valid hypothesis

for human decision-making behaviour, which can be rational or non-rational depending on the way in which the problem is framed.

The ACO algorithms exhibit rational behaviour during exploitation, and they exhibit non-rational behaviour during exploration. The challenge is how to balance these two behaviours. In this thesis, we propose PT as a transition strategy in ACO algorithms. By using PT, the balance between exploration and exploitation can be adjusted by the PT model through changing the framing problem. Moreover, PT can handle the issue of how to deal with the uncertainty that has become one of the major issues in ACO optimization. By proposing to include PT in the ACO algorithms, elements of human behaviour are introduced into ACO algorithms for the first time.

1.3 Research Objective

The objective of the research is to develop a novel state transition strategy for the Ant Colony Optimization algorithms that can improve the overall performance of the algorithms. The state transition strategy in ACO algorithms is a decision-making process to select the next position during the solution construction phase, and it is very important to determine the overall performance of the algorithms.

The usage of prospect theory in ACO algorithms as a framework for a state transition strategy is investigated for the first time. This new framework will be tested in both discrete and continuous problems in order to validate the improvement of the performance that has been made. This framework is also tested in some mathematical problems and some real problems.

To achieve these objectives, the following tasks have been carried out:

- To verify that the new proposed framework can be applied successfully in Ant Colony Optimization based on the prospect theory, which represents human-like decision-making under risk for both discrete and continuous optimization problems.
- To test the new framework in the travelling salesman problem (TSP), Water Distribution Systems (WDSs) and various continuous problems.
- To compare the proposed framework with the results reported in literature, based on better solutions, lower cost, a reduction in time consumption (in most cases).

1.4 Research Contribution

The research reported in this thesis contributes to the field of Ant Colony Optimization by formulating a new state transition strategy using decision-making under risk. The proposed framework provides the following unique contributions:

A novel state transition strategy model for Ant Colony Optimization based on human decision behaviour under risk is presented. The proposed model captures common human decision-making attitudes towards risk, i.e., risk aversion and risk seeking, hence improving the exploration/exploitation of Ant Colony Optimization during the iteration process.

The proposed framework exhibits comparatively: 1) fewer function evaluations than other state-of-the-art decision-making models when applied to solve the same problems; 2) greater flexibility, as it can be used for several different systems, where decision-

making under risk is paramount. This framework has been tested through 3 different systems to:

- Find appropriate decision variables for several optimization problems among preferred operating decision ranges.
- Find an optimal network design for a water distribution system (WDS).

1.5 Organization of the Thesis

The remainder of this thesis is organized as follows:

Chapter 1 provides an introduction and highlights issues related to the thesis, its motivations, aims, objectives, contributions to knowledge, and the thesis structure. Chapter 2 presents a review of previous work in ACO. It contains a brief introduction, approaches to build ACO algorithms and an overview of different types of ACO algorithms. Chapter 3 provides a general overview of decision-making under risk. It starts with introducing the general decision making and goes on to examine decision making in ACO. This is followed by a review of decision making theory and prospect theory, which is a concept that will be used throughout this thesis. Chapter 4 presents the implementation of prospect theory in the ACO algorithms as a new framework for decision making in ACO. This implementation will be carried out for discrete problems. The TSP will be used as a standard problem and a water distribution system will be used a discrete problem with constraints. Chapter 5 reviews the concept of ACO for solving the continuous optimization problems and presents the implementation of the prospect theory in ACO algorithms as a new framework for decision-making. Chapter 6 presents the analysis of the new methods compare to Genetic Algorithms and Particles Swarm Optimization algorithms. Chapter 7 presents conclusions and some suggestions for future research.

Chapter 2 Survey of Ant Colony Optimization

This chapter presents a review of previous work on ACO. It contains a brief introduction, approaches to building ACO algorithms and an overview of different types of ACO algorithms.

2.1 Introduction

Many researchers have been inspired by the metaphor of the social insect for solving problems. This approach shows how from direct or indirect interactions among a group of simple insects collective intelligence can emerge that offers a flexible and robust solution for relatively complex problems. Insects that live in colonies, such as ants, bees, wasps and termites, have fascinated biologists. An insect colony is highly organized; nevertheless a single insect seems to have its own plan and does not require any supervisor. For example, *Leafcutter* ants *(Atta)* cultivate fungi by cutting leaves from plants and trees (Figure 2.1). Workers search for leaves hundreds of meters away from the nest, truly organizing a freeway to and from their foraging sites (Hölldobler & Wilson, 1990). *Weaver* ant *(Oecophylla)* workers cross a wide gap by forming chains with their own bodies, and work together to carry thick leaf edges to form a nest (Figure 2.2). An even bigger chain can be formed by joining several chains and workers run back and forth using this chain. The chain can also create enough force to pull leaf edges together. The ants will glue both edges when the leaves are in place using a

continuous thread of silk by squeezing larva (Figure 2.3) (Hölldobler & Wilson, 1990) (Camazine, et al., 2003).

There are many more examples that show the remarkable abilities of social insects. An insect is a complex creature that has many sensory inputs, its behaviour is controlled by many stimuli, including interactions with nest-mates, and it makes decisions on the basis of a large amount of information. Nevertheless, the complexity of an individual insect is still not sufficient to explain the complexity of what social insect colonies can achieve or how the cooperation between individuals arises.

Figure 2.1 Leafcutter ants *(Atta)* bringing back cut leaves to the nest

Figure 2.2 Chains of *Oecophylla longinoda*

Figure 2.3 Two workers holding a larva in their mouths

Heredity determines some of the mechanisms of cooperation, for example, anatomical differences between individuals, such as the differences between minors and majors in polymorphic species of ants, can organize the division of labour. However, many aspects of the collective activities of social insects are *self-organized.* Theories of self-organization (SO) in social insects show that complex behaviour may emerge from interactions among individuals that exhibit simple behaviour (Camazine, et al., 2003). This process is spontaneous; it is not directed or controlled by any individual. The resulting organization is wholly decentralized or distributed over all the individuals of the colony. Recent research shows that SO is indeed a major component of a wide range of collective phenomena in social insects (Serugendo, Gleizes, & Karageorgos, 2011).

In self-organization, the structures appear at the global level as a result of a set of dynamical mechanisms of interactions between its lower-level components. These interactions are based only on local information without reference to the global pattern, which is an emergent property of the system, or an external ordering influence. For example, in the case of foraging in ants, the structures emerge as a result of organizing pheromone trails.

There are a considerable number of researchers, mainly biologists, who study the behaviour of ants in detail. Biologists have shown experimentally that it is possible for certain ant species to find the shortest paths in foraging for food from a nest by exploiting communication based only on pheromones, an odorous chemical substance that ants can deposit and smell. This behavioural pattern has inspired computer scientists to develop algorithms for the solution of optimization problems. The first attempts in this direction appeared in the early 1990s, indicating the general validity of the approach. Ant Colony Optimization (ACO) algorithms are the most successful and widely recognized algorithmic techniques based on ant behaviours. These algorithms have been applied to numerous problems; moreover, for many problems ACO algorithms are among the current high performing algorithms.

The first ant colony optimization (ACO), called the ant system, was by Marco Dorigo and was inspired by studying the behaviour of ants in 1991 (Dorigo & Stützle, 2004). An ant colony is highly organized, in which one ant interacts with others through pheromones in perfect harmony. Optimization problems can be solved through simulating ants' behaviours. Since the first ant system algorithm was proposed, there has been a lot of development in ACO. Since its invention, ACO has been successfully applied to a broad range of NP hard problems such as the travelling salesman problem (TSP) or the quadratic assignment problem (QAP), and is increasingly gaining interest in solving real life engineering and scientific problems. A modest survey of ACO algorithms will be presented in this chapter.

2.2 Foraging Behaviour of Real Ants

Although ant species are almost blind, they can still communicate with the environment and with each other by means of substances they release. Some ant species in particular,

such as *Lasius Niger*, use a special kind of substance called pheromones to reinforce the optimum paths between food sources and their nest. To be more specific, these ants lay pheromones on the paths they take and these pheromone trails act as stimuli because the ants are attracted to follow the paths that have relatively more pheromones. Consequently, an ant that has decided to follow a path due to the pheromone trail on that path reinforces it further by laying its own pheromone too. This process can be thought of as a self-reinforcement process since the more ants follow a specific path the more likely that it becomes the path to be followed by the ants in the colony.

Deneubourg, et al. (1990) demonstrated the foraging of a colony of ants through the double-bridge experiments. In these experiments, the nest and the food sources are connected via two different paths and they examine the behaviour by varying the ratio between the lengths of the two paths as shown in Figure 2.4 (a) and Figure 2.4 (b).

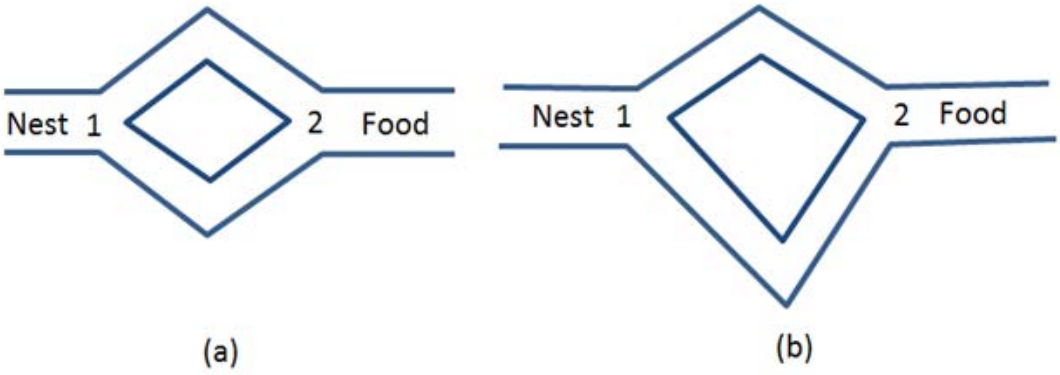

Figure 2.4 Double Bridge Experiments. a) The two paths are equal length. b) The lower path is twice as long as the upper path.

In the first experiment the two paths were set to be an equal length as can be seen above (Deneubourg, et al., 1990). The result showed that initially the ants chose the two paths randomly since there was no pheromone on either of the paths yet. After a while, one of the two paths was followed by a few more ants due to random fluctuations and as a

result more pheromone accumulated on that path. Eventually, the whole colony converged to follow that same path. In the second experiment, the length of one path was two times as long as the other one. Initially, the ants again choose either of the two paths randomly. The ants that have chosen the shortest path arrived at the food source faster and began their return to nest earlier. Consequently, pheromone accumulated faster on the shortest path, and most of the ants converged to this path.

Deneubourg, et al. (1990) investigated further to see what would happen if a shorter path was added after the ants get convergence to one path. They found that the shorter alternative that was offered after convergence was never discovered by the colony. The majority of the ants kept following the longer branch reinforcing it more. This stagnation is caused by the fact that pheromone does not evaporate, and the real ants always follow the suboptimal optimal path even there is a shorter one. This behaviour would have to be overcome.

2.3 The Design of Artificial Ants

Real ant colonies make probabilistic movement based on the intensity of pheromone to find the shortest paths between their nest and the food source. ACO algorithms use similar agents called artificial ants. Artificial ants have the properties of the real ants. The differing characteristics of the artificial ants from the real ants were explained in (Blum, 2005):

- In foraging for food, the real ants will directly evaluate the intensity of pheromone during their way from the nest to the food. While artificial ants will evaluate a solution with respect to some quality measure, which is used to determine the intensity of pheromone during their return trip to the nest.

- The real ants might not take the same path on their way to the food sources and return trip to their nest. Meanwhile, each of the artificial ants moves from the nest the food sources and follows the same path to return.
- The real ants lay pheromone whenever they move from and back to the nest. However, the artificial ants only deposit artificial pheromone on their way back to the nest.

In order to solve problems in engineering and computer science, which is the intention of ACO, some capabilities need to be added to the artificial ants which are not found in the real ants (Dorigo, Maniezzo & Colorni, 1996):

- Memory is used by artificial ants to save the path that they have taken while constructing their solutions. The amount of pheromone is determined based on the quality of their solution, and they retrace the path in their memory to deposit it. This means that the intensity of pheromone depends on the quality of the solutions. The better the solutions the more pheromones are received.
- The transition policy of artificial ants does not only depend on the pheromone trail information but also specific heuristic information. Real ants make their movement with respect to the pheromone deposits on the environment.
- Pheromone evaporation is added to encourage exploration in order to prevent the colony from trapping in a suboptimal solution whereas in real ant colonies, pheromone evaporation is too slow to be a significant part of their search mechanism.

2.4 The Ant Colony Optimization Metaheuristic

Some of the combinatorial optimization problems are difficult to solve optimally in polynomial computational time. Metaheuristic is an alternative to solve this kind of

problems by using approximate methods that try to improve a candidate solution with problem-specific heuristic. Metaheuristics give a reasonably good solution in a short time, although they do not guarantee an optimal solution is ever found. Many metaheuristic implement some stochastic optimization. For example, greedy heuristic can be used to build the solution, which is constructed by taking the best action that improves the partial solution under construction. However, these heuristic methods produce a very limited variety of solutions, and they can easily get trapped in local optima. There are metaheuristic methods proposed to solve these problems; e.g. simulate annealing that guides local search heuristic to escape local optima (Dorigo & Stützle, 2004). A metaheuristic is a general framework that guides a problem specific heuristic.

In the Ant Colony Optimization, ants use heuristic information, which is available in many problems, and pheromone that they deposit along paths which guides them towards the most promising solutions. The most important feature of the ACO metaheuristic is that the ants search experience can be used by the colony as the collective experience in the form of pheromone trails on the paths, and a better solution will emerge as a result of cooperation.

The basic principle of the ACO algorithm is graphically shown in Figure 2.5. The principle works as follows: first, a finite set $\mathcal{C}$ of solution components has to be derived to construct solutions to the combinatorial optimization (CO) problem. Secondly, the pheromone model, which is a set of pheromone values $\mathcal{T}$, has to be defined. This set of values is used to parameterize the probabilistic model. The pheromone values $\tau_i \in \mathcal{T}$ are usually associated with solution components. The ACO approach solves iteratively the optimization problems in two steps:

- candidate solutions are constructed using probability distribution from a pheromone model;
- the candidate solutions are used to bias the sampling by modifying the pheromone value in order to obtain high-quality solutions.

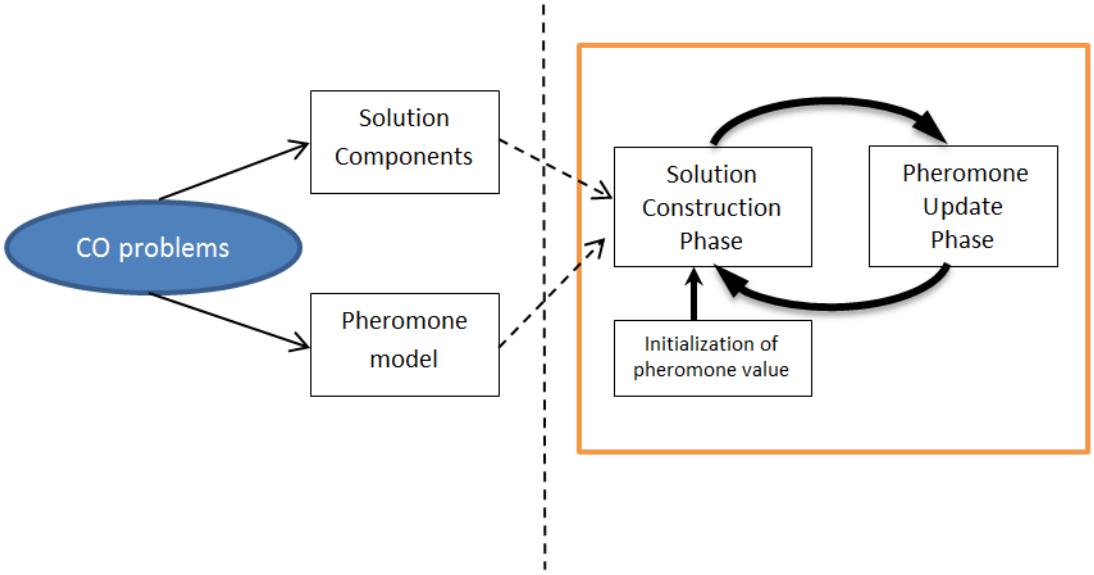

Figure 2.5 Basic principle of AC (Source: Blum, 2005)

Basically, the ACO metaheuristic is composed of 3 main phases (Dorigo and Stützle, 2004): Solution Construction Phase, Update Pheromone Phase and Daemon Actions. Each phase can be briefly described as follows:

Solution construction phase implements a stochastic transition policy, which is a probabilistic constructive, as a function of the pheromone trail, problem-specific heuristic, and the constraints are defined by the problem. This policy controls the movement of the ants to one adjacent state allowed in their vicinity. Once the ants have completed their solutions, they evaluate the quality of their solution, which will be used in the Pheromone Update Phase.

Pheromone update phase guides the search to the region that contains high-quality solutions. In this phase, the pheromone trails are adjusted based on the latest experience of the colony. The update phase consists of decreasing and increasing the pheromone intensity on the trails. Decreasing pheromone can be achieved through pheromone evaporation. Evaporating pheromone encourages exploration and prevents stagnation. Increasing pheromone is implemented by depositing new pheromone trails on the paths used in the solutions of the ants in the previous stage. The amount of pheromone that is deposited depends on the quality of the particular solutions that each path belongs to. The paths that are used in many solutions and/or in better solutions receive more pheromone. The intensity of pheromone will be biased towards the best solution found so far. Each of ACO variants has a different pheromone update method.

Daemon actions phase is an optional phase where there is extra enhancement to the original solutions or a centralized action is implemented, which cannot be done by a single ant. For example, the use of local search methods or to lay extra pheromone to the best solution found so far.

The pseudo code of the ACO algorithm is shown in Figure 2.6.

Algorithm *Ant Colony Optimization*

```
Set parameters, initialize pheromone trials
while (termination condition not met) do
   Solution Construction Phase
   Pheromone Update Phase
   Daemon Actions Phase //optional
end while
return best solution
```

Figure 2.6 ACO metaheuristic in a high-level pseudo-code

2.5 ACO Problems

ACO algorithms are very good candidates for solving combinatorial problems since the artificial ants build the solution constructively by adding one component at a time. The ACO is also suitable for the problems where the environment may change dynamically, as ACO algorithms can be run continuously and adapted to changes in real time.

The following are characteristics that should be defined for a problem to be an ACO problem as presented in Dorigo and Stützle (2004) and Blum (2005):

- There exists a finite set of components such that $C = \{c_1, c_2, \dots, c_n\}$.
- There is a set of constraints Ω defined for the problem to be solved.
- The states of the problem can be described in the form of a finite-length sequence of components such that $q = \langle q_i, q_j, \dots, q_u, \dots \rangle$. Let Q denote the set of all sequences q and $\tilde{Q}$ denote the set of feasible sequence in Q satisfying the constraint Ω.
- There exists a neighbourhood structure defined between the states. q_2 is a neighbour of q_1 if q_2 can be reached from q_1 in one valid transition between the last component of q_1 and the first of component of q_2 and both q_1 and q_2 belong to Q.
- There exists a set of candidate solutions S such that $S \subseteq Q$ (also $S \subseteq \tilde{Q}$ if we don't allow the unfeasible solutions to be constructed at all).
- There is a non-empty set of optimal solutions S^* such that $S^* \subseteq \tilde{Q}$and $S^* \subseteq S$.
- There is an objective function $f(s)$ to evaluate the cost of each solutions in S.
- There may also be an objective function associated with states to be able to calculate the partial solution under construction.

Given this representation, an ACO problem can be seen as a graph $G = (C, E)$where C is a set of nodes and E is a set of edges connecting these nodes. The paths in G

correspond to the states in S, a set of candidate solutions. The edges in E correspond to valid transitions between the states in S. The transition costs may be associated with the edges explicitly. Pheromone trails are associated to either the nodes or the edges, yet the latter is more common (García, Triguero, & Stützle, 2002).

In this graph, each ant in the colony searching the minimum length solution path starts from node c in G and stores it in its memory sequentially. The solution is gradually built by making the transition from its current state to one of the states in its neighbouring area, according to its stochastic transition policy, which is a function of the level of pheromone on the trail, problem-specific heuristic and the number of constraints defined by the problem. When the tour is complete, that is, all nodes have been visited once, the artificial ant uses its memory to evaluate its solution via the objective function and to retrace its nodes solution to create a path and deposit pheromone on it.

2.6 Ant Colony Optimization Algorithms

Many attempts have been done in order to improve overall performance or on a specific problem, as a result several variants of ACO have been developed. Dorigo et al. (1996) developed a variant called the elitisit ACO. In this variant only the best ants from each generation are allowed to update the pheromone. The other variant is the rank-based ACO (Bullnheimer, Hartl, & Strauß, 1999). This variant is a slight extension of the elitist ACO, in which only the best number of ants, n, deposit pheromone and the amount deposited is proportional to their rank. Stützle & Hoos (2000) developed the MAX-MIN ACO algorithm in an attempt to more explicitly control pheromone level, in which an upper and lower limit are placed on the amount of pheromone permitted on any edge.

Some more recent ACO adaptation ideas have been proposed as a hybrid version combining ACO with other methods. Hybridization is nowadays recognized to be an essential aspect of high performing algorithms (Blum, 2005). Hybridization algorithms are more exquisite than the original one. In fact, many of the current state-of-art ACO algorithms include components and ideas originating from other optimization techniques. The earliest type of hybridization was the incorporation of local search based methods such as local search, tabu search, or iterated local search, into ACO. However, these hybridizations often reach their limits when other large-scale problem instances with a huge search space or highly constrained problems for which it is difficult to find feasible solutions are concerned. Therefore, some researchers recently started investigating the incorporation of more classical Artificial Intelligence and Operational Research methods in ACO algorithms.

The ACO algorithm was originally introduced for combinatorial optimization. Recently, ACO algorithms have been developed to solve continuous optimization problems. These problems are characterized by the fact that the decisions variables have continuous domains, in contrast to the discrete problems. Early applications of ant-based algorithms for continuous optimization include algorithms such as Continuous ACO (CACO) (Bilchev & Parmee, 1995), the API algorithm (Monmarché, Venturini, & Slimane, 2000), and the Continuous Interacting Ant Colony (CIAC) (Dréo & Siarry, 2004). However, all these approaches are conceptually quite different from ACO for discrete problems. Socha & Dorigo (2008) proposed a new approach to ACO for continous domain. This approach is closest to the spirit of the ACO for discrete problems.

2.6.1 Ant System

Construction Phase

Artificial ants construct solutions from a sequence of solution components taken from a finite set of n available solution components $\mathcal{C} = \{c_1, \dots, c_n\}$. A solution construction starts with an empty partial solution $S^p = \emptyset$. Then, at each construction step, the current partial solution S^p is extended by adding a feasible solution component from the set $\mathcal{N}(S^p) \in \mathcal{C} \backslash S^p$, which is defined by the solution construction mechanism. The process of constructing graph $G_{\mathcal{C}} = (\boldsymbol{V}, \boldsymbol{E})$ the set of solution components $\mathcal{C}$ may be associated either with the set $\boldsymbol{V}$of vertices of the graph $G_{\mathcal{C}}$, or with the set $\boldsymbol{E}$of its edges.

The allowed paths $G_{\mathcal{C}}$ are implicitly defined by a solution construction mechanism that defines the set $\mathcal{N}(S^p)$ with respect to a partial solutionS^p. The choice of solution component from $\mathcal{N}(S^p)$ is done probabilistically at each construction step. The exact rules for probabilistic choice of solution components vary across different variants of ACO.

In the example of Ant System (AS) applied to TSP the solution construction mechanism restricted the set of traversable edges to the ones that connected the ants' current node to unvisited nodes. The choice of solution component from $\mathcal{N}(S^p)$ is at each construction step performed probabilistically with respect to the pheromone model. In most ACO algorithms the respective probabilities – also called transition probabilities – are defined as follows:

$$p(c_i|s) = \frac{[\tau_i]^{\alpha}.[\eta(c_i)]^{\beta}}{\sum_{c_j \in \mathcal{N}(s)}[\tau_j]^{\alpha}.[\eta(c_j)]^{\beta}}, \forall c_i \in \mathcal{N}(S^p), \qquad 2\text{-}1$$

where η is an optional weighting function, that is, sometimes depending on the current sequence, assigns at each construction step a heuristic value $\eta(c_j)$ to each feasible solution component $c_j \in \mathcal{N}(S^p)$. The values that are given by the weighing function are commonly called *heuristic information*. τ_i is the amount of pheromone. Furthermore, the exponents α and β are positive parameters whose values determine the relation between pheromone information and heuristic information. In TSP examples, we chose not to use weighting function η, and we have set α to 1. It is interesting to note that by maximizing Eq. (2-1) deterministically (i.e. $c \leftarrow argmax\{\eta(c_i)|c_i \in \mathcal{N}(S^p)\}$), we obtain a deterministic greedy algorithm.

Pheromone Update

Different ACO variants mainly differ in the update of the pheromone values they apply. In the following, we outline a general pheromone update rule in order to provide the basic idea. This rule consists of two parts. First, a *pheromone evaporation*, which uniformly decreases all the pheromone values, is performed. From a practical point of view, pheromone evaporation is needed to avoid a too rapid convergence of the algorithm towards a sub-optimal region. It implements a useful form of forgetting, favouring the exploration of new areas in the search space. Secondly, one or more solutions from the current and/or from earlier iterations are used to increase the values of pheromone trail parameters on solution components that are part of these solutions:

$$\tau_i \leftarrow (1-\rho).\tau_i + \rho. \sum_{\{s \in S_{upd} | c_i \in S\}} w_s.F(s) \qquad 2\text{-}2$$

For $i = 1, \ldots, n$. Hereby, S_{upd} denotes the set of solutions that are used for update. Furthermore, $\rho \in (0, 1]$ is a parameter called evaporation rate, $F: S \longmapsto \mathbb{R}^+$ is a so-

called quality function such that $f(s) < f(s') \Longrightarrow F(s) \gg F(s'), \forall s \neq s'$. In other words, if the objective function value of solution s is better than the objective function value of a solution s', the quality of solution s will be at least as high as the quality of solutionss'. Eq. (2-2) also allows an additional weighting of the quality function, i.e., $w_s \in \mathbb{R}^+$denotes the weight of a solution s.

Instantiations of this update rule are obtained by different specifications of S_{upd} and by different weight settings. In many cases, S_{upd} is composed of some of the solutions generated in the respective iteration (henceforth, denoted by S_{iter}). Solution S_{bs} is often called the best-so-far solution. A well-known example is the *AS-update* rule, that is, the update rule of AS. The AS-update rule, which is well-known due to the fact that AS was the first ACO algorithm to be proposed in the literature, is obtained from the update rule in Eq. 2-2 by setting:

$$S_{upd} \leftarrow S_{iter} \text{ and } w_s = 1 \; \forall s \in S_{upd} \qquad 2\text{-}3$$

All solutions that were generated in the respective iteration are used to update the pheromone update and the weight of each of these solutions is set to 1. An example of a pheromone update rule that is used more widely in practice is the *IB-update* rule (where IB stands for *iteration-best*). The IB-update rule is given by:

$$S_{upd} \leftarrow \{s_{ib} = argmax\{F(s) | s \in S_{iter}\}\} \; with \; w_{isb} = 1 \qquad 2\text{-}4$$

Only the best solution generated in the respective iteration is used to update the pheromone values to construct the IB-update. This solution, denoted bys_{ib}, is weighted by 1. The IB-update rule introduces a much stronger bias towards the good solutions found than the AS-update rule. However, this increases the danger of premature

convergence. An even stronger bias is introduced by the *BS-update* rule, where BS refers to the use of the best-so-far solution s_{bs}. In this case, S_{upd} is set to $\{s_{bs}\}$ and s_{bs} is weighted by 1, i.e. $w_{S_{bs}} = 1$. In practice, ACO algorithms that use variations of the IB-update or the BS-update rule and that additionally include mechanisms to avoid premature convergence achieve better results than algorithms that use the AS-update rule.

Eq. 2-2 shows that the pheromone update in ACO is done in 2 steps. In the first step, all pheromone trails are decreased by a constant rate ρ, where $0 < \rho \leq 1$, due to pheromone evaporation. This evaporation in AS is implemented as Eq. 2-5 below.

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} \quad \forall (i,j) \in E \tag{2-5}$$

In the second step, the pheromone quantities,$\Delta\tau_{ij}$, to be laid on each edge (i,j) are calculated according to Eq. 2-6. In this equation $\Delta\tau_{ij}^{k}$ corresponds to the quantity of pheromone deposited by ant k and L^k stands for the cost of the solution found by ant k.

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k},$$

$$where\ \Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{L^k}, if\ edge(i,j) is\ in\ the\ solution\ of\ ant\ k \\ 0, \qquad\qquad otherwise \end{cases} \tag{2-6}$$

$$m\ is\ the\ number\ of\ ants$$

Eq. 2-6 shows that the edges used in shorter tours and/or used by more ants receive more pheromone trails.

Daemon Actions

Daemon actions can be used to implement centralized control which cannot be performed by a single ant. Examples are the application of local search methods to the constructed solutions, or the collection of global information that can be used to decide whether it is useful or not to deposit additional pheromone to bias the search process from a non-local perspective. As a practical example, the daemon may decide to deposit extra pheromone on the solution components that belong to the best solution found so far.

2.6.2 Elitist Ant Colony Optimization

A first improvement on initial AS, called the elitist strategy for Ant System (EAS), was introduced in Dorigo, Maniezzo & Colorni (1996). The idea was to provide strong additional reinforcement to the edge belonging to the best tour found since the start of the algorithm; this tour is denoted as T^{bs} (best-so-far tour) in the following. Note, that this additional feedback to the best-so-far tour (which can be viewed as additional pheromone deposited by additional ant called best-so-far ant) is another example of a daemon action of the ACO metaheuristic.

Pheromone Update

The additional reinforcement of tour $\Delta\tau^{bs}$ is achieved by adding a quantity e/C^{bs} to its edge, where e is a parameter that defines the weight given to the best-so-far tour T^{bs}, and C^{bs} is its length. Thus, Eq. 2-2 for the pheromone deposit becomes

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + e\,\Delta\tau_{ij}^{bs} \qquad 2\text{-}7$$

Where $\Delta\tau_{ij}^{k}$ is defined as follows:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & if\ j \in \mathcal{N}(s) \\ 0 & otherwise \end{cases} \qquad 2\text{-}8$$

Where L_k is the length of edge (i, j), Q is a constant parameter, and $\Delta\tau_{ij}^{bs}$ is defined as follows:

$$\Delta\tau_{ij}^{bs} = \begin{cases} \frac{1}{C^{bs}} & if\ j \in T^{bs} \\ 0 & otherwise \end{cases} \qquad 2\text{-}9$$

T^{bs} is the best-so-far tour

Note that in EAS, as well as in other algorithms presented, pheromone evaporation is implemented as in Ant System.

2.6.3 Rank-based Ant System

Bullnheimer, Hartl & Strauß (1999) proposed another improvement of AS called Rank-based AS (RAS). In RAS, the pheromone update is obtained from the update rule (eq. 2-3) by filling S_{upd} with $m-1$ (where $m-1 \leq n_a$, number of artificial) solutions from S_{iter}, and by additionally adding the best-so-far solutions s_{bs} to S_{upd}. The weights of the solutions are set to $w_s = m - r_s \; \forall s \in S_{upd} \backslash \{s_{bs}\}$, where r_s is the rank of solution s. Finally, the weight $w_{S_{bs}}$ of solution S_{bs} is set to m. This means that at each of the iterations the best-so-far solution has the highest influence on the pheromone update, while a selection of the best solutions constructed at the current iteration influences the update depending on their rank.

2.6.4 MAX-MIN Ant System (MMAS)

One of the most successful ACO variants today is the MAX-MIN Ant System (MMAS) (Stützle & Hoos, 2000), which is characterized as follows. It prevents the search from becoming stagnated by imposing the quantity of pheromone trails on each edge to be in the range $[\tau_{min}, \tau_{max}]$. Depending on some convergence measure, for each of the iterations either the IB-update or BS-update rule is used for updating the pheromone values. At the start of the algorithm the IB-update rule is used more often, while during the run of the algorithm the frequency with which the BS-update rule is used is increased. MMAS algorithms use an explicit lower bound $\tau_{min} > 0$ for the pheromone values. In addition to this lower bound, MMAS algorithms used $F(S_{bs})/\rho$ as an upper bound to pheromone values. The value of this bound is updated each time a new best-so-far is found by the algorithm. The best value for τ_{max} is agreed to be $1/\rho L^{bs}$, where $F(S_{bs}) = 1/L^{bs}$, as explained in Stützle & Hoos (2000) so τ_{max} increases as shorter paths are found by the colony. And τ_{min} is set to τ_{max}/a where a is a parameter. Initially, pheromone trails are set to τ_{max} on all edges; therefore ants behave in a more explorative way in the beginning. After a while, the search becomes biased towards the shortest tours found so far; however, if the colony cannot find any improved solution in a predetermined number of iterations, the pheromone trails are reinitialized.

2.6.5 Ant Colony System (ACS)

Dorigo, et al. (1996) introduced ACS, which differs from the original AS algorithm in more aspects than just in the pheromone update. First, instead of choosing at each step during a solution construction the next solution to Eq. 2-1, an artificial ant chooses, with probability q_0, the solution component that maximizes $[\tau_i]^{\alpha}.[\eta(c_i)]^{\beta}$, or it performs, with probability $1 - q_0$, a proportional construction step according to eq.

2-1. This type of solution construction is called *pseudo-random proportional*. So, the transition rule will be

$$s = \begin{cases} \arg max_{j \in N_i^k} \{[\tau_{ij}]^{\alpha} . [\eta_{ij})]^{\beta} & if\ q \leq\ q_0\ (exploitation) \\ S, & otherwise\ (exploration) \end{cases} \qquad 2\text{-}10$$

According to this transition rule, ant k at node i chooses to move to node j using the best edge in terms of the pheromone trail and heuristic value with q_0 probability, therefore exploit the current knowledge of the colony. Otherwise, it selects a random node in its neighbourhood; therefore, it explores a new solution.

Second, ACS uses the BS-update rule with the additional particularity that the pheromone evaporation is only applied to values of pheromone trail parameters that belong to solution components that are in S_{bs}. Third, after each solution construction step, the following additional pheromone update is applied to the pheromone value τ_i whose corresponding solution component c_i was added to the solution under construction:

$$\tau_i \leftarrow (1 - \rho).\tau_i + \rho.\tau_0 \qquad 2\text{-}11$$

Where τ_0 is a small positive constant such that $F_{min} \geq\ \tau_0 \geq c, F_{min} \leftarrow min\{F(s) |\ s\ \in S\}$, and c is the initial value of the pheromone value. In practice, the effect of this local pheromone update is to decrease the pheromone values on the visited solution components, making in this way these components less desirable for the following ants. This mechanism increases the exploration of the search space within each of iterations.

After all the ants in the colony construct their solution, only the tour corresponding to the best solution so far receives a pheromone update. As a result, the search is biased towards the vicinity of the best solutions so far. The equation 2-12 implements the global pheromone update is:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \rho\Delta\tau_{ij}, \quad \forall(i,j) \in T^{bs}, \qquad \text{2-12}$$

$$\text{where } \Delta\tau_{ij} = 1/L^{bs}$$

In the equation 2-12, T^{bs} stands for the best solution so far and L^{bs}stands for the cost of that solution. The parameter ρ again can be thought of as the pheromone evaporation rate as in AS, but it also weighs the pheromone to be deposited in ACS. In other words, the new pheromone trails become a weighted average of the existing pheromone on the trails and the pheromone that is to be deposited (Dorigo & Stützle, 2004).

2.6.6 Hyper-Cube Framework (HFC)

One of the most recent developments is the Hyper-Cube Framework for ACO (Blum & Dorigo, 2004). Rather than being an ACO variant, the HFC is a framework for implementing ACO algorithms which are characterized by a pheromone update that is obtained from update rule by defining the weight of each solution in S_{upd} to be $(\sum_{\{s\in S_{upd}\}} F(s))^{-1}$. Remember that in Eq. 2-3 solutions are weighted. The set S_{upd} can be composed in any possible way. This means that ACO variants such as AS, ACS, or MMAS can be implemented in HCF. The HCF comes with several benefits. On the practical side, the new framework automatically handles the scaling of the objective function values and limits the pheromone value within the interval [0, 1]. On the theoretical side, the new framework makes it possible to prove that in the case of AS

algorithm applied to unconstrained problems, the average quality of the solutions produced continuously increases in asymptotically (Blum & Dorigo, 2004). The name Hyper-Cube Framework stems from the fact that with the weight setting as outlined above, the pheromone update can be interpreted as a shift in a hyper cube.

2.7 The Travelling Salesman Problem (TSP)

The TSP is considered a standard test-bed for evaluation of new algorithmic ideas for discrete optimization; indeed, good performance for TSP is considered reasonable proof of an algorithm's usefulness. The TSP is the problem of a salesman who wants to find the shortest possible trip through a set of cities on his tour of duty, visiting each and every city exactly once.

The problem space can, essentially, be viewed as a weighted graph containing a set of nodes (cities). The objective is to find the minimal-length of a loop of the graph. To solve this problem using ACO algorithms, initially, each ant is put on a randomly chosen city. The ant has a memory which stores the partial solution that it has constructed so far (initially the memory contains only the start city). Starting from its start city, an ant iteratively moves from a city to city. When being in a city i, an ant k chooses to go to a still unvisited city j with a probability given by

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l\in\aleph_i^k}[\tau_{il}(t)]^{\alpha}[\eta_{il}]^{\beta}} & if\ j \in \aleph_i^k \\ 0 & otherwise \end{cases} \qquad 2\text{-}13$$

Where $\eta_{ij} = 1/d_{ij}$, d_{ij} is the distance between i and j, a priori available heuristic information, α and β are two parameters which determine the relative influence of the pheromone trail and heuristic information, and $\aleph_i^k$ is the feasible neighbourhood of

ant k, i.e. the set of cities which ant k has not yet visited. Parameters α and β influence the algorithm behaviour. If $\alpha = 0$, the selection probabilities are proportional to heuristic information and the closest cities will be more likely to be selected. In this case, AS corresponds to a classical stochastic greedy algorithm (with multiple starting points since ants are initially randomly distributed on the cities). If $\beta = 0$, only pheromone amplification is at work. This will lead to the rapid emergence of a stagnant situation with the corresponding generation of the tours which, in general, are strongly suboptimal.

In each step of the algorithm, each ant chooses the next city based on pheromone intensity and distance which indicates a greedy approach. After completing their tours at a step t, pheromone contributions for each ant k and edge (i, j) between city i and j are computed using

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \frac{Q}{L^{k}(t)} & if\ j \in \aleph_i^k \\ 0 & otherwise \end{cases} \qquad 2\text{-}14$$

where Q is a constant and $L^k(t)$ is the tour length of the k-th ant. The pheromone trail is now updated using formula

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \Delta\tau_{ij}(t) \qquad 2\text{-}15$$

Where $\Delta\tau_{ij}(t)$ is computed across all m ants

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) \qquad 2\text{-}16$$

2.8 Ant Colony Optimization Algorithm for Continuous Problems

The class of optimization problems are not only discrete but there are a class of problems that require choosing values for continuous variables. These problems can be tackled by using ant algorithms for continuous function optimization with several possibilities. To apply the original ACO to continuous domain was not straightforward. One possible method is to use simplified direct simulation of the real ants' behaviour or other method is to extend ACO metaheuristic to explore continuous spaces. This extension can be done by the suitable discretization of a search space or by probabilistic sampling.

The use of the ACO metaheuristic to solve continuous optimization problems has been reported by Liao, et al., (2011). Bilchev and Parmee (1995) proposed Continuous ACO (CACO), Monmarché, et al., (2000) suggested the API algorithm, Dréo and Siarry (2004) offered the Continuous Interacting Ant Colony (CIAC), and Socha and Dorigo (2008) recommended the extended ACO application (ACOR). The main differences of the various versions of ACO are the pheromone laying methods, the way of handling the solution construction, pheromone update and optional daemon action, and how to schedule and synchronize these components according to the characteristics of the problem being considered.

ACOR is one of the most popular ACO-based algorithms for continuous domain. Recently, Leguizamón & Coello (2010) proposed a variant of ACOR on six benchmark functions. However, the results obtained are far from competitive with the results obtained by the state-of-the-art continuous optimization algorithms that have recently featured in a special issue of the Soft Computing journal, *SOCO* (Herrera, Lozano, & Molina, 2010), (Lozano, Herrera, & Molina, 2011). Further improvements to ACOR

have been carried out, for example, Liao, et al. (2011) proposed Incremental Ant Colony Optimization (IACO). More detail on ACOR will be presented in Chapter 5.

2.9 Summary

This chapter has reviewed the metaphor of the ant colony to solve optimization problems and the approaches adopted to build ACO algorithms. These approaches include the solution construction phase, the update pheromone phase and the daemon actions. ACO algorithms are different to one another in the way these phases are adapted. The solution construction phase comprises a decision-making process to construct the solutions based on the certain utility. In the next chapter, the decision-making process will be discussed in more detail. This decision-making process is an essential concept that will be used to build the cognitive framework of the ACO.

Chapter 3 Decision-making under Risk

This chapter provides a general overview of decision-making under risk. We start by introducing general decision making and go on to examine decision making in ACO. This is followed by a review of decision making theory and prospect theory, which is a concept that will be used throughout this thesis.

3.1 Introduction

Decision making is an important skill for business and life. Decision making theory looks at how decisions are made and how a decision maker chooses from a set of alternatives (options) that may lead to an optimal decision. There are two goals of decision making theory: firstly, to prescribe how decisions should be made; and secondly, to describe how decisions are actually made. The first goal refers to normative decision making theory, which considers how decisions should be made in order to be rational. The second goal aims to find tools and methodologies to help people make the best decision, and relates to people's beliefs and preferences as they are, not as they should be.

There are several different conditions under which decisions are made: certainty, risk, uncertainty, and ignorance. Decisions are made under certainty when the decision-maker knows what the outcome will be for each alternative, while decision making under risk occurs when each decision leads to a set of possible outcomes, and the probability of each outcome is known. If the decision maker has only partial

information about the probabilities of different outcomes then the decision is made under uncertainty; while if no prior knowledge is used about the probabilities the decision is made under ignorance.

Until the end of the 1970s, the best approximation to describe the decision-maker's behaviour was the Expected Utility Theory (EUT) (Wakker, 2010). EUT has been adopted as the standard decision-making methodology used under risk conditions, but it is incapable of fully capturing actual human decision-making behaviours under uncertainty. Kahneman and Tversky (1979) proposed the prospect theory (PT) as a major breakthrough to capture this behaviour, and it was the first descriptive decision making theory that explicitly incorporated non-rational behaviour in an empirically realistic way. Later, in 1992, they introduced an improved version of PT called Cumulative Prospect Theory (CPT) (Tversky & Kahneman, 1992).

In the decision-making process, people make decisions in risky and riskless situations. The example of decision making under risk is the acceptability of a gamble that yields monetary outcomes with specific probabilities. A typical riskless decision concerns the acceptability of a transaction in which an item or a service is exchanged for money or labour, for example, to buy something. A classic decision making model under risk is a rational model which does not treat gains and losses separately, but there is much evidence that people respond differently to gains and losses.

The first psychophysical approach to decision making was by Daniel Bernoulli in an essay published in 1738 (Kahneman & Tversky, 2000). In this essay, Bernoulli explained why people are generally unfavourable to risk and why risk aversion decreases with increasing wealth. He suggested that people evaluate prospects by the

expectation of the subjective value of monetary outcomes and not by the expectation of the objective value of these outcomes.

Kahneman, Slovic & Tversky (1982) suggest that most judgements and choices are made intuitively and the rules that direct intuition are similar to the rules of perception (visual). Intuitive judgements or choices are spontaneous, without a conscious search or computation, and without effort; however, some monitoring of the quality of mental operations and behaviour still takes place. Intuitive thinking can also be powerful and accurate. Both perception and intuitive evaluations of outcomes are reference-dependent. In standard economic analyses the utility of decision making outcomes is assumed to be determined entirely by their final stakes, and is, therefore, reference-independent (Kahneman, 2003). Preference appears to be determined by attitudes to gains and losses, defined relative to a reference point. This will cause the framing of a prospect to change the choice that the individual decision-maker makes.

This thesis proposes a novel approach for modelling human decision-makers and non-rational behaviour in ACO. This non-rational behaviour is based on PT (Kahneman & Tversky, 1979). This model was used to design a state transition strategy algorithm in ACO. Therefore, instead of fine-tuning the existing model the use of PT is proposed and evaluated.

3.2 Decision Making in ACO

As mentioned in the previous chapter, an ant builds a solution to a problem under consideration in an incremental way with an empty initial solution and iteratively adding aptly defined solution components without backtracking until a complete solution is obtained.

Random-proportional rule (see Eq. 2.1) was the first idea of decision-making applied in AS. Each ant starts from a random node and makes a discrete choice to move the next edge from node i to another node j of a partial solution. At each step, ant k computes a set of feasible expansions, according to a probability distribution. The probability distribution is derived from a utility. This utility is computed from the pheromone intensity and the attractiveness of the edge. This pheromone can be considered as the global information and the attractiveness of the edge which is obtained from some heuristic information represents the local information. Thus, the higher the intensity of the pheromone and the value of the heuristic information are, the higher the utility is, the more profitable it is to include node j in the partial solution. This process is usually accomplished through the use of scaling factors in the form of powers or constants boosting or shrinking an individual utility. The combination of the utility is the cornerstone to the success or failure of this technique and the combined utilities must provide an accurate deflection of the perceived strengths or weaknesses of an edge.

The above decision making may obtain the solution faster, achieving a reasonable quality decision; however, the solution may be far from the optimum and generate only a limited number of different solutions. AS gives a very poor solution quality compared to state-of-the-art algorithms for TSP. To improve the random-proportional rule in AS, Dorigo and Gambardella (1997) introduced the pseudo-random proportional policy. Using this policy, an action that gives maximum reward in the given node is selected with ε probability and the random action selection is chosen with probability $1 - \varepsilon$ (see Eq.2-10). For example, if an experiment is about to run 10 times and the value of ε, is 0.8 , this policy selects the action that gives maximum reward on eight occasions and random actions twice. This policy is called ε-greedy construction.

In an uncertain environment, the ant's probabilistic choices during the construction of the partial solution might lead to an error and create a risk because its choices might be influenced by the ant's attitude to that uncertainty. The common ant-selection models are based on the assumption that the ants are rational, homogenous, and have perfect knowledge. Based on these assumptions, most of the existing state-selection models use utility theory, which is based on the utility maximization assumption. However, these models do not give good results; a bias is needed to allow the ants to choose randomly using a probability distribution.

Reasons to add random choices are to make artificial ants heterogeneous in preferences, calculation errors, unobserved attributes, and insensitivity to small differences. The random component may also reflect the variations in artificial ants' perception of the options. Utility models do not offer much insight into the way in which the ants estimate the choices. Moreover, the uncertainty presented in these models is from the view of the modeller; it provides no hypothesis as to how the ants might themselves consider uncertainty.

Efforts to improve the state transition of the ants have been made by some researchers, such as Zheng, et al., 2010 who proposed a new transition probability for ACO, called Global Random-proportional Rule; Liu, Dai & Tao (2011) argued that their state transition strategy has improved the ACO performance by changing the probability selection function. All these attempts only focus on the modification of the ACO's utility, and never try to shift to paradigm on how an ant like human makes a decision as a human being does.

3.3 Utility Theory

One of the first theories of decision making under risk was based on the expected value. The expected value of an outcome is equal to its payoff times its probability. This model failed to predict the outcomes in many situations because it was apparent that the value was not always directly related to its precise monetary worth.

Daniel Bernoulli was the first to see this flaw and propose a modification to the expected value notion in 1738 in a paper entitled “Exposition of a New Theory on the Measurement of Risk (as cited in (Hansson, 1994)). In fact, Bernoulli was the first to introduce the concept of systematic bias in decision making based on a "psychophysical" mode (McDermott, 2001). By this concept, people do not always make decisions based on the expected value of the outcomes but can be influenced by such factors as the probability of the outcomes and cognitive bias. Bernoulli proposed a utility function to explain people’s behaviour in making choices. The utility function did not use a linear function of wealth, but rather a subjective one, and is concave as shown in Figure 3.1. Bernoulli assumed that people tried to maximize their utility rather than their expected value and that people are typically risk averse.

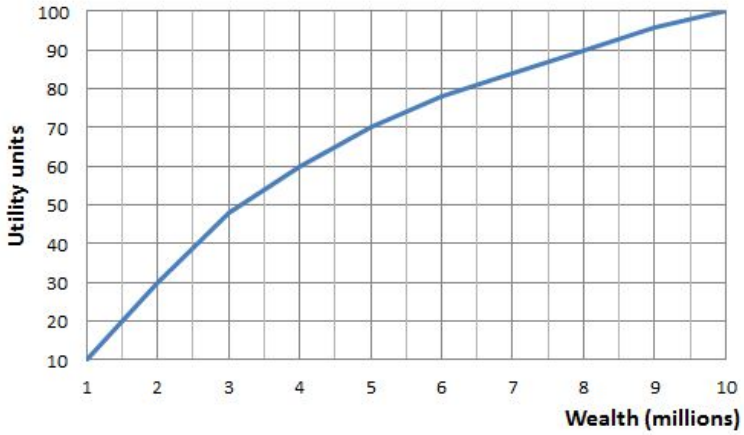

Figure 3.1 Typical utility function

Utility Theory (UT) is an attempt to model human choice (Fishburn, 1970). The modeller knows the preferences of the decision-makers regarding the choice set, i.e. by ranking the options. The modeller must create a utility function that assigns a numerical value to each option to reflect the ranking established by decision-makers for the given choice set.

When adopting a utility function to represent the decision making process, the decisions are always deterministic, i.e. the option with the highest utility is the option chosen. According to the UT, if the utility function is correctly selected it expresses a sharp logic: “the winner takes all”.

The utility function is personal, i.e. each person has their own evaluation process – for this reason, it is called Subjective Utility Theory (SUT). Moreover, according to the UT if the attributes are kept constant, the decision does not change. The only possibility for change is if the attributes or parameters of the utility function change, but the function itself is fixed and constant. Nevertheless, the changes occur exclusively in the parameters and never in the function. Some aspects of the decision making process may remain non-captured by the model and for them a stochastic behaviour is assumed.

The basic concept behind rationality is utility maximization. According to rational behaviour and utilitarianism, when confronted with a decision making problem, human beings evaluate the utility of each option and choose the one with the highest utility. This decision is always deterministic and as long as the environment does not change (the attributes remain the same) the choice is then the same.

3.4 Expected Utility Theory

In 1944, John von Neumann and Oscar Morgenstern reinterpreted Bernoulli's expected utility theory by proposing a formal model called expected utility theory (EUT). EUT used preferences to derive utility, in contrast to Bernoulli's model; utility was used to define a preference, because the highest utility is more preferable. Von Neumann and Morgenstern's axioms do not only rank the utility by an individual's preference but also take into account the possible relationships between the individual's preferences. As long as the relationship between an individual's preference satisfies certain axioms such as consistency and coherence, it became possible to construct an individual utility function for that person; then the decision can be made to maximize the subjective utility.

This subjective expected utility model made no clear distinction between normative and descriptive aspects. This model assumed not only the way the decision should be made but also how people make decisions.

People seek to maximize their subjective expected utility; one person may not share the same utility curve as another, but each follows the same normative axioms in striving toward their individually defined maximum subjective expected utility.

The expected utility theory (EUT) deals with the analysis of choices among risky or uncertain alternatives by comparing their expected utility, i.e., the weighted sums computed by adding the utility values of outcomes multiplied by their respective probabilities. Indeed, EUT could, more precisely, be called "probability-weighted utility theory". It suggests that choices are coherently and consistently made through weighing outcomes by their probabilities. The alternative which has the maximum expected utility is selected and the one which provides less utility is discarded. In

addition, EUT predicts that the choice is invariant, that is, the way the alternatives are presented should not influence the decision makers to choose the alternatives.

The expected utility (EU) is defined as follows:

$$EU = \sum_{i=1}^{n} u(x_i)p_i \qquad 3\text{-}1$$

where $u(x_i)$ and p_i represent the utility function of the outcome x_i and probability of that outcome, respectively. The decision makers then choose the option with the highest expected utility (EU).

In EUT, the utility is often assumed to be a function of final states or wealth. The utility function can be a concave, convex, or linear function. The concave function shows that a person is risk averse, meaning that he/she prefers the certain prospect (x) to any risky (uncertain) prospect with expected value. Thus, the preference of £50 for sure over a 50 – 50 chance of receiving £100 or nothing is an expression of risk aversion. Whereas a risk seeking person has a convex utility function meaning that he/she prefers a chance prospect over a sure outcome of greater expected utility. The one who is risk neutral has a linear utility function; he/she has no preference to the options. These behaviours are shown in Figure 3.2.

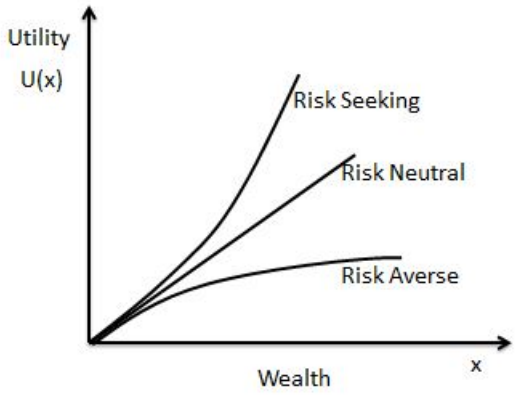

Figure 3.2 Utility functions based on wealth

3.5 Prospect Theory

Expected utility theory has been confronted with several critiques. Some were suggestions that its underlying assumptions were unreasonable, some felt its predictions did not conform to the experimental or empirical evidence, and some of them combined the two lines of criticism. For examples, the Ellsberg paradox and the Allais paradox pointed out the deficiencies of classical expected utility theory. Numerous alternative theories have been offered, one of which is Prospect theory. Prospect theory was formulated first by Kahneman and Tversky (1979) as an alternative method of explaining choices made by individuals under conditions of risk.

Prospect theory is the most accepted description of subjected expected value decision-making by human beings. It describes the subjective human decision making process, specifically in the subjective assessment of probabilities and utilities of the outcomes, and their combination in gambles (lotteries). Prospect theory suggests a nonlinear transformation of the probability scale (Kahneman & Tversky, 1979) (Tversky & Kahneman, 1992). According to the prospect theory, the value V of a simple prospect that pays £x with probability, p and pays nothing with probability (1 – p) is given by:

$$V(x,p) = v(x)w(p) \qquad 3\text{-}2$$

where v measures the subjective value of the consequence x, and w measures the impact of the probability p on the attractiveness of the prospect. The value function,v, is a function of gains and losses relative to some reference point (usually the status quo), with $v(0) = 0$. The value of w is called decision weights; they are normalised so that $w(0) = 0$ and $w(1) = 1$.

Prospect Theory divides the decision making process into two stages. The first stage is the ‘screening’ or editing stage, where probabilities and utilities are subjectively assessed, and the second stage is the ‘evaluation’ stage, where the subjective probabilities and utilities are combined. This two-staged process has three consequences. First, gains and losses are evaluated to a reference point, often assumed to be equivalent to the status quo. People are risk averse in the domain of gains and risk seeking in losses. Secondly, people tend to overweigh small probability events while underweighing medium and high probability events. Moreover, people overweigh certainty, so that they tend to treat highly probable events as certain and highly improbable events as if they were impossible. Finally, the choice of alternatives is influenced by the way the alternatives are presented (Kahneman & Tversky, 1979).

The editing stage comprises four major sequential operations, namely coding, combination, segregation, and cancelation. Coding involves the setting of a reference point by the decision-maker by which all gains and/or losses are measured. Combination consists of the aggregation of probabilities associated with identical outcomes. Segregation involves separating the risky components of the prospect from the riskless components of the prospects. Cancellation is where components shared by all prospects are discarded. Simplification is where probabilities and outcomes are simplified, such as rounding. Elimination by domination is where dominated prospect alternatives are eliminated (Kahneman & Tversky, 1979).

The function of the editing phase is to organize and reformulate the options so as to simplify subsequence evaluation and choice. Editing consists of the application of several operations that transform the outcomes and the probabilities associated with the offered prospects. The probabilities are transformed non-linearly, where small

probabilities are overestimated and large probabilities are underestimated. Overestimated small probabilities generate optimism (high decision weight for less favourable outcomes), while underestimating large probabilities generates pessimism (low decision weight for favourable outcomes, and shows a risk averse attitude).

In the evaluation phase, the decision-maker evaluates the prospects that are attainable to him or her after the conclusion of the editing phase. The decision-maker then chooses the prospect with the highest value.

The Value Function

In PT, the notion of "utility" is replaced by "value". People rate value functions to gains and losses differently. Tversky & Kahneman (1979) found that people value a certain gain more than a probable gain with equal or greater expected value. Gains and losses are valued from a subjective reference point. It can be inferred from their research that the displeasure associated with losses is greater than the pleasure associated with the same amount of gains. It can be further inferred that people respond differently, depending on whether the choice is framed in terms of gains or in terms of losses. Figure 3.3 shows a graph of the value function.

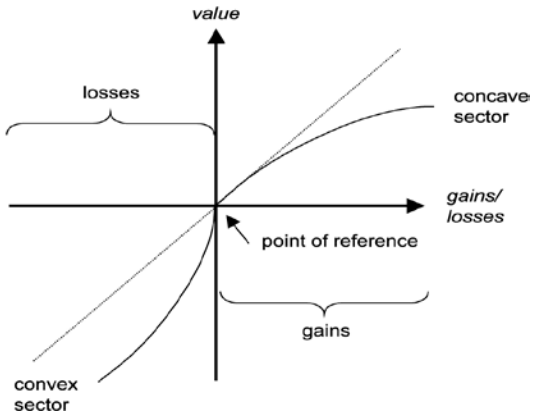

Figure 3.3 Value function

There are three main characteristics of the value function in prospect theory, which are as follows:

1. It is a function of gains or losses. The reference point is used to define the gains and losses. It is a gain if an outcome is greater than the reference point and a loss if the outcome is less than the reference point.
2. It is, generally, concave for gains favouring risk aversion, and, commonly, convex for losses favouring risk seeking.
3. The function is sharply curled at the reference point, steeper for losses than for gains, that is, losses loom larger than gains and people have a tendency towards loss aversion.

One of the essential features of PT is that the overall value of a prospect is based on changes in a decision-maker's wealth to a reference point rather than on final wealth. The new component of the PT is reference dependence. People's value judgements are highly dependent on the reference point, that is, people are more focused on changes in their value (utility) states than the states themselves. The reference point depends on the aspect of framing. This is a bigger deviation from utility and probability weighting. This deviation is of major empirical importance. More than half of risk aversion observed has nothing to do with utility curvature or with probability weighting Instead, it is generated by loss aversion, the main empirical phenomenon regarding reference dependence (Wakker, 2010).The essence of reference dependence is not that outcomes are modelled as changes with respect some reference point (initial wealth). The reference point can change during the analysis. Deviations from a variable reference point are a major breakaway from final-wealth models.

The value function in the prospect theory has the power function form

$$v(x) = x^{\alpha}, \qquad for\ x \geq 0, \qquad 3\text{-}3$$

$$v(x) = -\lambda(-x)^{\beta}, \qquad for\ x < 0, where\ \alpha > 0, \beta > 0, \lambda > 0 \qquad 3\text{-}4$$

With loss aversion $= \lambda > 1\ and\ \alpha = \beta$ (Al-Nowaihi, Bradley & Dhami, 2008), so that losses are over weighted relative to gains. The opposite, $\lambda < 1$ seeking for gains with little attention for losses. $\lambda = 2$ is interpreted as decision weight, losses are taken twice as important for decisions as gains. Overweighting can be deliberate if a decision maker thinks that more attention should be paid to losses than gains, or perceptual, with losses simply drawing more attention. The parameters of the value function, as based on the empirical study Kahnemann & Tversky (1992), were: $\alpha = \beta = 0.88$, and $\lambda = 2.25$.

Probability Weighting Function

People might be using weighting schemes to evaluate probability of the outcomes. People have tendency to over weighing very low probability events and under weighing very high probability events and this depends on how the problem is presented or framed. Also, they put a decision weight based on the probability of that outcome. This distortion of probability is captured by PT's probability weighting function. Figure 3.4 shows the probability weighting function.

The weighting function:

$$\pi(p) = \frac{p^{\gamma}}{(p^{\gamma}+(1-p)^{\gamma})^{1/\gamma}} \qquad 3\text{-}5$$

The probability weighting function has a psychological interpretation. People are less sensitive to the changes of probability as they move away from "certainly will not happen" or "certainly will happen". This notion is called *diminishing sensitivity*

(Tversky & Kahneman, 1992). Under this principle, the changes of probability near the endpoint are larger than in the middle of the scale. This means that people are risk-seeking for low probability gains and risk averse for low probability losses. In contrast, in the medium to high probabilities are given less weight than they would receive using EUT. Such underweighing makes decision maker be risk averse for medium to high probabilities gains, and risk seeking for medium to high probabilities losses.

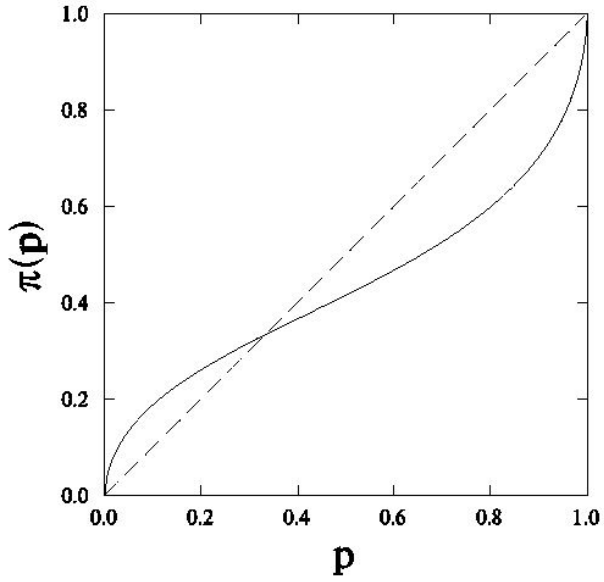

Figure 3.4 Probability weighting function

Evaluation of the prospect

V is defined in two scales π and v. Accordingly, the first scale, π, associates with each probability p a decision weight $\pi(p)$, which reflects the impact of p on the overall value of the prospect. The second scale, v, assigns to each outcome x a number $v(x)$, which reflects the subjective value of the outcome. These scales are combined to form the basic equation of the theory which determines the overall value of regular prospect.

The total prospect is

$$V(x) = \sum_{i \in X} v(x_i)\pi(p_i) \qquad 3\text{-}6$$

3.6 Summary

Decision-making can be studied from two different theoretical approaches. Normative decision making theory focuses on how decisions should be made in order to be rational, while descriptive decision-making theory aims to find tools and methodologies to help people to make the best decision, and relates to people's beliefs and preferences as they are, not as they should be.

Utility Theory (UT) combines a mixture of descriptive and normative elements as an attempt to model human choice under risk. Expected Utility Theory (EUT) states that the decision maker (DM) chooses between risky or uncertain prospects by comparing their expected utility values, i.e., the weighted sums obtained by adding the utility values of outcomes multiplied by their respective probabilities. Subjective Expected Utility Theory (SEUT) is a method in decision making theory in the presence of uncertainty, and Von Neumann-Morgenstern Theory (VNMT) is a method in the presence of risk. Von Neumann and Morgenstern provided an axiomatic system: a set of conditions that were necessary and sufficient for expected utility. Axioms have a descriptive as well as a normative benefit. However, its predictions did not conform to the experimental or empirical evidence. Prospect theory is the most accepted alternative method of explaining choices made by individuals under conditions of risk.

In ACO, an ant makes a decision to move from the current state to the next one using a random proportional rule or pseudo-random rule strategies. These strategies might influence the performance of the ACO algorithms.

In this thesis, PT as a descriptive theory of decision making under risk will be applied in the state transition strategy in ACO algorithms. These algorithms will be explained in Chapter Four and Chapter Five.

Chapter 4 A New Framework for Ant Colony Optimization for Discrete Optimization Problems

In this chapter, prospect theory will be implemented in ACO algorithms as a new framework for decision making in ACO to select the candidate solution during the construction phase. This implementation will be carried out for discrete problems. The TSP will be used as a standard problem and a water distribution system will be used as a discrete problem with constraints. The experimental setup and the result are presented in the chapter.

4.1 Introduction

As mentioned in Chapter 2, there are three phases to ACO algorithms, namely the construction phase, the pheromone update phase, and the optional daemon phase. In the construction phase, the ants iteratively construct candidate solutions on which they may deposit pheromones. An ant constructs a candidate solution starting with an empty solution and iteratively adds the solution component until the complete candidate solution is generated. Each point at which an ant has to decide which solution component to add to its current partial solution is called as a decision point. After the solution construction is completed then the ant will enter the pheromone update phase. In this phase, the ant gives feedback on the solution that has been constructed by

depositing pheromone on that solution component. Normally, solution components which are part of better solutions or are used by many ants will receive a higher amount of pheromone and, hence, will more likely be used by the ants in the future iteration of the algorithm. To avoid the search getting stuck, typically before the pheromone trails to get to reinforce, all pheromone trails are decreased by a factor ρ, which is called as the evaporation factor (Stützle & Dorigo, 1999).

The ants' solutions are not guaranteed to be optimal with respect to local changes and, hence, more explorations are needed to search in global changes. However, the balance between exploration and exploitation has to be considered carefully. Excessive exploitation will reduce the diversity of the solution by focusing only in the neighbourhood and lead the search to local optima quickly. At the same time, extreme exploration will increase the diversity of solutions but slow down the search's speed. So, the improper balance will lead to ineffective algorithms. To set the balance between the exploitation and exploration the simple ε-greedy approach is used. One problem with this approach is how to decide the optimal ε. By setting ε fixed the ants will not learn from their experience; the designer has to decide how many ants do the exploitation and to do the exploration.

Several attempts have been made to improve the ε-greedy approach. Guo, Liu & Malec (2004) proposed ε as a function of the number of ants having passed the edge ij during the latest iteration, the number of artificial ants, and the number of the cities. Consequently, the more iterations their algorithm runs, the closer ε is to one, and the more quickly the algorithm will converge. In other words, their algorithm encourages the ants to explore at the beginning of the iterations and to exploit at the end of the iterations. Guo et al. (2004) adapted ε using a new Q-learning algorithm, where ε is

reduced during the learning process. They argued that their algorithm will not only improve the ability of the ants to attain the new knowledge, but will also allow the algorithm to avoid a performance decrease due to the constant value of ε (and, thus, constant probability of exploration).

In this thesis, prospect theory was introduced in order to improve the exploitation-exploration capability of the ants. In prospect theory, highly probable outcomes are under-weighted. This action creates risk seeking behaviour. The low probable outcomes are over-weighted, and this results in risk aversion behaviour. These behaviours cannot be achieved with linear probability as happens in the normal ACO.

In this new framework, the way of calculating of the probability of choice in the construction phase is different to the normal ACO. However, the rest of the phases is not changed, so this framework can be easily adapted to any variant of ACO because almost all the variants of ACO have the same formula for calculating their probability of choice.

By introducing prospect theory, the exploration and exploitation are inherited in the decision process. Highly probable outcomes that lead to stagnation are under weighed, and very low probability outcomes are not neglected so the exploration is being encouraged.

The standard and proposed framework for ACO are presented in Figure 4.1.

4.2 Formulation of Prospect Theory in TSP (Travelling Salesman Problem)

In TSP, as mentioned in Chapter 2, at the decision point the ant will choose the next location using the equation below:

$$P_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}.[\eta_{ij}]^{\beta}}{\sum_{j\in\mathcal{N}_k(s)}[\tau_{ij}]^{\alpha}.[\eta_{ij}]^{\beta}}, \forall j \in \mathcal{N}_k(S) \quad 4\text{-}1$$

where τ_{ij} is the amount of pheromone trail on the edge(i,j); η_{ij} is the a priori available information, and $\mathcal{N}_k(S)$ is the set of feasible features that have not been visited by ant k. The amount of pheromones (τ) is related to the quality of the solutions, so, the better the solution, in this case the shorter the tour length, the higher the value of τ. The heuristic information (η) is related to the distance between neighbouring cities; the closer the distance the higher η.

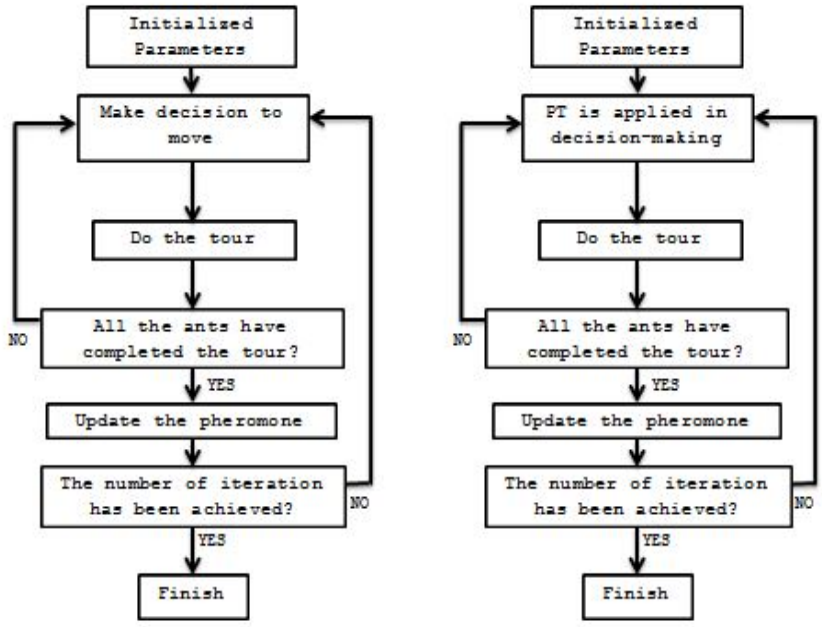

Figure 4.1 The Framework of ACO. On the left is the standard ACO and on the right is ACO-PT

As introduced in Chapter 3, prospect theory has two phases which are the editing phase and the evaluating phase. Below, we will introduce these phases of ACO algorithms.

Editing Phase

The heuristic information (η_{ij}) was considered as an outcome and the amount of pheromone of the edge (τ_{ij}) as the probability of the outcome. So, if a city is closer to its neighbour then it has a higher outcome, and if a level of pheromone on the edge is high then the probability that the ant will choose that edge is high.

The pheromone is always in the gain region, so the reference point is zero. With this frame setting, the ants are always in the gains domain and the behaviour towards the risks is only influenced by the probability weighting. For very low probabilities the ant is risk averse; however, for higher probabilities the ant is risk seeking. By doing this, the exploration of the ants is increased and the local optima can be avoided. In the normal ACO the pheromone importance parameter α is greater than zero and its value has to be defined usually by trial and error, and it may be influenced by the problems that have studied. The parameter α in the prospect theory is found by empirical study, typically equal to approximately 0.88 and always less than 1.00. When the exponent $\alpha < 1.00$, the curve will accelerate negatively (if $\alpha = 1.00$, the function would be linear; and if $\alpha > 1.00$, it would accelerate positively). This means the effect of higher intensity pheromones is reduced, which will increase the exploration by the ants.

Evaluation Phase

The outcome τ_{ij} is the amount of the pheromones in the edge(i, j), the reference point is zero, so the outcome is always in the gains region, and the value function is

$$v(\tau_{ij}) = \tau_{ij}{}^{\alpha}, where\ \alpha = 0.88 \quad (4\text{-}2)$$

The probability is derived from the heuristic information, η_{ij},

$$P\eta_{ij} = \frac{\eta_{ij}}{\sum_{l\in\mathcal{N}(s)}\eta_{il}}, where\ \mathcal{N}(s) is\ a\ set\ of\ feasible\ solution \quad (4\text{-}3)$$

The weighting function is

$$\pi(P\eta_{ij}) = \frac{P\eta_{ij}{}^{\gamma}}{(P\eta_{ij}{}^{\gamma} + (1 - P\eta_{ij})^{\gamma})^{1/\gamma}}, where\ \gamma = 0.68 \quad (4\text{-}4)$$

and the prospect is

$$V_{ij}^{k} = v(\tau_{ij})\pi(P\eta_{ij}) \quad (4\text{-}5)$$

This prospect will be used as a basis for choosing the next movement. The way we make a selection remains the same.

4.3 Formulation of Prospect Theory in Water Distribution System (WDS) Problems

In many areas that are undergoing rapid urbanization, methods for the evaluation of a nation's water distribution system (WDS) need to consider not only the rehabilitation of existing one but also its future development to serve expanding population centres. Both the adaptation of existing technologies and the development of new technologies will be required to improve the efficiency and cost-effectiveness of future and existing WDSs and to anticipate the industrial growth.

The optimization of WDSs is loosely defined as the selection of the lowest cost combination of appropriate component sizes and component settings such that the

criteria of demands and other design constraints are satisfied. (Maier, et al., 2003) These constraints are often considered as a pre-specified range defined by maximum and minimum velocity in the pipe and the nodal pressure. Here, each pipe is a decision point at which the diameter of the pipe is to be determined. The components of the decisions set, $D = \{d_1, d_2, \ldots d_i \ldots, d_n\}$ are therefore, the existing pipes of the network, where d_i represents the i-th pipe of the network. The pipe diameters are usually selected from a set of commercially available diameters, $\varphi = \{\varphi_{ij}\}$, which may or may not be the same for all the pipes, then $\varphi = (\varphi_1, \varphi_2, \ldots \varphi_J)$ would present the list of available options at each and every decision point of the problem. If C is defined as the unit length cost of the pipe with diameter φ_j, cost c_{ij} associated with option φ_j at decision point d_i can now be calculated as the product of per unit cost uc_j and the length le_i of the link under consideration. The cost of a trial solution, $f(\varphi)$, may be low, however, the solution may or may not be feasible.

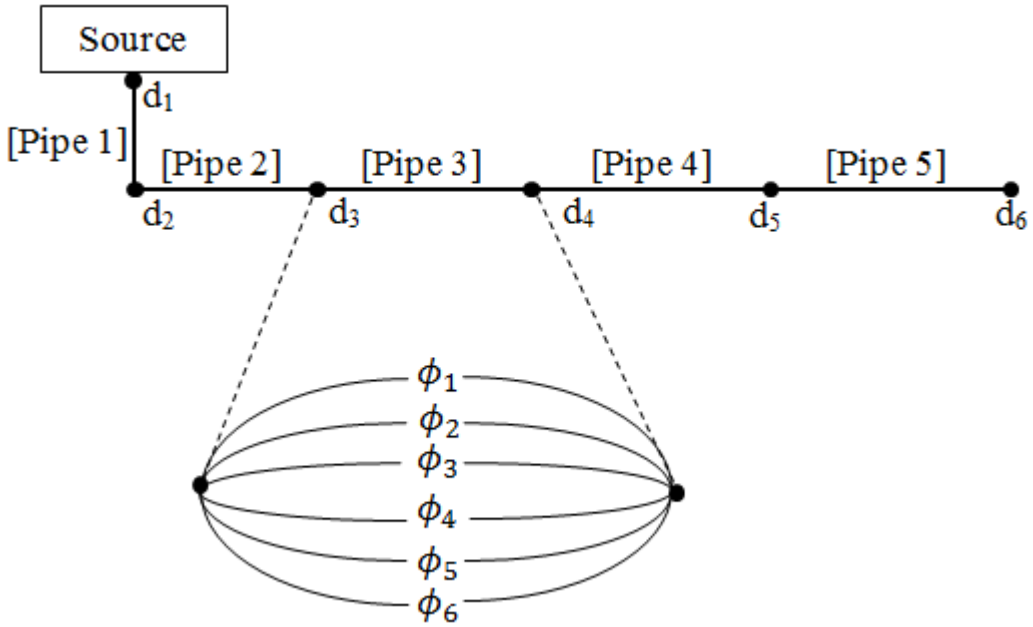

Figure 4.2 Representation of WDS Problem (Source: Maier et al., 2003)

ACO has been used to solve water distribution system optimization problems and has been shown to be extremely competitive. ACO was applied initially for WDS management by Maier et al. (2003). Zecchin et al. (2007) studied the parameterization of ACO for WDS and suggested guidelines for selecting the parameters of ACO. López-Ibáñez, Prasad & Paechter (2008) used ACO for optimizing the operation of pumping in WDS. Christodoulou & Ellinas (2010) proposed an ACO algorithm for efficient routing of piping networks for improving efficiency and robustness.

The way to formulate WDS problems is different from other combinatorial optimization, such as the travelling salesman problem, due to the nature of the constraints. In the travelling salesman problem, the only constraints are that each city must be visited once only and that the finishing point must be the same as the starting point. To solve this situation, tabu lists can be used to store only feasible solutions. However, the constraints that need to be satisfied in the optimal design of WDS are of a different nature. The feasibility of a particular trial solution (e.g., whether minimum pressure constraints have been satisfied) can only be assessed after it has been constructed in its entirety, and consequently, the constraints cannot be taken into account explicitly during the construction of trial solutions (Maier, et al., 2003).

The optimization of WDSs can be formulated as a constrained minimization problem. Essentially, WDS optimization involves the selection of the lowest cost set of diameters of pipes within the network such that the design pressure constraints are not violated.

ACO, as for all evolutionary algorithms (EAs), is unable to deal directly with constrained optimization problems as it cannot follow constraints that separate feasible regions of the search space from infeasible regions. The standard technique of EA to

handle these problems is to convert constrained problems to unconstrained problems using penalty functions.

The only immediate way to do this is either to impose a “death penalty” (infeasible solutions are never rewarded) or relax the constraints by introducing soft penalties. (Runarsson & Yao, 2000). The problem now becomes how to set the penalty so that the optimal balance in the search between focussing on feasibility and emphasizing optimality is achieved. Many different schemes for static, dynamic, and adaptive penalties have been explored in the context of EA (Coello Coello, 2002), but, unsurprisingly no single technique stands out that is best independently of the problem. What makes matters more difficult is that the relaxation method can, in principle, not guarantee that solutions are truly feasible (as opposed to having only a minimal amount of constraint violation). Unfortunately, in practice, constraints are often hard constrains and must be satisfied exactly.

An alternative to these widely-used ways of handling constraints in constructive meta-heuristics is to incorporate some look-ahead algorithm into the construction phase so that only feasible solutions are generated. A survey carried out by Dorigo & Stützle (2003) notes that as yet only a very simple look-ahead procedure had been investigated with ACO. Construction methods using look-ahead tables, of course, need to be specific to the problem constraints. A generalized construction method based on a declarative mathematical model of the problem would be desirable. The hybridization of ACO with Constraint Programming (CP) achieves this goal (Khichane, Albert, & Solnon, 2008).

In WDS optimization problems the solutions were assessed as being infeasible by EPANET which was the benchmark hydraulic analysis tool for this work. EPANET is public domain software that may be freely copied and distributed.

As in the TSP, we only focus on how to formulate prospect theory in the decision making process during the construction phase. We do not pay attention to how to update the pheromone or how to control the pheromone. The pheromone update and the way the pheromones are controlled depend on the variant of the ACO that has been used.

In the WDS the probabilistic choice of the ants for all variants is done by:

$$P_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}} \qquad 4\text{-}6$$

In WDS, the amount of pheromones, τ, depends on the cost of the solutions (network cost), $NC(\varphi)$, and heuristic information, η, reflects the cost of choosing the pipe in one section. The summary of ACO to be applied to the WDS problem is presented in Table 4-1.

Table 4-1 The summary of the conversion of general ACO Problem to WDS Problem (Source, Zecchin et al., 2005)

General ACO problem formulation		WDSP equivalent	
Element	Symbol	Element	Symbol
Path	S	Design	Ω
Edge	(i,f)	Option	$option_{i,j}$
Set of edges available from decision point i	θ_i	Set of options available for pipe i	$\begin{pmatrix} option_{i,1},\cdots \\ \cdots, option_{i,NO_i} \end{pmatrix}$
Objective function	$f(S)$	Network cost	$NC(\Omega)$

As in the TSP, the role of prospect theory in WDS is to improve the decision making process during the selection of a candidate solution by each ant. The processes are the same, that is the editing and the evaluation phase.

Editing Phase

The outcome is derived from the cost functions and the probability of the outcomes is obtained from their violation. If the violation is high then the probability of the outcome is low. The reference point is set to be the average of the cost function of trial solutions. So, if the solution cost of a particular ant is lower than the reference point then the ant considers this situation as a loss, and if the solution cost of a particular ant is higher than the reference point, the ant accepts this situation as a gain.

Evaluation Phase

The value function is:

$$v(x) = \begin{cases} x^{\alpha}, & for\ x \geq 0 \\ -\lambda(-x)^{\beta}, & for\ x < 0 \end{cases} \quad where\ \alpha > 0, \beta > 0, \lambda > 0 \qquad 4\text{-}7$$

where $x = NC(\varphi) - \overline{NC(\varphi)}$, $\alpha = \beta = 0.88\ and\ \lambda = 2.27$; $\overline{NC(\varphi)}$ is the average of all cost functions.

The probability function:

$$p_i = \frac{\phi_i}{\sum_{l \in S} \phi_l} \qquad 4\text{-}8$$

where $\phi_i = 1/g_i$ and $g(x)$ is the constraint function.

The weighting probability function is

$$\pi(p_i) = \frac{{p_i}^{\gamma}}{({p_i}^{\gamma} + (1 - p_i)^{\gamma})^{1/\gamma}}, where\ \gamma = 0.68 \quad \text{4-9}$$

So, the prospect for each choice is

$$V_{ij}^{k} = v(\tau_{ij})\pi(P\eta_{ij}) \quad \text{4-10}$$

The best solution is the candidate solution with largest prospect. This best solution will be used to calculate the amount of pheromone.

4.4 Experimental Results

4.4.1 TSP Problems

Comparisons were made between ACO and ACO-PT. We have compared the same variant of ACO algorithms, in order to evaluate the influence of PT alone on the ACO, especially on the state transition. We have not compared ACO-PT to the other type of algorithms because we only want to examine the effects of PT on the algorithm as a result of changing its decision-making method and not because of other modifications such as a daemon or local search. We tested the AS, ACS and MMAS algorithms with PT in TSP problems.

Comparison of AS algorithm and the AS-PT algorithm

In this experiment the test problem presented in Dorigo, Maniezzo & Colorni (1996) was used to compare the performance of AS and AS-PT. Dorigo et al. (1996) showed that AS ant cycle was the best AS algorithm variant. This variant was used to compare with the proposed AS-PT. The experiment was run for 10 times, the same as in the experimental setup in Dorigo et al. (1996). The results were presented in Table 4-2.

Table 4-2 Performance of AS and AS-PT on Oliver30 problem

Problem name	AS Algorithms (Ant Cycle)		AS-PT Algorithms	
	Average results	Best result	Average results	Best result
Oliver30 (30-city problem)	424.250	423.741	423.965	423.741

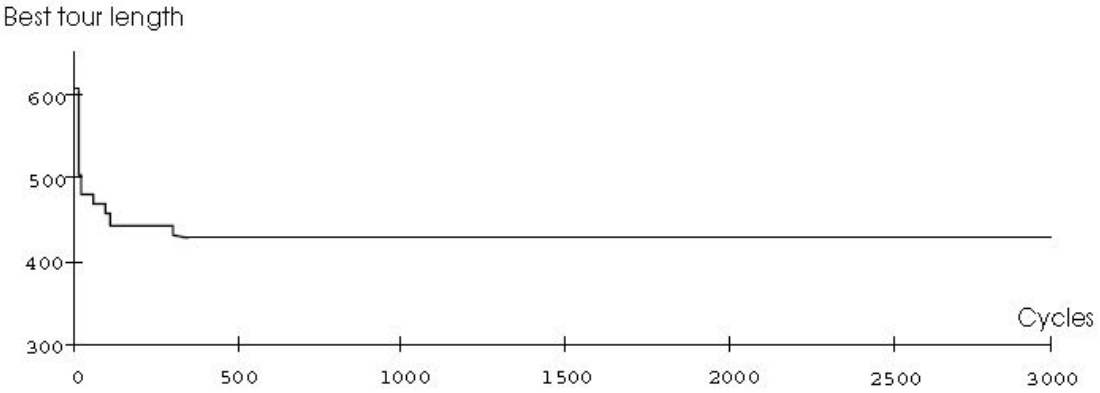

Figure 4.3 Evolution of best tour length (Oliver30). Typical run (Source: Dorigo et al., 1996)

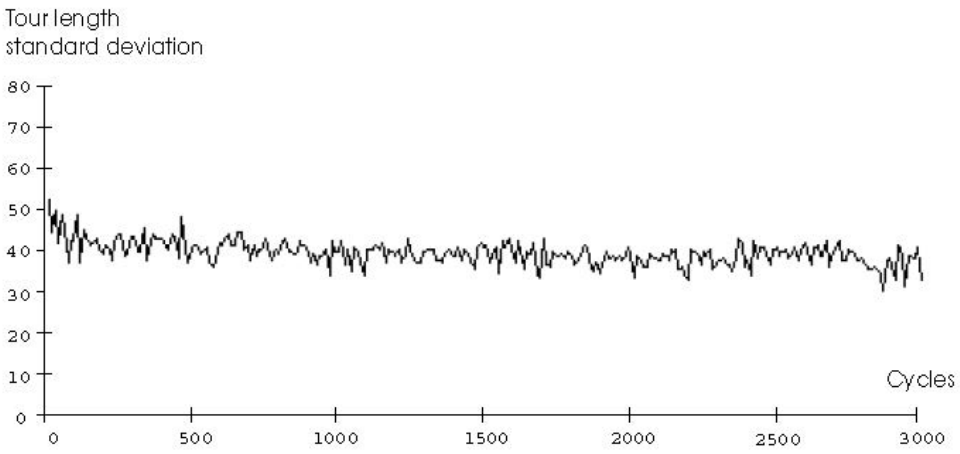

Figure 4.4 Evolution of the standard deviation of the population's tour lengths (Oliver30). Typical run (Source: Dorigo et al., 1996)

Table 4-2 shows the results of AS and AS-PT for Oliver30 test problem. The average and the best results were compared. Figure 4.5 shows that the iterative time of AS-PT algorithm was shorter than the cycle times of AS algorithm as shown in Figure 4.4. Also, the standard deviation of AS-PT algorithm was larger than AS algorithm as shown in Figure 4.5. It can be concluded that the search exploration of AS-PT algorithm is wider than the search exploration of AS Algorithm. Obviously, prospect theory improves AS performance for Oliver30 problem both in finding the best solutions and the length of time for finding them.

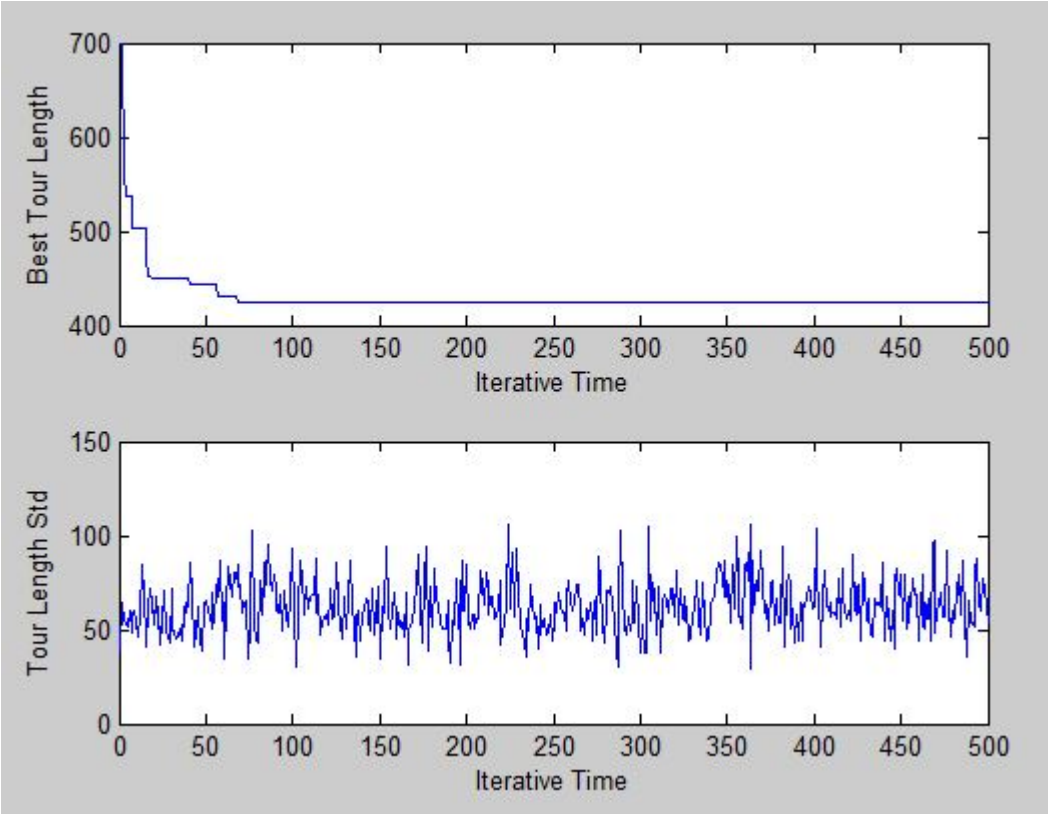

Figure 4.5 Best Tour Length and its standard deviation for the best result from the experiment.

The source code for Ant System in Matlab can be found in Appendix A and this can be downloaded from http://www.sourcecodedownloads.com/1077518/. We modified the selection strategy using PT.

Comparison of ACS algorithm and ACS-PT algorithm

In all experiments of the following sections the numeric parameters are set to the following values: $\beta = 2, q_0 = 0.9, \alpha = \rho = 0.1$, $\tau_0 = (n \cdot Lnn) - 1$, where Lnn is the tour length produced by the nearest neighbour heuristic and n is the number of cities. These values were obtained by a preliminary optimization phase, in which we found that the experimental optimal values of the parameters were largely independent of the problem, except for τ_0 for which, as we said, $\tau_0 = (n \cdot Lnn) - 1$. The number of ants used is $m = 10$. Regarding their initial position, ants are placed randomly, with at most one ant in each city.

It is important to test ACS on both random and geometric instances of the TSP because these two classes of problems have structural differences that can make them difficult for a particular algorithm and at the same time easy for another one.

Table 4-3 reports the results on the random and geometric instances. The number of runs was 2,500 iterations using 10 ants. ACS-PT almost always offers the best performance.

Table 4-3 Comparison of ACS with ACS-PT on 50=city problems random instances, and Oliver30 of TSP

Problem Name	ACS			ACS-PT		
	average	std dev	best	average	std dev	best
City Set 1	61.38	0.03	61.34	61.33	0.05	61.16
City Set 2	61.04	0.02	61.01	60.03	0.07	59.61
City Set 3	60.03	0.03	59.97	57.81	0.03	57.23
City Set 4	65.14	0.04	65.05	63.35	0.02	63.05
City Set 5	58.82	0.02	58.64	58.38	0.03	58.23
Oliver30	424.74	2.83	423.74	424.65	1.65	423.74

Discussion

We believed that, in order to escape from local minima, and to increase the speed of the search, one possible extension of ACS is the introduction of PT to the state selection during the solution construction. The behaviour of PT is to overweigh the lower probability; consequently the low level of pheromone but better objective function will get more attention. On the hand, PT is to underweigh the higher probability, so the high level of pheromone but worse objective function will be discouraged to be chosen. With this setting, the ants will try to improve the exploration to find the new solution. As a consequence, ants never converge to a common path. This fact, which was observed experimentally, is a desirable property given that if ants explore different paths then there is a higher probability that one of them will find an improving solution than there is in the case that they all converge to the same tour (Deneubourg, et al, 1990).

This evidence can be seeing in Figure 4.6 and Figure 4.7, at the beginning of the iteration process the length of the global solution of ACS-PT (Figure 4.7) is higher than the length of ACS. As a result, the ants take risks to explore more solutions in the beginning of the iteration process. ACS-PT also presents faster convergence when compared to the ACS alone.

We can conclude that PT greatly influences the selection function in state transition strategy and affects the performance of an ant colony algorithm greatly.

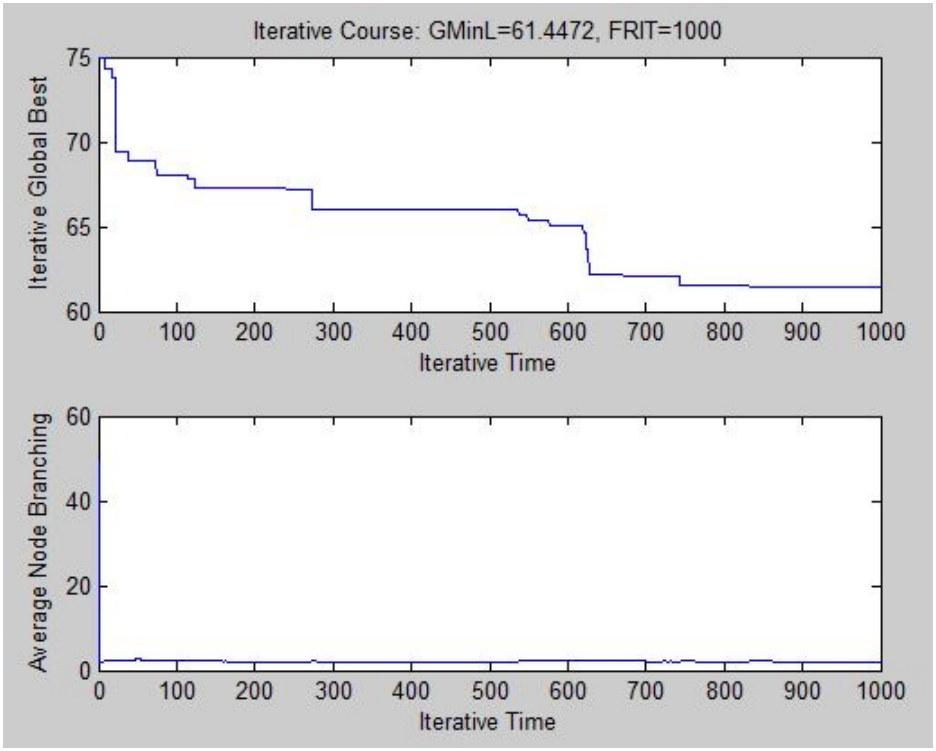

Figure 4.6 Typical result for ACS

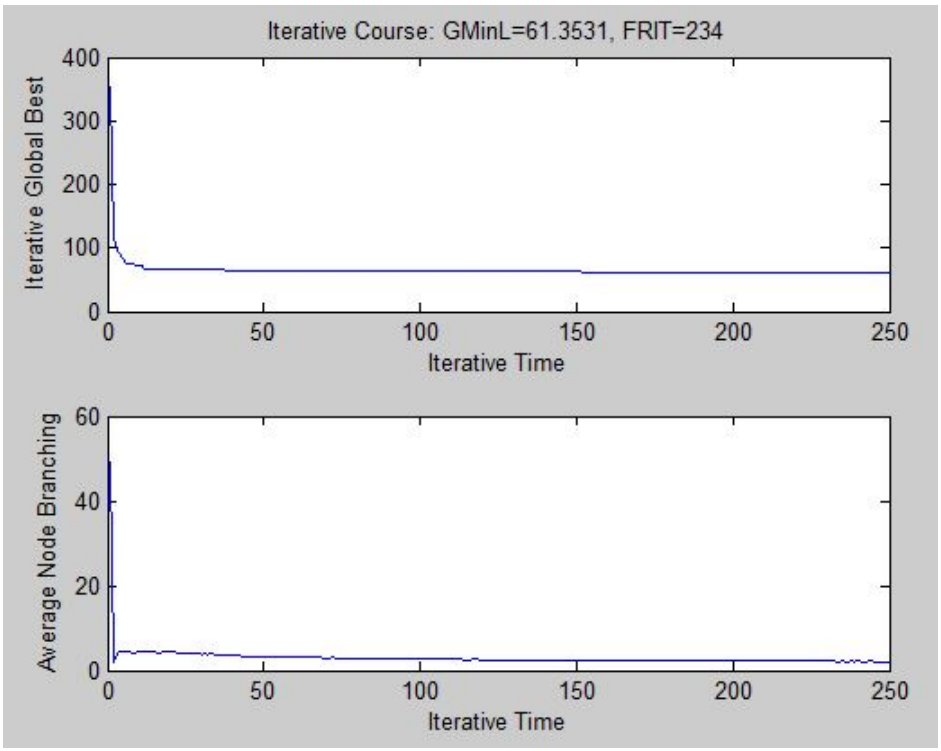

Figure 4.7 Typical result for ACS-PT

4.4.2 Water Distribution Problems

New York Water Tunnel Problem (NYWTP)

The New York Tunnel consists of 20 nodes connected via 21 tunnels. The system is formed into two primary tunnels, City Tunnel No. 1 and City Tunnel No 2. In order to meet the minimum allowable total head requirements, the network is to be modified using duplicate tunnels in parallel with the existing tunnels. The objective of the optimization is to decide which of the 21 tunnels need to be duplicated, and if so, what the diameter of tunnel should be constructed. For each duplicate tunnel there are 16 allowable options; 15 different diameter sizes and the option of no tunnel. More details about NYTP and the duplicate tunnel options are given in Maier, et al. (2003). The NYTP configuration is presented in Figure 4.8.

For these experiments, the MMAS variant of ACO is used with parameters as follows: the number of ants, $m = 84$, $p_{best} = 0.01$, the importance of pheromone parameter $\alpha = 1.0$, the importance of heuristic information parameter $\beta = 0.5$, and the evaporation factor $\rho = 0.98$. (Zecchin A. C., et al., 2003). For MMAS-PT, the parameters $\alpha = \beta = 0.88$ and $\lambda = 2.27$. The results are based on 20 test runs.

The known-optimum solution is \$38.64 million found first by ACOA (a version of ACO with a scheme to MMAS in Maier, et al. (2003)).

Table 4-4 Comparison of algorithm performance for the New York Tunnels Problem. Performance statistic are ordered as follows; minimum, [mean] and {maximum}

Algorithm	Best-cost ($M), (%deviation from known optimum	Evaluations number
MMAS[a]	38.64, (0.0) [38.7, (0.2)] {38.95,(0.8)}	23,542 [24,978] {27,285}
MMAS-PT	38.64 (0.0) [38.75, (0.28)] {38.88, (0.66)	16.044 [20.645] {24.964}

a. (Zecchin A. C., et al., 2003)

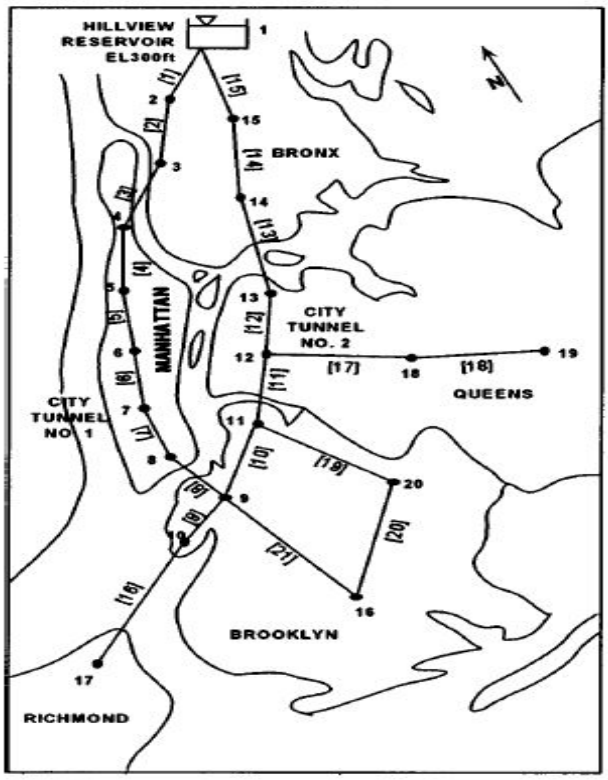

Figure 4.8 New York Tunnel Problem (Source: Maier et al., 2003)

As can be seen in Table 4-4 the least costly feasible solution for MMAS and MMAS-PT was $38.64 million, but the evaluations number for MMAS-PT is better than MMAS. The typical results for MMAS and the proposed MMAS-PT for NYWTP are shown in Figure 4.9 Evolving process of objective function using MMAS for NYWTP and Figure 4.10, respectively. The best fitness for each of iterations of MMAS-PT for NYWTP is shown in Figure 4.11.

MMAS-PT performed better than MMAS for this case study. It can be deduced that the PT encourages exploration and yields an improved performance MMAS-PT algorithm. MMAS-PT put greater emphasis on exploration in the risk solution and high objective function lead to better solutions.

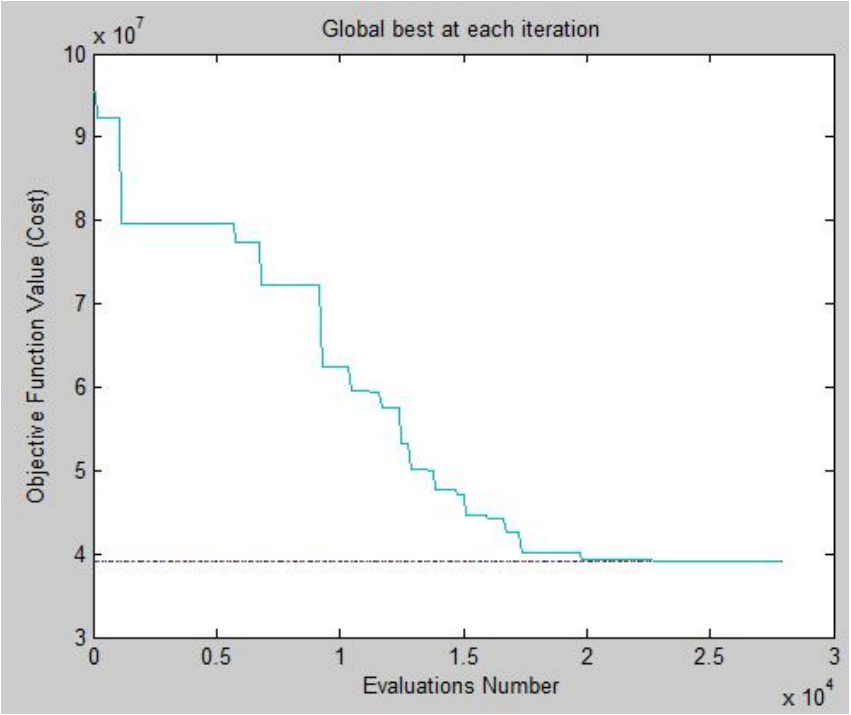

Figure 4.9 Evolving process of objective function using MMAS for NYWTP

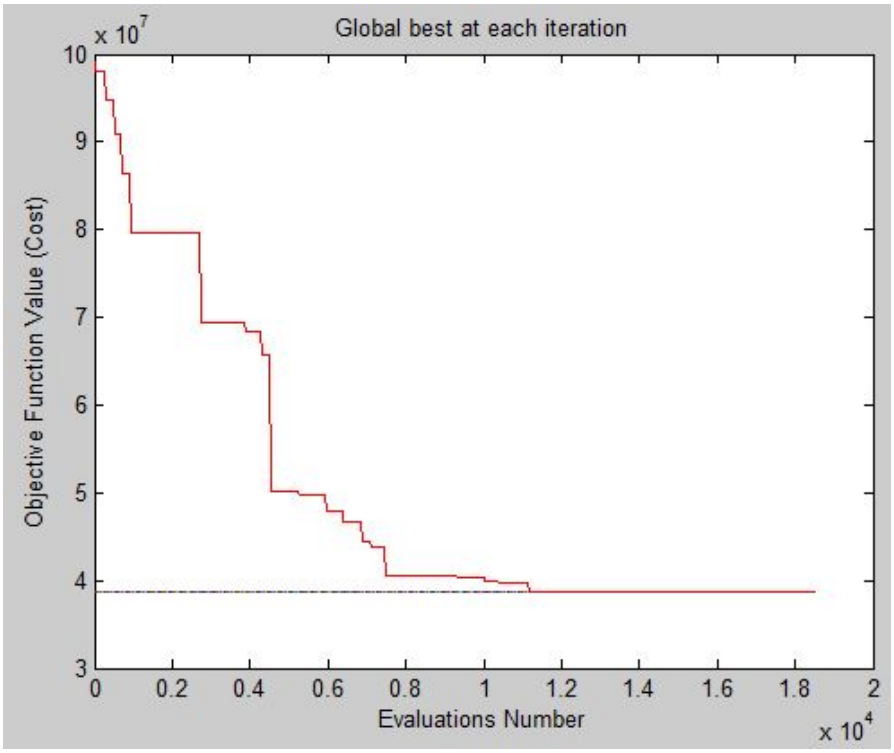

Figure 4.10 Evolving process of objective function using MMAS+PT for NYWTP

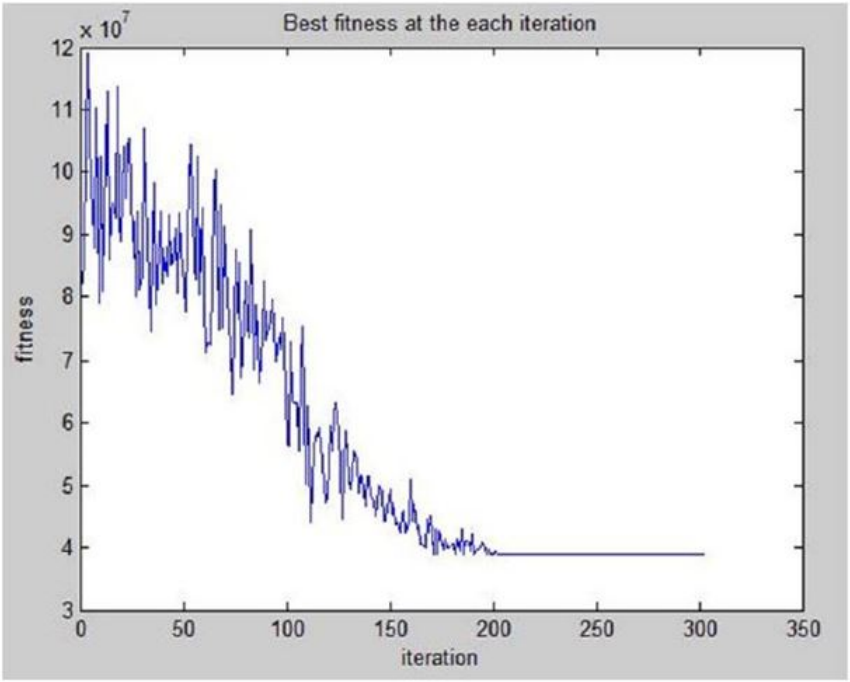

Figure 4.11 Best fitness of each of iterations for MMAS for NYTP

Hanoi Network Problem (HNP)

The water distribution system in Hanoi (Vietnam) comprises 34 pipes, 32 nodes organized in three loops. This network was originally investigated by Fujiwara & Khang (1990), and was later used by several authors (Cunha & Sousa, 1999; Eusuff & Lansey, 2003; Liong & Atiquzzaman, 2004; Zecchin, Maier, Simpson, Leonard & Nixon, 2007). The system is gravity fed by a single reservoir and has only a single demand case (see Wu et al. (2001) for network details). For each link there are six different new pipe options where a minimum diameter constraint is enforced. The Hanoi Network is presented in Figure 4.12.

By using only the diameters originally used by Fujiwara & Khang (1990) the solution presented by Cunha & Sousa (1999) achieved the lowest cost of $6.06 m. However, pressures originated from these references were lower than 30 m in some network nodes, for α equal to 10.67 (default value according to EPANET2). The dimensioning solution by Lenhsnet model provided an optimal system cost of $6.42 m. (Gomes, Bezerra, De Carvalho, & Salvino, 2009)

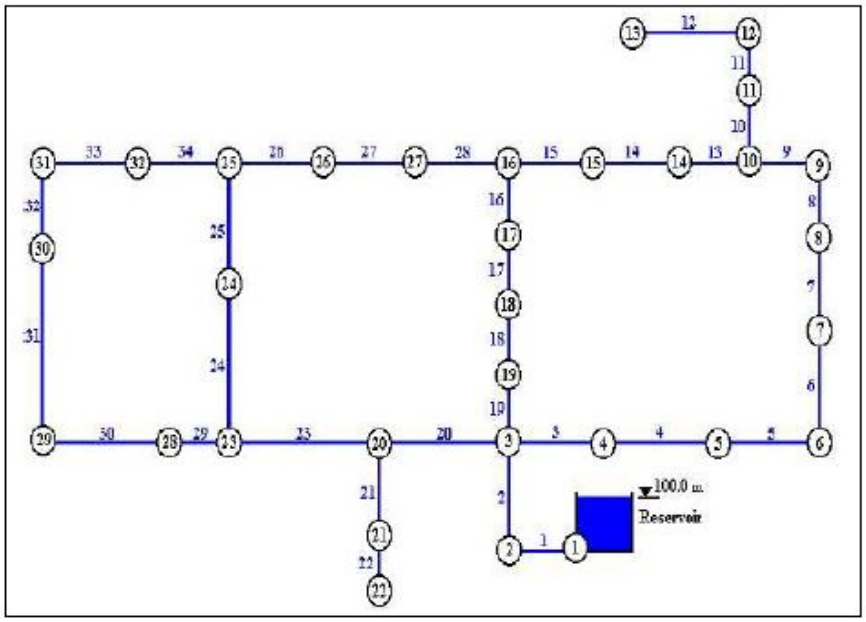

Figure 4.12 Hanoi Network Problem

This case study has a problem size of approximately 2.87×10^{26} possible designs. For MMAS, m = 83 and $p_{best} = 0.9$. Q was set to 1.1×10^{7}.

Table 4-5 Comparison of algorithm performance for Hanoi Network Problem. Performance statistics are ordered as follows; minimum, [mean] and {maximum}

Algorithm	Best-cost ($M), (% deviation from known optimum)	Search time (evaluation number)
MMAS[a]	6.412,(3.7) [6.685, (8.1)] {6.905, (11.7)}	25,092 [31,595] {38,693}
MMAS-PT	6.201, (2.5) [6.434, (7.2)] {6.432, (3.7)}	24,034 [28,667] {30,213}

a. (Zecchin A. C., et al., 2003)

For the HNP, MMAS-PT was unable to find known-optimum in the literature, however, its performance is better than MMAS. The reason for this is that MMAS-PT algorithm takes a risk to search the infeasible region to lead the search to the feasible region. The typical results of MMAS and MMAS-PT are shown in Figure 4.13 and Figure 4.14 respectively.

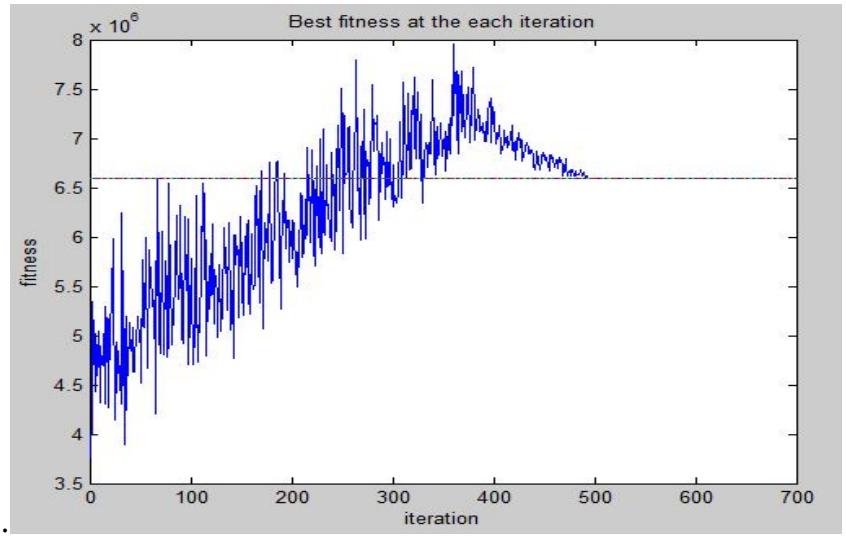

Figure 4.13 Typical result for MMAS for HNP

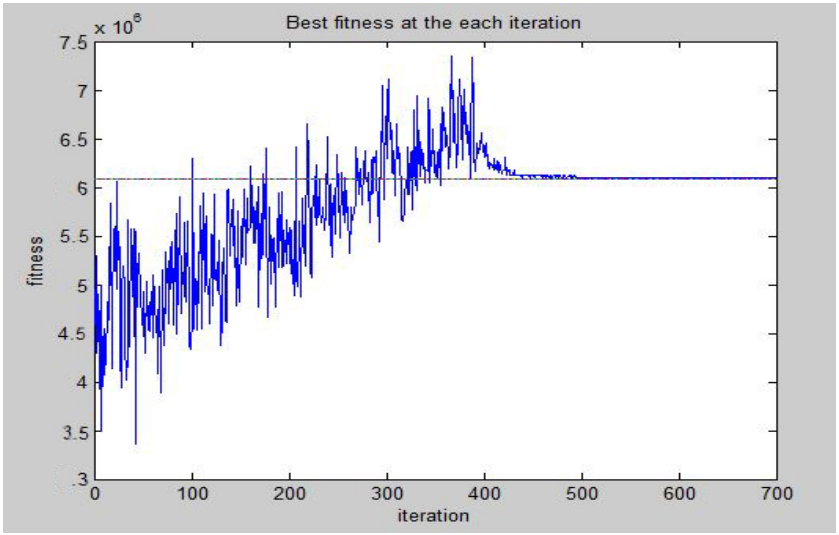

Figure 4.14 Typical result for MMAS-PT for HNP

4.5 Summary

Prospect theory has been implemented in the ant colony optimization algorithms. The results obtained in the experiment show that the performance of the ACO has been improved. Tests were carried out on the travelling salesman problem, water distribution system for combinatorial optimization problems. AS, and MMAS were used.

The results obtained for the travelling salesman problem and water distribution problem outperform the original ACO. The implementation of prospect theory in ACO can be done in two ways. Firstly, it can be used by inserting it during the construction phase to decide where to move next. Secondly, we use prospect theory in the update pheromone phase, when we want to choose which suitable solution is to be used to update the pheromone; in this case, the best solution so far is not always chosen as a candidate to update the pheromone.

In TSP, prospect theory is used in making a decision about which city will be visited next during the construction phase. The probability is derived from the heuristic information of the problem; while the outcome is obtained from the amount of pheromones that have been deposited in each edge. The prospect of each decision point is calculated by multiplying the value function and weighting probability. The outcomes are framed in the gain region, so the reference point is zero. In this scheme, the weighting probability takes on the main role for determining the behaviour of decision making.

In the water distribution system problem, the New York Water Supply Tunnel Problem and Hanoi Network Problem were used as study cases. The implementation of prospect theory in WDS is in the update pheromone phase to choose which candidate solutions to update the pheromone. This scheme eliminates the use of the penalty function. The solution is selected through its prospects which depend on its objective functions and the violation of the constraints. This approach avoids the problem of using the penalty function, which can over-penalize or under-penalize. The constraint violation is used to obtain the probability; a solution with more violations will have smaller probability to

be chosen. The value function is derived from the objective function. The reference point is calculated by averaging the objective costs of all solutions obtained.

In choosing the candidate solution for WDS, if all solutions are unfeasible or feasible then a solution is chosen probabilistically based on their prospect. However, if one of the solutions is feasible the solution candidate has to be chosen from the feasible one even if the prospect of the unfeasible solution is better. This approach is to ensure the final solution is always feasible if there is one.

Chapter 5 A New Framework for Ant Colony Optimization for Continuous Optimization Problems

In this chapter, the concept of ACO for solving continuous optimization problems will be reviewed and how to implement prospect theory ACO algorithms as a new framework for decision-making will be explained. Tests and evaluation of the proposed framework will be performed using mathematical benchmark functions.

5.1 Introduction

Initially, ACO algorithms were intended to solve combinatorial optimization problems with decision variables in discrete domains. Since, for optimization problems in the real world the decision variables can be in discrete or continuous domains, attempts to extend ACO algorithms as a general optimization tool for tackling discrete and continuous optimization problems have been considered (Blum, 2005).

There has been relatively little research into ant colony algorithms as applied to continuous space optimization problems. This is due to the fact that the course of the ant colony algorithms is usually applied to discrete object problems. In order to enable ant colony algorithms to be applied to continuous space optimization problems, ant colony algorithms with discrete space must be improved.

There are two possibilities for extending ant algorithms for continuous optimization problems. The first method uses simplified direct simulation of real ant behaviour, and the second method is to extend the ACO metaheuristic to explore continuous spaces. This extension can be done by a suitable discretization or by probabilistic sampling of a search space.

In this thesis, prospect theory has been used as a new framework for decision making in the ACO metaheuristic with probabilistic sampling in continuous optimization problems for the first time.

5.2 ACO for Continuous Domains (ACOR)

We have mentioned ACO for continuous optimization problems in Chapter 2. In this section, we will explain ACOR (Socha & Dorigo, 2008) in detail as a basis algorithm to be improved by adding PT in its decision making rule during the solution construction phase.

Socha & Dorigo (2008) proposed to use the Probability Density Function (PDF) instead of the discrete probability function in the decision making process in their algorithm for solving continuous optimization problems. This PDF is obtained from sampling a set of solutions called a solution archive. In the first iteration this solution archive is filled with k randomly generated solutions. In the next iteration, m new solutions are generated to add to the previously found k solutions so that the number of solutions found are $k + m$. These solutions are then sorted according to their quality, from the best first to worst. After that, the best k solutions are stored in the archive. The solution archive is shown in equation 5-1.

$$Solution\ archive, T = \begin{bmatrix} s_1^1 & s_1^2 \dots & s_1^i \dots & s_1^n \\ s_2^1 & s_2^2 \dots & s_2^i \dots & s_2^n \\ & & \vdots & \\ s_j^1 & s_j^2 \dots & s_j^i \dots & s_j^n \\ & & \vdots & \\ s_k^1 & s_k^2 \dots & s_k^i \dots & s_k^n \end{bmatrix}_{k \times n} \quad 5\text{-}1$$

where n is the number of variables or solution components, k is the number of solutions that are stored in the archive and s_j^i is the i-th solution component of j-th solution. Each row in the solution archive corresponds to a found solution, and by calculation of the objective function, the quality of each solution is measured.

Socha & Dorigo (2008) used a multimodal one-dimensional probability density function based on a Gaussian kernel which comprises of a weighted sum of several Gaussian functions g_j^i, where j is a solution index and i is a coordinate index. The Gaussian kernel for coordinate i is:

$$G^i(x) = \sum_{j=1}^{k} w_j g_j^i(x) = \sum_{j=1}^{k} w_j \frac{1}{\sigma_j^i \sqrt{2\pi}} e^{-\frac{(x-\mu_j^i)^2}{2{\sigma_j^i}^2}} \quad 5.2$$

where $j \in \{1, \dots, k\}, i \in \{1, \dots, D\}$ with D being the problem dimensionality, and $w_j$is a weight with the ranking of solution j in the archive, $rank(j)$. The weight is calculated using a Gaussian function:

$$w_j = \frac{1}{qk\sqrt{2\pi}} e^{-\frac{(rank(j)-1)^2}{2q^2k^2}} \quad 5.3$$

where q is a parameter of the algorithm. When q is small, the best rank solutions are strongly preferred, when it is large, the probability becomes more uniform.

In the solution construction process, each coordinate is treated independently. Each ant chooses probabilistically one of the solutions in the archive according to its corresponding weight.

$$p_j = \frac{w_j}{\sum_{i=1}^{k} w_i} \tag{5.4}$$

Then, the algorithm samples around the selected solution component s_j^i using a Gaussian PDF with $\mu_j^i = s_j^i$, and σ_j^i equal to

$$\sigma_j^i = \xi \sum_{r=1}^{k} \frac{\left|s_r^i - s_j^i\right|}{k-1} \tag{5.5}$$

which is the average distance between the i-th variable of the solution s_j and the i-th variable of the other solutions in the archive, multiplied by a parameter ξ. This parameter has an effect similar to that of the pheromone evaporation rate in ACO. The higher the value of ξ, the lower the convergence speed of the algorithm.

The solution construction process is repeated m times for each dimension $i = 1, \dots, n$. After each solution construction, the pheromone update is performed by adding m newly generated solutions to the solution archive T and then discarding the same number of worst solutions, so that the total number of solutions in the archive remains k solutions. This process keeps the number of the better solutions the same in the archive, but better quality, so that the ants will be guided effectively in the search process. An outline of ACOR is given in Algorithm 1.

Algorithm 1 Outline of ACOR

Input: k, m, n, q, ξ and termination criterion
Output: The best solution found.
  Initialize and evaluate k solutions
  // Sort solutions and store them in the archive
  $T = Sort(S_1 \ldots S_k)$
  while Termination criterion is not satisfied **do**
   // Generate m new solutions
    for $l = 1$ to m **do**
    // Construct solution
      for $i = 1$ to n **do**
        Select Gaussian g_j^i according to weights
        Sample Gaussian g_j^i with parameter μ_j^i, σ_j^i
      end for
      Store and evaluate newly generated solution
    end for
    // Sort solutions and select the best k
    $T = Best(Sort(S_1 \ldots S_{k+m}), k)$
end while

Figure 5.1 ACOR Algorithm (Liao, et al., 2011)

5.3 Formulation of Prospect Theory for Continuous Unconstrained Problems

In ACO, ants choose their paths by both exploitation of the accumulated knowledge about the problem and exploration of new edges. In ACOR, pheromone intensity is modelled using a Gaussian PDF, the mean is the last best global solution and its variance depends on the aggregation of the promising areas around the best one. So it contains exploitation behaviour. On the other hand, a Gaussian PDF permits all points of the search space to be chosen, either close to or far from the current solution. So, it also contains exploration behaviour. It means that ants can use a random generator with a Gaussian PDF as the state transition rule to choose the next point to move to (Nobahari & Pourtakdoust, 2005).

The new proposed framework of ACOR based on PT modifies the state transition during the construction phase. So, instead of directly using a random generator it is using PT to choose the next point. PT looks at two parts of this state transition: the editing, or framing phase, and the evaluation phase. In the editing phase, wealth, which is this case the objective function, is framed to losses or gains to the reference point, while in the evaluation phase the decision maker will choose options which are influenced by subjected value and perceptual likelihood (McDermott, 2001, p. 20).

The procedure to build the proposed ACOR-PT algorithm is described in the algorithm 2, while a more detailed explanation is presented below:

Calculate the reference point

During the solution construction process, a solution is constructed by searching its variables one by one until all variables are found. To do this, first, a solution is chosen with a probability to its weight. Then, the algorithm samples around the selected solution using a Gaussian PDF. In our new framework, the selected solution is not chosen from the probability of its weight alone but also from its objective function. The reference point is the average of the value of the objective function of all solutions in the solution archive:

$$Reference\_point = mean(f(S_1 \dots S_k) \quad 5.6$$

Calculate the value function

The value function is derived from the objective function, $f(S_1 \dots S_n)$. If the objective function of a particular solution is less than the reference point then it is in the loss

region, conversely, if it is larger than the reference point then the solution is in the gain region.

The value function is formulated as:

$$v(x_l) = \begin{cases} x_l^{\alpha}, & for\ x_l \geq 0 \\ -\lambda(-x_l)^{\beta}, & for\ x_l < 0 \end{cases} \quad where\ \alpha > 0, \beta > 0, \lambda > 0 \qquad 5.7$$

where $x_l = f(S_l) - Reference\_point\ ,\ \alpha = \beta = 0.88\ and\ \lambda = 2.27; l = 1, \dots k$

Calculate the probability weighting function

Prospect theory is used to choose the best solution. In the normal ACO the solution is chosen based on the probability function:

$$p_l = \frac{\omega_l}{\sum_{r=1}^{k} \omega_r} \qquad 5.8$$

$$\omega_l = \frac{1}{qk\sqrt{2\pi}} e^{-\frac{(l-1)^2}{2q^2k^2}} \qquad 5.9$$

Which, essentially, defines the weight to be a value of Gaussian function with argument l, mean 1.0, and standard deviation qk, where q is a parameter of the algorithm. When q is small, the best-rank solutions are strongly preferred, and when it is large, the probability becomes more uniform.

The weight of the ranking of the solution archive in the new framework is changed to the weighting probability function as follows:

$$\pi(p_l) = \frac{{p_l}^{\gamma}}{\left({p_l}^{\gamma} + (1 - p_l)^{\gamma}\right)^{1/\gamma}}, where\ \gamma = 0.68 \qquad 5.10$$

Calculating the prospect

The prospect for each choice is

$$V_l = v(x)\pi(p_l) \qquad 5.11$$

The best solution is the candidate solution with largest prospect. This best solution will be used to calculate the amount of pheromone.

5.4 Experiment Setup

In this experiment, the comparison is not based on CPU time but on the number of function evaluations needed to achieve a certain solution as a criterion of comparison. Socha & Dorigo (2008) argue that this approach gives several advantages: it solves the problem of algorithms being implemented using different programming languages; it is insensitive to the code-optimization skills of the programmer (or to the compiler used); and it allows the results obtained on different machines to be compared easily. The drawback of this approach is that it does not take into consideration the time-complexity of the algorithms compared. However, in view of the other numerous disadvantages over of using CPU time as a criterion, this approach is an acceptable methodology.

The use of the number of function evaluations as a criterion allows us to run the experiments only with modified ACO-PT and compare the results to those found in the literature and to ensure a fair comparison; we replicate the experimental setup used by ACOR algorithm.

Algorithm 2 Outline of ACOR-PT

Input: k, m, n, q, ξ and termination criterion
Output: The best solution found.
  Initialize and evaluate k solutions
  // Sort solutions and store them in the archive
  $T = Sort(S_1 \ldots S_k)$
  while Termination criterion is not satisfied **do**
  //Generate m new solutions
  for $l = 1$ to m **do**
  //Construct solution
  for $i = 1$ to n **do**
  // Calculate the reference point
  $Reference\_point = mean(f(S_1 \ldots S_k)$
  // Calculate the value function of PT
  $(obj\_val_1^i \ldots obj\_val_k^i) = value\_function(Reference\_point, (s_1^i \ldots s_k^i))$
  // Calculate the probability weighting function of PT
  $(wp_1^i \ldots wp_k^i) = wp\_function(p(w_1^i \ldots w_k^i))$
  // Calculate the prospect of PT
  $(pros_1^i \ldots pros_k^i) = pros\_funct((obj\_val_1^i \ldots obj\_val_k^i), (wp_1 \ldots wp_k))$
  $maxprospect = \max(pros_1^i \ldots pros_k^i)$
  Select Gaussian g_j^i according to $maxprospect$
  Sample Gaussian g_j^i with parameter μ_j^i, σ_j^i
  end for
  Store and evaluate newly generated solution
  end for
  // Sort solutions and select the best k
  $T = Best(Sort(S_1 \ldots S_{k+m}), k)$
end while

Figure 5.2 The Proposed ACOR-PT Algorithm

Table 5-1 Summary of the parameters of ACOR (Source: Socha & Dorigo (2008))

Parameter	Symbol	Value
No. of ants used in an iteration	m	2
Speed of convergence	ξ	0.85
Locality of the search process	q	10^{-4}
Archive size	k	50

To compare ACOR and ACOR-PT, we have run this algorithm on a number of test functions. The summary of the parameters of ACOR is presented in Table 5-1. The list of tests along with the number of dimensions used and the initialization interval are

presented in Appendix A. The numbers of runs to perform this test were 20 and the stopping criteria were the number of iterations or the following function:

$$|f - f^*| < \epsilon_1 f + \epsilon_2 \qquad 5.11$$

where f is the value of the best solution found by the algorithm, f^* is the (known a priori) optimal value of the given problem, and $\epsilon_1$and ϵ_2 are the relative absolute error respectively, we used $\epsilon_1 = \epsilon_2 = 10^{-4}$.

Table 5-2 shows the results obtained using prospect theory in ACOR. Obviously, for some results ACOR-PT outperforms ACOR but in others this is not the case, such as for Diagonal Plane, Rosenbrock (R2 and R5), Hartmann ($H_{6,4}$) and Griewangk (Gr_{10}).

The best result for Sphere, Ellipsoid, Brainin, Goldstein and Price, Hartman ($H_{3,4}$) and Shekel are presented in Figure 5.3, Figure 5.4, Figure 5.5, Figure 5.6, and Figure 5.7, respectively.

Table 5-2 Results obtained by ACOR (Socha & Dorigo, 2008) compared to ACOR-PT

Test function	ACOR (median number of function evaluation)	ACOR-PT (median number of function evaluation)
Diagonal Plane	170	520
Sphere	1507	890
Ellipsoid	11570	673
Brainin RCOS	857	210
Goldstein and Price	393	102
Rosenbrock (R_2)	816	2319
Rosenbrock (R_5)	2570 (97%)	-
Hartmann ($H_{3,4}$)	342	55
Hartmann ($H_{6,4}$)	722	1032
Shekel ($S_{4,5}$)	793 (57%)	437
Shekel ($S_{4,7}$)	748 (70%)	173
Shekel ($S_{4,10}$)	715 (81%)	172
Zakharov (Z_2)	293	12
Griewangk (Gr_{10})	1390 (61%)	-

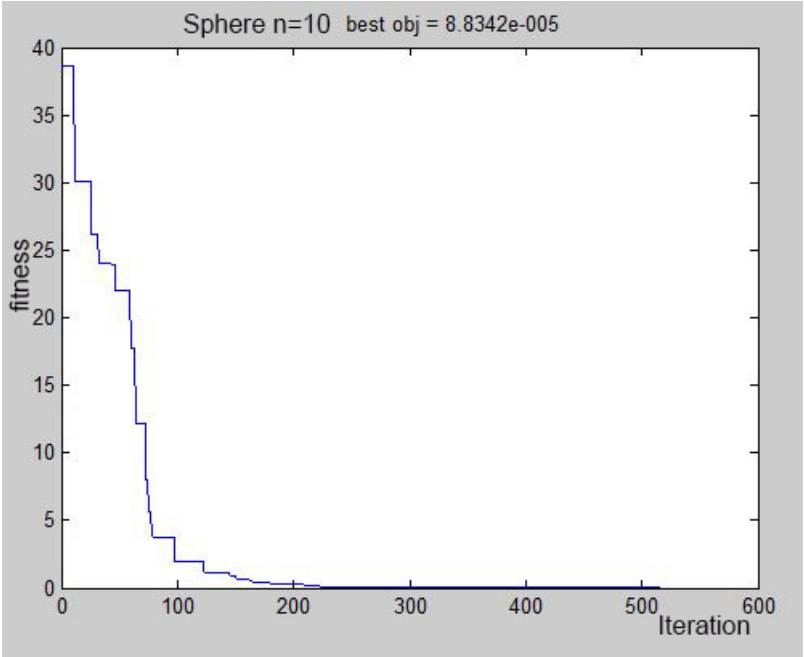

Figure 5.3 The best result for Sphere Function, number evaluation = 840

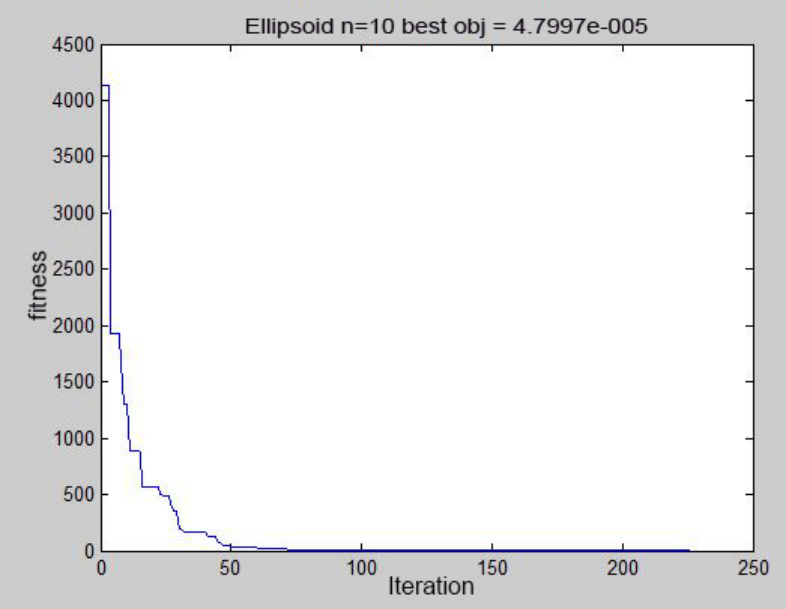

Figure 5.4 The best result for Ellipsoid Function, number evaluation = 300

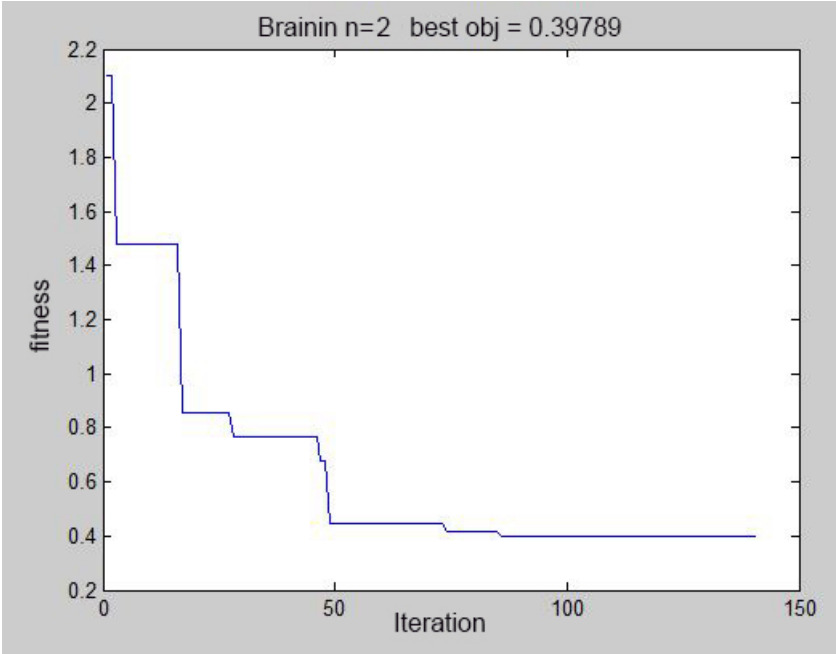

Figure 5.5 The best result for Brainin Function, number evaluation = 170

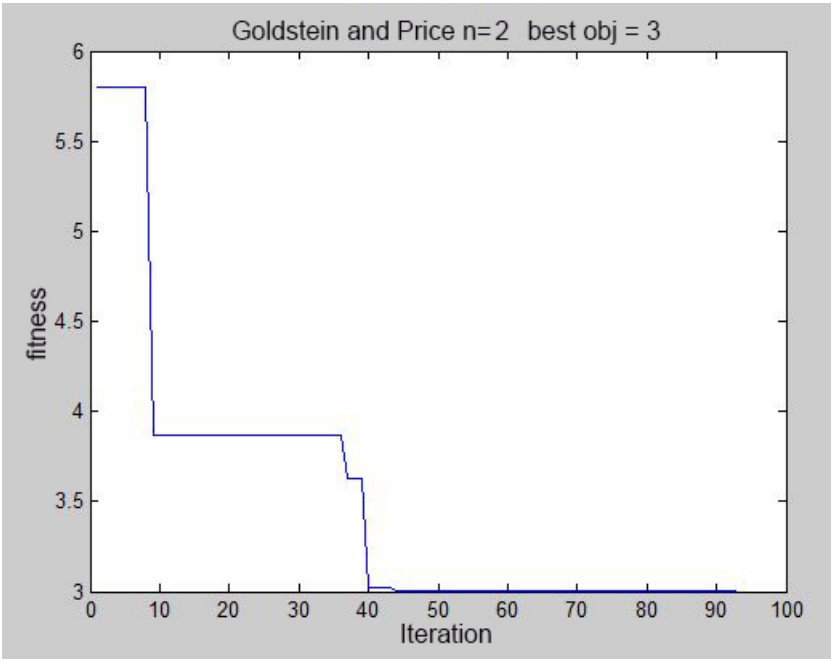

Figure 5.6 The best result for Goldstein and Price Function, number evaluation = 88

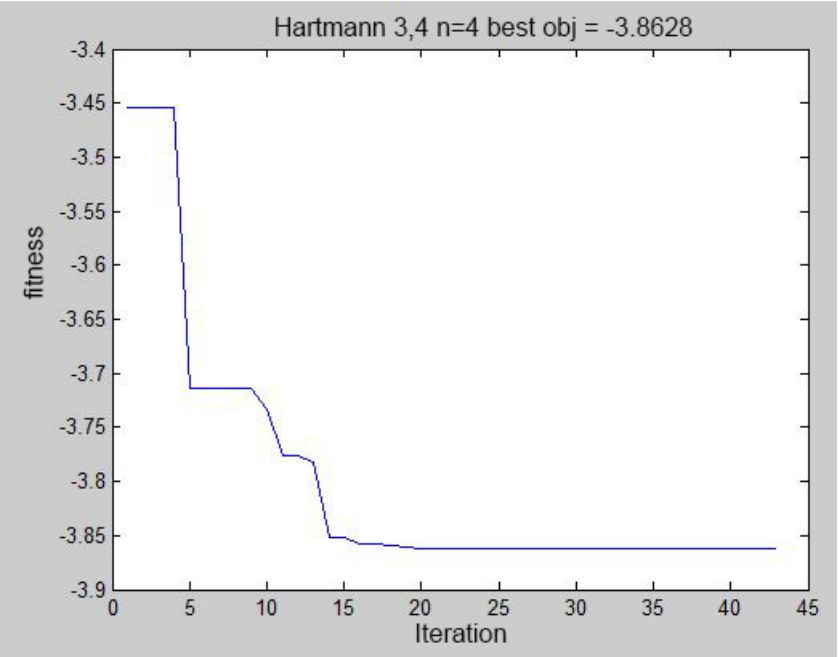

Figure 5.7 The best result for $Hartmann_{3,4}$ Function, number evaluation = 40

The performance of the proposed algorithm has been compared with ACO. The results indicate that the state transition strategy of the proposed algorithm has greatly improved the performance of the algorithms.

Conclusions are drawn as follows:

1. State transition strategy for the ACO algorithm not only used random-proportional rule and pseudo-random proportional rule, but, moreover, different state transition strategies can be obtained by using prospect theory (PT).
2. A selection function in the state transition strategy affects the performance of an ant colony algorithm greatly. Prospect theory can be used as a new framework as a selection function.

5.5 Summary

Ant Colony Optimization algorithms were initially developed to solve combinatorial optimization problems and due to the principle and the structure of the algorithms, it

was difficult to apply them directly to continuous optimization problems. Some methods have been developed to apply ACO to continuous optimization problems but some basic concepts of ACO have been altered in these methods. There are two types of ant-based algorithms for solving continuous optimization problems. The first type directly simulates the behaviour of the ants which are continuous in the real world. This type of algorithm changes the basic method of ACO algorithms which constructs the solutions step by step. The second type is to discretize the variable solutions and use probabilistic sampling. In this type, the accuracy is poor, while the probabilistic sampling still follows the basic principles of the ACO.

Prospect theory has been used to improve decision making in ACOR and has shown the improvement of the previous ACOR in some test functions.

Chapter 6 An Analysis of the New Framework of Ant Colony Optimization Compare to Genetic Algorithm and Particles Swarm Optimization

This chapter provides a comparison between the new framework of ACO and Genetic Algorithms and Particles Swarm Optimization.

6.1 Introduction

The new proposed framework is primarily to improve the state transition strategy of ACO during the solutions construction phase. This improvement is done by introducing the prospect theory into the decision-making process of selection of a new solution. By inserting this prospect theory into the construction phase, the solutions with low probability will get more weight and the ones with the high probability will get less weight compare to the ones with moderate probability. By doing so, the exploration of new solutions will be encouraged.

The exploration of the new solutions is an important characteristic in evolutionary algorithms such as genetic algorithm and swarm intelligence algorithms such as particles swarm optimization to improve its searching capabilities and to avoid to be

trapped in the local optima. However, we should have internal mechanism to avoid the algorithms explore too far; they need to balance between their exploration and exploitation. In this chapter, we will compare the exploration and the exploitation characteristics of genetic algorithm and particles swarm optimization algorithm with ant colony optimization algorithm.

6.2 Comparison between ACOAs and Genetic Algorithms (GAs)

An Evolutionary Algorithm (EA) is a population-based optimization approach inspired by a natural evolution, which consists of reproduction, mutation, recombination and selection of population candidate solutions, and through the evolution, a better solution to a problem to be found. GAs and ACOAs generate population-based candidate solutions to find an optimal solution. The number of trial (candidate) solutions in GAs depends on the size of population, while in ACOAs it depends on the number of ants. Consequently, the population size in GAs is analogous to the number of ants in ACOAs, and, whereas, the number of generations in GAs is analogous to the number of iteration in ACOAs. Also, one cycle in ACOAs is equivalent to the evaluation of an individual member of a population in GAs.

In both GAs and ACOAs, trial solutions are generated using biological evolutionary inspired methods. Genetic algorithms adopt the principle of survival of the fittest, while ACOAs are based on the foraging behaviour of ant colonies. Both algorithms construct trial solutions based on a probabilistic approach. In GAs, this process is controlled by the probabilities of crossover and mutation, and in ACOAs, by pheromone intensities and local heuristic information.

In order to encourage wider exploration, GAs applied mutation or crossover operators, while ACOAs utilized pheromone evaporation mechanism. However, it should be noted that pheromone evaporation in ACOAs is deterministic, whereas mutation in GAs is stochastic. The new proposed framework uses the prospect theory to improve the exploration. GA will update all chromosomes by their fitness at each generation (iteration). In other words, if a particular chromosome has better fitness (shorter path distance) than other chromosomes, then that particular chromosome is more likely to win the competition and clone itself. Thus, a chromosome with good fitness has a much higher probability than other inferior chromosomes to appear in the next generation.

GAs and ACOAs are difference in terms of the way the trial solutions are generated. In GAs, a trial solution is represented by encoded, as a bit string, of each problem parameters. The new solutions are obtained by modifying previous solutions. Consequently, the memory of the system is embedded in the actual trial solutions. In ACOAs, system memory is contained in the environment, rather than the trial solutions. As ants step through this environment, trial solutions are constructed incrementally based on the information contained in the environment.

Improved trial solutions are obtained by modifying the environment via a form of indirect communication called stigmergy (Dorigo et al. 2000). Consequently, ACOAs may have advantages over GAs in certain types of applications. For example, ACOAs may be more useful in an operational setting, where the system is dynamically changing. By maintaining pheromone trails and continuously exploring different options, ants have backup plans and are therefore prepared to respond to changes in their environment.

ACOAs may also have an advantage in situations where sequential decisions have to be made in order to construct a trial solution, and the selection of some component solutions restricts subsequent choices. When the environment is dynamically changed, the fitness function is difficult to be properly defined, and GA may converge towards local optima. Operation on dynamic sets is difficult for GA, so it is not appropriate choice for constraint based optimization problem.

ACO algorithm has an advantage over the Genetic algorithm in terms of the algorithm execution time. No matter how many obstacles are present, this algorithm does not devote an in ordinate amount of time in iteration process. GA Strengths, convergence by survival of the fittest, well described, widely used, mutation and recombination, binary operators allow for fast operations. GA Weaknesses, discretization of the solutions, integer approach to optimization of continuous functions, no improvement of individual, - does not find local minimum.

One of the main reasons for ACO's good performance is due to the behaviour of the algorithm i.e. how it works to initialize the population. The way GA initialize the population is based on random approaches where the next node to be visited will determine easily by choosing any of random adjacent feasible node. With this random process, the algorithm needs to go through the process of selection and some other process to choose the optimal node which will contribute to increment of time and iteration especially when the number of nodes is increasing. It was different with ACO where it has an efficient state transition rules which is efficient and can skip to the process of finding optimal path. With this efficient initialization approach, it will simultaneously help ACO to reduce the time and iteration while finding optimal path either in a simple environment or complex environment.

In addition, the quality of population in each generation will also influence algorithm performances. In ACO, the global and local updating approaches is efficient because it will improve the number of optimal path in each generation, this will make the solution faster with fewer number of iteration. However, this differs with GA where GA will carry the good population to produce the next child in the next generation however there is no guarantee to improve the solution rapidly as the child is produced in random approach. Thus the child may come from good categories or worst categories which will then influence the population of the next generation. Hence, this will affect the time and number of iterations that GA takes to find the solution will be greater than ACO's. Furthermore, when the number of nodes increases, the size of chromosomes length also need to be adjusted based on the requirement in each case. Small number of nodes only needs a small number of lengths to allocate the paths within the chromosomes while the complex numbers of nodes will need a complex number of chromosomes. The increment of length will affect the whole process of GA to find optimal path because the population of the next child in each generation will be produced based on the cross over and mutation process. During this process, the point to be cross and mutate will be determined randomly based on the length of the chromosomes. The more the length, the more possibility of GA to have a variety of population which will cause the process of finding optimal path more challenging and simultaneously will contribute to the increment of time and iteration. It is different with ACO where the length of chromosomes size will not influence the next ant's population as it is based on heuristic and pheromone value carried by the previous ants. Therefore, length will increase the time ants need to traverse from one node to another nodes and not influences the next node to be traversed by the ants.

With the increment of length usually GA also need to increase the population in order to get the optimal path. This will cause the process of GA to find path to become slower. In contrast with GA, for ACO, it is not necessary to increase the population because it will not affect the process. Thus this helps ACO to minimize the time and number of iterations. Based on the reason and the results obtained above, ACO is practical to be use either in small or complex number of nodes compared to GA. This is because it can optimize the path in an efficient way compared to GA in ever different complexity of environment. ACOA can find the optimal path and satisfy the optimization criteria at a faster rate than GA.

In addition, the settings for ACOA are compared to GA. In ACOA, only the length needs to be changed in each cases, however for GA, when settings are changed, there is a need to ensure the balancing of the value of population, length and convergence criteria. ACOA can also be used to optimize not only global paths but also local paths. GA and ACOA were performed successfully in dynamic environments to find a path that has optimization criteria. The performance of both algorithms has been studied globally when they were applied for routing in various dynamic environments. The results showed that ACOA in comparison with the genetic algorithm, in a lesser time or more quickly, has a lower number of iterations in each dynamic complex environment. (reference). Moreover, the adaptability of the ant algorithm parameters in the complex, dynamic environment is easier than genetic algorithm. Advantages and limitations of both algorithms can spread and combine a variety of applications in planning the robot's path in future.

6.3 Comparison between ACOAs and Particle Swarm Optimization (PSO) Algorithms

The ACO is inspired by the foraging behaviours of ant colonies. At the core of this behaviour the indirect communication between the ants enables them to find short paths between their nest and food sources. This characteristic of real ant colonies is exploited in ACO algorithm to solve, discrete optimization problems. The PSO technique modelled on the social behaviours observed in animals, such as bird flocking and fish schooling. It has gained increasing popularity among researches and practitioners as a robust and efficient technique for solving difficult robust and population-based stochastic optimization problems (Deb & Padhye, 2010). So, Both ACOA and PSO algorithms are the optimization algorithms by implementing swarm behaviour.

In terms of applications, ACOA is more applicable for problems where source and destination are predefined and specific. At the same time PSO is an optimization algorithm in the areas of mutliobjective, dynamic optimization and constraint handling. ACOA is more applicable for problems that require crisps results and PSO is applicable for problems that are fuzzy in nature.

The main concept of PSO resembles the way birds travel to find sources of foods, which all birds in a flock are influenced by each other. At the beginning of the optimization process, all particles are initially randomised in the search space of the problem. The positions of the particles inside the swarm (or population) are treated as different solution to a given problem. These particles then move, or travel through the search space looking for new and/or better solutions to the problem. During the development of several movements, only the most promising particle can share

information onto the other particles to accomplish the optimum solution to the problem (Rini & Shamsuddin, 2011).

The communication mechanism among particles in PSO is rather direct without altering the environment, while, in ACO adopted an indirect communication mechanism among ants, called stigmergy, which means interaction through the environment. So, ACO is more suitable than PSO for the applications where the environment is dynamic.

ACO was originally used to solve combinatorial (discrete) optimization problems, but it was later modified to adapt continuous problems. PSO was originally used to solve continuous problems, but it was later modified to adapt binary/discrete optimization problems. In ACO, a solution space is typically represented as a weighted graph, called construction graph. In PSO, a solution space is typically represented as a set of n-dimensional points. ACO is commonly more applicable to problems where source and destination are predefined and specific, while PSO is commonly more applicable to problems where previous and next particle positions at each point are clear and uniquely defined. In ACO, the objective is generally searching for an optimal path in the construction graph while, in PSO, is generally finding the location of an optimal point in a Cartesian coordinate system.

6.4 Summary

ACOA always gives solution and solution gets better with time. It is more useful and efficient when search space is large, complex and poorly known or no mathematical analysis is available. The GA is well suited to and has been extensively applied to solve complex design optimization problems because it can handle both discrete and

continuous variables, and nonlinear objective functions without requiring gradient information.

Though ant colony algorithms can solve some optimization problems successfully, we cannot prove its convergence. It is prone to falling in the local optimal solution because the ACO updates the pheromone according to the current best path GA memory less, ACO memory intensive.

The ACO is inferior to GA method in the sense that this method approach takes some unnecessary steps, so that the algorithm does not return the best solution. Furthermore, a global attraction term had to be added to lead ant to reach the goal point. Eliminating this term may cause not only the ant wander around in the map, but also the ant may become stuck at a point.

ACO is commonly more applicable to problems where source and destination are predefined and specific, while PSO is commonly more applicable to problems where previous and next particle positions at each point are clear and uniquely defined. In ACO, the objective is generally searching for an optimal path in the construction graph while, in PSO, is generally finding the location of an optimal point in a Cartesian coordinate system.

Chapter 7 Conclusions and Further Research

This chapter provides a final conclusion to the thesis and identifies some areas for further research.

7.1 Conclusions

The metaphor of the foraging behaviour of real ants in a colony has been adopted to build the ACO algorithms and these approaches have been accepted as a generic framework for solving both discrete and continuous optimization problems in real applications. The algorithms have three basic phases: a construction phase, an update pheromone phase and Daemon Actions. ACO algorithms are different to one another in the way of the adapting these phases. This thesis has focused on the solution construction phase, in which a decision making process takes place to construct the solutions based on a certain utility.

We have implemented prospect theory into different variants of ACO algorithms. The tests were carried out on the travelling salesman problem and the water distribution system for discrete optimization problems; while mathematical unconstrained benchmark problems were used to test the continuous problems.

We have tested the proposed algorithm against AS and ACS algorithms for solving the TSP and the proposed algorithm was tested against the MMAS algorithm for solving

the water distribution system. For solving the continuous optimization problems, the proposed algorithm was tested against the ACOR algorithm. According to the comparison shown in Chapter 4 and Chapter 5, we can conclude that the proposed algorithm is highly competitive when compared with current standard ACO.

The implementation of PT in ACO was carried out by inserting it into the ACO algorithms in order to make a decision about where to move during the construction solution phase. In TSP, prospect theory was used to decide which city would be visited next during the construction solution phase. The probabilities were derived from the amount of pheromones of the edges; while the outcomes were obtained from the heuristic information. As a result, the closer the ants are in their vicinity the higher the outcome would be, and the more pheromones on the edge the higher the probability that the ant would choose that edge. The prospect of each decision point was calculated by multiplying the value function and weighting probability of this point. The outcomes were framed in the gain region, so that the reference point is always zero. In this scheme, the weighting probability function was taking the main role for determining the behaviour of ants. This means that ants were risk-seeking for a low probability of pheromone and risk averse for a high probability of pheromone when the next city was close to the current city. These behaviours forced ants to explore at the beginning of the iteration where the level of pheromones was low and decreased the exploration when the pheromones were high during the end of iteration.

In the water distribution system, the New York Water Supply Tunnel Problem and Hanoi Network Problem were used as study cases. The implementation of prospect theory in WDS was in the construction phase to choose which candidate solutions to update the pheromone. The solution was selected through its prospects which depend

on its objective functions and the violation of the constraints. This approach avoided the problem of using the penalty function which can over-penalize or under-penalize. The constraint violations were used to obtain the probabilities; the more violations, the lower the probability that outcomes would be used as a solution. The value function was derived from the objective function. The reference point was calculated by averaging the objective costs of all solutions obtained.

In choosing the candidate solution for WDS, if all solutions were unfeasible or feasible, then a solution was chosen probabilistically based on their prospect. However, if one or more solutions were feasible, the solution candidate had to be chosen from the feasible ones even the prospect of the unfeasible solution was better. This approach was to ensure that the final solution is always feasible if there is one.

In continuous unconstrained problems, probability was derived from ranking the solutions. The rank was arranged from best to worst and weight was given to this rank. This rank was used to construct the probability. The value function was derived from the objective function. The reference point was calculated from the average of the objective costs of the solutions. The prospect of each outcome would be used as a base for selection.

The novelty of the proposed algorithms is that for the first time human-like behaviour has been incorporated into the state transition strategy of ACO. This behaviour was formalized by Kahneman & Tversky (1979) into mathematical equations represented by PT. PT was used in the proposed algorithms to guide the ants to make decisions. The proposed algorithms tended to improve the search ability and, consequently, improved the convergence to the optimal solution. To investigate the proposed algorithms, the following analyses were undertaken:

The proposed framework used prospect theory (PT), which is a leading behavioural model of decision-making processes under conditions of risk, and has been used successfully in economics and management (Trepel, Fox, & Poldrack, 2005). In the ant colony system, for the first time, the present updated mechanism for this system inspired by the behaviour of PT has been used to improve the exploration of the region in order to prevent the algorithm from being trapped in a local minimum by taking the risk to explore the lower probability of pheromones.

A novel framework for the ant colony algorithms based on actual human-like decision-making behaviour under risk has been presented, which obtains better convergence efficiency than state-of-the-art algorithms when applied to solve different problems.

The proposed framework is flexible and it is believed it can be applied efficiently to different variants of ACO without major changes where decision-making under risk is dominant.

Through several different experiments, the proposed method has demonstrated high performance in situations of decision making under risk. The observed results of this work have shown that the proposed framework can find its major application in discrete and continuous optimization problems.

7.2 Future Research

While the ACO-PT algorithms perform well, there are still possibilities for improvement. In this research the parameters of prospect theory were kept constant. The parameters were obtained from the empirical study of PT (Kahneman & Tversky, 1979). These parameters should be investigated further, as to whether they can give the

best performance for a certain problem, so the parameters can be changed depending on the characteristics of the problem.

The weighting probability function was the most influential on the resulting behaviour, so the shape of this function should be investigated further to find the most appropriate one for a specific problem.

Implementing prospect theory in the ACO for tackling continuous constrained problems can be a challenge. Also, the use of prospect theory could be investigated in the real time application of ACO in the mobile robot path planning where the robot interacts with the environment.

When algorithms inspired by the behaviour of ants to solve the optimization problems were first introduced in the early 90s it seemed questionable. Since many results from ACO research have shown successful applications the perspective has changed and now they are considered one of the most promising approaches (Dorigo, Birattari & Stützle, 2006). The use of human behaviour in ACO was also questionable, so a better understanding of the theoretical properties is certainly another research direction that will be pursued in the future in order to obtain more fundamental properties of ACO-PT algorithms.

Appendices A

First part of the test function for comparing ACOR (Socha & Dorigo, 2008) and ACOR-PT

Function	Formula
Plane (PL) $\vec{x} \in [0.5,1.5]^n, n = 10$	$f_{PL}(\vec{x}) = x_1$
Diagonal plane (DP) $\vec{x} \in [0.5,1.5]^n, n = 10$	$f_{DP}(\vec{x}) = \frac{1}{n}\sum_{i=1}^{n} x_i$
Sphere (SP) $\vec{x} \in [-3,7]^n, n = 10$	$f_{SP}(\vec{x}) = \sum_{i=1}^{n} x_i^2$
Ellipsoid (EL) $\vec{x} \in [-3,7]^n, n = 10$	$f_{EL}(\vec{x}) = \sum_{i=1}^{n} \left(100^{\frac{i-1}{n-1}} x_i\right)^2$
Branin RCOS (RC) $\vec{x} \in [-5,15]^n, n = 2$	$f_{RC}(\vec{x}) = (x_2 - \frac{5}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6)^2 + 10\left(1 - \frac{1}{8\pi}\right)cosx_1 + 10$
Golstein and Price (GP) $\vec{x} \in [-2,2]^n, n = 2$	$f_{GP}(\vec{x}) = (1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 13x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2))$
Rosenbrock(R_n) $\vec{x} \in [-5,10]^n, n = 2,5$	$f_{R_n}(\vec{x}) = \sum_{i=1}^{n-1} 100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2$
Zakharov (Z_n) $\vec{x} \in [-5,10]^n, n = 2,5$	$f_{Z_n}(\vec{x}) = \left(\sum_{j=1}^{n-1} x_j^2\right) + \left(\sum_{j=1}^{n} \frac{jx_j}{2}\right)^2 + \left(\sum_{j=1}^{n} \frac{jx_j}{2}\right)^4$
Griewangk (GR_n) $\vec{x} \in [-5.12,5.12]^n, n = 10$	$f_{GR_n}(\vec{x}) = \left(\frac{1}{10} + \left(\sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1\right)\right)^{-1}$

Second part of the test function for comparing ACOR (Socha & Dorigo, 2008) and ACOR-PT

Function	Formula
Hartman $(H_{3,4})$ $\vec{x} \in [0,1]^n, n = 4$	$f_{H34}(\vec{x}) = -\sum_{i=1}^{4} c_i e^{-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2}$ $a_{ij} = \begin{Bmatrix} 3.0 & 10.0 & 30.0 \\ 0.1 & 10.0 & 35.0 \\ 3.0 & 10.0 & 30.0 \\ 0.1 & 10.0 & 35.0 \end{Bmatrix}$, $c_i = \begin{Bmatrix} 1.0 \\ 1.2 \\ 3.0 \\ 3.2 \end{Bmatrix}$, $p_{ij} = \begin{Bmatrix} 0.3689 & 0.1170 & 0.2673 \\ 0.4699 & 0.4387 & 0.7470 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.0381 & 0.5743 & 0.8828 \end{Bmatrix}$
Hartman $(H_{6,4})$ $\vec{x} \in [0,1]^n, n = 4$	$f_{H64}(\vec{x}) = -\sum_{i=1}^{4} c_i e^{-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2}$ $a_{ij} = \begin{Bmatrix} 10.0 & 3.00 & 17.0 & 3.50 & 1.50 & 8.00 \\ 0.05 & 10.0 & 17.0 & 0.10 & 8.00 & 14.0 \\ 3.00 & 3.50 & 1.70 & 10.0 & 17.0 & 8.00 \\ 17.0 & 8.00 & 0.05 & 10.0 & 0.10 & 14.0 \end{Bmatrix}$, $c_i = \begin{Bmatrix} 1.0 \\ 1.2 \\ 3.0 \\ 3.2 \end{Bmatrix}$, $p_{ij} = \begin{Bmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{Bmatrix}$
Shekel $(S_{4,k}\ k = 5,7,10)$ $\vec{x} \in [0,10]^n, n = 4$	$f_{S_{4k}}(\vec{x}) = -\sum_{i=1}^{k} ((\vec{x} - \overrightarrow{a_i})^T(\vec{x} - \overrightarrow{a_i}) + c_i)^{-1}$ $a_{ij} = \begin{Bmatrix} 4.0 & 4.0 & 4.0 & 4.0 \\ 1.0 & 1.0 & 1.0 & 1.0 \\ 8.0 & 8.0 & 8.0 & 8.0 \\ 6.0 & 6.0 & 6.0 & 6.0 \\ 3.0 & 7.0 & 3.0 & 7.0 \\ 2.0 & 9.0 & 2.0 & 9.0 \\ 5.0 & 3.0 & 5.0 & 3.0 \\ 8.0 & 1.0 & 8.0 & 1.0 \\ 6.0 & 2.0 & 6.0 & 2.0 \\ 7.0 & 3.6 & 7.0 & 3.6 \end{Bmatrix}$, $c_i = \begin{Bmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \\ 0.7 \\ 0.5 \\ 0.5 \end{Bmatrix}$

Appendices B Source Code for AS and AS-PT

```
function ACO(inputfile)
%% Example: ACO('oliver30.tsp')
disp('AS is reading input nodes file...');
[Dimension,NodeCoord,NodeWeight,Name]=FileInput(inputfile);
disp([num2str(Dimension),' nodes in',Name,' has been read in']);
disp(['AS start at ',datestr(now)]);
%%%%%%%%%%%%% the key parameters of Ant System %%%%%%%%%
MaxITime=500;
AntNum=Dimension;
alpha=1;
beta=5;
rho=0.5;
%%%%%%%%%%%%% the key parameters of Ant System %%%%%%%%%
fprintf('Showing Iterative Best Solution:\n');
[GBTour,GBLength,Option,IBRecord] = ...
AS(NodeCoord,NodeWeight,AntNum,MaxITime,alpha,beta,rho);
disp(['AS stop at ',datestr(now)]);
disp('Drawing the iterative course''s curve');
figure(1);
subplot(2,1,1)
plot(1:length(IBRecord(1,:)),IBRecord(1,:));
xlabel('Iterative Time');
ylabel('Iterative Best Cost');
title(['Iterative Course: ','GMinL=',num2str(GBLength),',
FRIT=',num2str(Option.OptITime)]);
subplot(2,1,2)
plot(1:length(IBRecord(2,:)),IBRecord(2,:));
xlabel('Iterative Time');
ylabel('Average Node Branching');
figure(2);
DrawCity(NodeCoord,GBTour);
title([num2str(Dimension),' Nodes Tour Path of ',Name]);
figure(3);
subplot(2,1,1)
plot(1:length(IBRecord(end,:)),IBRecord(end-1,:));
xlabel('Iterative Time');
ylabel('Best Tour Length');
subplot(2,1,2)
plot(1:length(IBRecord(2,:)),IBRecord(end,:));
xlabel('Iterative Time');
ylabel('Tour Length Std');
%--------------------------------------------------------------------

function [Dimension,NodeCoord,NodeWeight,Name]=FileInput(infile)
if ischar(infile)
    fid=fopen(infile,'r');
else
    disp('input file no exist');
    return;
end
```

```
if fid<0
    disp('error while open file');
    return;
end
NodeWeight = [];
while feof(fid)==0
    temps=fgetl(fid);
    if strcmp(temps,'')
        continue;
    elseif strncmpi('NAME',temps,4)
        k=findstr(temps,':');
        Name=temps(k+1:length(temps));
    elseif strncmpi('DIMENSION',temps,9)
        k=findstr(temps,':');
        d=temps(k+1:length(temps));
        Dimension=str2double(d); %str2num
    elseif strncmpi('EDGE_WEIGHT_SECTION',temps,19)
        formatstr = [];
        for i=1:Dimension
            formatstr = [formatstr,'%g '];
        end
        NodeWeight=fscanf(fid,formatstr,[Dimension,Dimension]);
        NodeWeight=NodeWeight';
    elseif strncmpi('NODE_COORD_SECTION',temps,18) || ...
        strncmpi('DISPLAY_DATA_SECTION',temps,20)
        NodeCoord=fscanf(fid,'%g %g %g',[3 Dimension]);
        NodeCoord=NodeCoord';
    end
end
fclose(fid);
%---------------------------------------------------------------------

function plothandle=DrawCity(CityList,Tours)
xd=[];yd=[];
nc=length(Tours);
plothandle=plot(CityList(:,2:3),'.');
set(plothandle,'MarkerSize',16);
for i=1:nc
    xd(i)=CityList(Tours(i),2);
    yd(i)=CityList(Tours(i),3);
end
set(plothandle,'XData',xd,'YData',yd);
line(xd,yd);
%---------------------------------------------------------------------

function [GBTour,GBLength,Option,IBRecord]=AS(CityMatrix,...
WeightMatrix,AntNum,MaxITime,alpha,beta,rho)
%% (Ant System) date:070427
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reference
% Dorigo M, Maniezzo Vittorio, Colorni Alberto.
% The Ant System: Optimization by a colony of cooperating agents [J].
% IEEE Transactions on Systems, Man, and Cybernetics Part B,1996,26(1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global ASOption Problem AntSystem
ASOption = InitParameter(CityMatrix,AntNum,alpha,beta,rho,MaxITime);
Problem = InitProblem(CityMatrix,WeightMatrix);
AntSystem = InitAntSystem();
ITime = 0;
IBRecord = [];
```

```
if ASOption.DispInterval ~= 0
    close all
    set(gcf,'Doublebuffer','on');
    hline=plot(1,1,'-o');
end
while 1
    InitStartPoint();
    for step = 2:ASOption.n
        for ant = 1:ASOption.m
           %P = CaculateShiftProb(step,ant);     %AS alone
            P = CaculateShiftProspect(step,ant); %PT is used
            nextnode = Roulette(P,1);
            RefreshTabu(step,ant,nextnode);
        end
    end
    CloseTours();
    ITime = ITime + 1;
    CaculateToursLength();
    GlobleRefreshPheromone();
    ANB = CaculateANB();
    [GBTour,GBLength,IBRecord(:,ITime)] = GetResults(ITime,ANB);
    ShowIterativeCourse(GBTour,ITime,hline);
    if Terminate(ITime,ANB)
        break;
    end
end
Option = ASOption;
%% ---------------------------------------------------------------------

function ASOption = InitParameter(Nodes,AntNum, ...
alpha,beta,rho,MaxITime)
ASOption.n = length(Nodes(:,1));
ASOption.m = AntNum;
ASOption.alpha = alpha;
ASOption.beta = beta;
ASOption.rho = rho;
ASOption.MaxITime = MaxITime;
ASOption.OptITime = 1;
ASOption.Q = 100;
ASOption.C = 1;
ASOption.lambda = 0.15;
ASOption.ANBmin = 2;
ASOption.GBLength = inf;
ASOption.GBTour = zeros(length(Nodes(:,1))+1,1);
ASOption.DispInterval = 10;
rand('state',sum(100*clock));
%% ---------------------------------------------------------------------

function Problem = InitProblem(Nodes,WeightMatrix)
global ASOption
n = length(Nodes(:,1));
MatrixTau = (ones(n,n)-eye(n,n))* ASOption.C;
Distances = WeightMatrix;
SymmetryFlag = false;
if isempty(WeightMatrix)
    Distances = CalculateDistance(Nodes);
    SymmetryFlag = true;
end
Problem = struct('nodes',Nodes,'dis',Distances,'tau',MatrixTau, ...
'symmetry',SymmetryFlag);
```

```
%% ---------------------------------------------------------------

function AntSystem = InitAntSystem()
global ASOption
AntTours = zeros(ASOption.m,ASOption.n+1);
ToursLength = zeros(ASOption.m,1);
AntSystem = struct('tours',AntTours,'lengths',ToursLength);
%% ---------------------------------------------------------------

function InitStartPoint()
global AntSystem ASOption
AntSystem.tours = zeros(ASOption.m,ASOption.n+1);
rand('state',sum(100*clock));
AntSystem.tours(:,1) = randint(ASOption.m,1,[1,ASOption.n]);
AntSystem.lengths = zeros(ASOption.m,1);
%% ---------------------------------------------------------------

function Probs = CaculateShiftProb(step_i, ant_k)
global AntSystem ASOption Problem
CurrentNode = AntSystem.tours(ant_k, step_i-1);
VisitedNodes = AntSystem.tours(ant_k, 1:step_i-1);
tau_i = Problem.tau(CurrentNode,:);
tau_i(1,VisitedNodes) = 0;
dis_i = Problem.dis(CurrentNode,:);
dis_i(1,CurrentNode) = 1;
Probs = (tau_i.^ASOption.alpha).*((1./dis_i).^ASOption.beta);
if sum(Probs) ~= 0
    Probs = Probs/sum(Probs);
else
    NoVisitedNodes = setdiff(1:ASOption.n,VisitedNodes);
    Probs(1,NoVisitedNodes) = 1/length(NoVisitedNodes);
end
%% ---------------------------------------------------------------

%% Function for calculating prospect – using PT
function Probs = CaculateShiftProspect(step_i, ant_k)
global AntSystem ASOption Problem
CurrentNode = AntSystem.tours(ant_k, step_i-1);
VisitedNodes = AntSystem.tours(ant_k, 1:step_i-1);
tau_i = Problem.tau(CurrentNode,:);
tau_i(1,VisitedNodes) = 0;
dis_i = Problem.dis(CurrentNode,:);
dis_i(1,CurrentNode) = 1;
Outcomes = tau_i.^ASOption.alpha;
probs = (1./dis_i).^ASOption.betha;
if sum(probs) ~= 0
    probs = probs/sum(probs);
else
    NoVisitedNodes = setdiff(1:ASOption.n,VisitedNodes);
    probs(1,NoVisitedNodes) = 1/length(NoVisitedNodes);
end
gamma = 0.68;
den= probs.^gamma + (1 - probs).^gamma;
W_probs = (probs.^gamma)./(den.^(1/gamma));
Probs = Outcomes.*W_probs;
Probs = Probs /sum(Probs );
%---------------------------------------------------------------

function Select = Roulette(P,num)
m = length(P);
```

```
flag = (1-sum(P)<=1e-5);
Select = zeros(1,num);
rand('state',sum(100*clock));
r = rand(1,num);
for i=1:num
    sumP = 0;
    j = ceil(m*rand);
    while (sumP<r(i)) && flag
        sumP = sumP + P(mod(j-1,m)+1);
        j = j+1;
    end
    Select(i) = mod(j-2,m)+1;
end
%% ------------------------------------------------------------------

function RefreshTabu(step_i,ant_k,nextnode)
global AntSystem
AntSystem.tours(ant_k,step_i) = nextnode;
%% ------------------------------------------------------------------

function CloseTours()
global AntSystem ASOption
AntSystem.tours(:,ASOption.n+1) = AntSystem.tours(:,1);
%% ------------------------------------------------------------------

function CaculateToursLength()
global AntSystem ASOption Problem
Lengths = zeros(ASOption.m,1);
for k=1:ASOption.m
    for i=1:ASOption.n
        Lengths(k)=Lengths(k)+...
        Problem.dis(AntSystem.tours(k,i),AntSystem.tours(k,i+1));
    end
end
AntSystem.lengths = Lengths;
%% ------------------------------------------------------------------

function [GBTour,GBLength,Record] = GetResults(ITime,ANB)
global AntSystem ASOption
[IBLength,AntIndex] = min(AntSystem.lengths);
IBTour = AntSystem.tours(AntIndex,:);
if IBLength<=ASOption.GBLength
    ASOption.GBLength = IBLength;
    ASOption.GBTour = IBTour;
    ASOption.OptITime = ITime;
end
StdTour = std(AntSystem.lengths);
GBTour = ASOption.GBTour';
GBLength = ASOption.GBLength;
Record = [IBLength,ANB,IBTour,GBLength,StdTour]'; %%%%%
%% ------------------------------------------------------------------

function GlobleRefreshPheromone()
global AntSystem ASOption Problem
AT = AntSystem.tours;
TL = AntSystem.lengths;
sumdtau=zeros(ASOption.n,ASOption.n);
for k=1:ASOption.m
    for i=1:ASOption.n
```

```
sumdtau(AT(k,i),AT(k,i+1))=sumdtau(AT(k,i),AT(k,i+1))+ASOption.Q/TL(k);
    if Problem.symmetry
       sumdtau(AT(k,i+1),AT(k,i))=sumdtau(AT(k,i),AT(k,i+1));
    end
    end
end
Problem.tau=Problem.tau*(1-ASOption.rho)+sumdtau;
%% -------------------------------------------------------------------

function flag = Terminate(ITime,ANB)
global ASOption
flag = false;
if  ITime>=ASOption.MaxITime || ANB<=ASOption.ANBmin
    flag = true;
end
%% -------------------------------------------------------------------

function ANB = CaculateANB()
global ASOption Problem
mintau = min(Problem.tau+ASOption.C*eye(ASOption.n,ASOption.n));
sigma = max(Problem.tau) - mintau;
dis = Problem.tau - repmat(sigma*ASOption.lambda+mintau,ASOption.n,1);
NB = sum(dis>=0,1);
ANB = sum(NB)/ASOption.n;
%% -------------------------------------------------------------------

function Distances = CalculateDistance(Nodes)
global ASOption
Nodes(:,1)=[];
Distances=zeros(ASOption.n,ASOption.n);
for i=2:ASOption.n
    for j=1:i
        if(i==j)
            continue;
        else
            dij=Nodes(i,:)-Nodes(j,:);
            Distances(i,j)=sqrt(dij(1)^2+dij(2)^2);
            Distances(j,i)=Distances(i,j);
        end
    end
end
%% -------------------------------------------------------------------

function ShowIterativeCourse(IBTour,ITime,hmovie)
global Problem ASOption
num = length(IBTour);
if mod(ITime,ASOption.DispInterval)==0
    title(get(hmovie,'Parent'),['ITime = ',num2str(ITime)]);
    NodeCoord = Problem.nodes;
    xd=[];yd=[];
    for i=1:num
        xd(i)=NodeCoord(IBTour(i),2);
        yd(i)=NodeCoord(IBTour(i),3);
    end
    set(hmovie,'XData',xd,'YData',yd);
    pause(0.01);
end
```

Appendices C Source Code for ACS and ACS-PT

```
function ACO(inputfile)
%% Example: ACO('oliver30.tsp')
disp('ACS is reading input nodes file...');
[Dimension,NodeCoord,NodeWeight,Name]=FileInput(inputfile);
disp([num2str(Dimension),' nodes in',Name,' has been read in']);
disp(['ACS start at ',datestr(now)]);
%%%%%%%%%%%%% the key parameters of Ant Colony System %%%%%%%%%
MaxITime=1000;
AntNum= Dimension;
% The key parameter of ACS - Other parameters in InitParameter.m
alpha=1;
beta=5;
rho=0.5;
%%%%%%%%%%%%% ------------------------------------------------------
fprintf('Showing Iterative Best Solution:\n');
[GBTour,GBLength,Option,IBRecord] = ACS(NodeCoord,NodeWeight, …
AntNum,MaxITime,beta,rho);
disp(['ACS stop at ',datestr(now)]);
disp('Drawing the iterative course''s curve');
figure(1);
subplot(2,1,1)
plot(1:length(IBRecord(1,:)),IBRecord(1,:));
xlabel('Iterative Time');
ylabel('Iterative Global Best');
title(['Iterative Course: ','GMinL=',num2str(GBLength),',
FRIT=',num2str(Option.OptITime)]);
subplot(2,1,2)
plot(1:length(IBRecord(2,:)),IBRecord(2,:));
xlabel('Iterative Time');
ylabel('Average Node Branching');
%%%%%% ------------------------------------------------------------
figure(2);
if isempty(NodeCoord)
else
DrawCity(NodeCoord,GBTour);
title([num2str(Dimension),' Nodes Tour Path of ',Name]);
end
%%%%%% ------------------------------------------------------------
figure(3)
subplot(2,1,1)
plot(1:length(IBRecord(end-1,:)),IBRecord(end-1,:));
xlabel('Iterative Time');
ylabel('Iterative Best Cost');
subplot(2,1,2)
plot(1:length(IBRecord(end,:)),IBRecord(end,:));
xlabel('Iterative Time');
ylabel('Standard deviation');
%%%%%% ------------------------------------------------------------
```

```
function [GBTour,GBLength,Option,IBRecord]= ACS(CityMatrix,...
    WeightMatrix, AntNum,MaxITime,beta,rho)
%% (Ant Colony System) date:140092012
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reference
% Dorigo M, Gambardella L.M.
% Ant colonies for the traveling salesman problem
% BioSystems, Vol. 43, No.2, 1997,73-81
global ACSOption Problem ACS
ASOption = InitParameter(CityMatrix, ...
  WeightMatrix,AntNum,beta,rho,MaxITime);
Problem = InitProblem(CityMatrix,WeightMatrix);
ACS = InitACS();
ITime = 0;
IBRecord = [];
if ASOption.DispInterval ~= 0
    close all
    set(gcf,'Doublebuffer','on');
    hline=plot(1,1,'-o');
end
while 1
    InitStartPoint();
    for step = 2:ACSOption.n
      for ant = 1:ACSOption.m
        P = CaculateShiftProb(step,ant); %a state transition rule
      % P = CaculateShiftProspect(step,ant);%Using PT
        if rand <= ACSOption.q0         % ACS state transition rules
           [maxP nextnode] = max(P);
         else
           nextnode = Roulette(P,1);
         end
         LocalUpdatePheromone(step, ant, nextnode); %local updating
         RefreshTabu(step,ant,nextnode);
       end
    end   %End of step
    CloseTours();
    CaculateToursLength();
    ANB = CaculateANB();
    ITime = ITime + 1;
    [GBTour,GBLength,IBRecord(:,ITime)] = GetResults(ITime,ANB);
    GlobleRefreshPheromone();
    ShowIterativeCourse(GBTour,ITime,hline);
    if Terminate(ITime,ANB
      break;
    end
end
Option = ASOption;
%% ------------------------------------------------------------------

function Probs = CaculateShiftProb(step_i, ant_k)
global ACS ACSOption Problem
CurrentNode = ACS.tours(ant_k, step_i-1);
VisitedNodes = ACS.tours(ant_k, 1:step_i-1);
tau_i = Problem.tau(CurrentNode,:);
tau_i(1,VisitedNodes) = 0;
dis_i = Problem.dis(CurrentNode,:);
dis_i(1,CurrentNode) = 1;
Probs = tau_i.*((1./dis_i).^ASOption.beta);
if sum(Probs) ~= 0
    Probs = Probs/sum(Probs);
else
```

```
    NoVisitedNodes = setdiff(1:ASOption.n,VisitedNodes);
    Probs(1,NoVisitedNodes) = 1/length(NoVisitedNodes);
End
%% ------------------------------------------------------------------

%% function for calculating prospects
function Probs = CaculateShiftProspect(step_i, ant_k)
global ACS ACSOption Problem
CurrentNode = ACS.tours(ant_k, step_i-1);
VisitedNodes = ACS.tours(ant_k, 1:step_i-1);
tau_i = Problem.tau(CurrentNode,:);
tau_i(1,VisitedNodes) = 0;
dis_i = Problem.dis(CurrentNode,:);
dis_i(1,CurrentNode) = 1;
Outcomes = tau_i.^ACSOption.alpha;
probs = (1./dis_i).^ACSOption.beta;
if sum(probs) ~= 0
    probs = probs/sum(probs);
else
    NoVisitedNodes = setdiff(1:ACSOption.n,VisitedNodes);
    probs(1,NoVisitedNodes) = 1/length(NoVisitedNodes);
end
gamma = 0.68;
den= probs.^gamma + (1 - probs).^gamma;
W_probs = (probs.^gamma)./(den.^(1/gamma));
Probs = (Outcomes.^0.88).*W_probs;
Probs = Probs /sum(Probs );
%% ------------------------------------------------------------------

function ANB = CaculateANB()
global ACSOption Problem
mintau = min(Problem.tau+ACSOption.C*eye(ACSOption.n,ACSOption.n));
sigma = max(Problem.tau) - mintau;
dis = Problem.tau - repmat(sigma*ACSOption.lambda ...
        +mintau,ACSOption.n,1);
NB = sum(dis>=0,1);
ANB = sum(NB)/ASOption.n;
%% ------------------------------------------------------------------

function CaculateToursLength()
global ACS ACSOption Problem
Lengths = zeros(ACSOption.m,1);
for k=1:ACSOption.m
    for i=1:ACSOption.n
     Lengths(k)=Lengths(k)+Problem.dis(ACS.tours(k,i) ...
     ,ACS.tours(k,i+1));
    end
end
ACS.lengths = Lengths;
%% ------------------------------------------------------------------

function Distances = CalculateDistance(Nodes)
global ACSOption
Nodes(:,1)=[];
Distances=zeros(ACSOption.n,ACSOption.n);
for i=2:ACSOption.n
    for j=1:i
        if(i==j)
            continue;
        else
            dij=Nodes(i,:)-Nodes(j,:);
```

```
            Distances(i,j)= sqrt(dij(1)^2+dij(2)^2);
            Distances(j,i)=Distances(i,j);
        end
    end
end
%% ---------------------------------------------------------------


function CloseTours()
global ACS ACSOption
ACS.tours(:,ACSOption.n+1) = ACS.tours(:,1);
%% ---------------------------------------------------------------

function plothandle=DrawCity(CityList,Tours)
xd=[];yd=[];
nc=length(Tours);
plothandle=plot(CityList(:,2:3),'.');
set(plothandle,'MarkerSize',16);
for i=1:nc
    xd(i)=CityList(Tours(i),2);
    yd(i)=CityList(Tours(i),3);
end
set(plothandle,'XData',xd,'YData',yd);
line(xd,yd);
%% ---------------------------------------------------------------

function [Dimension,NodeCoord,NodeWeight,Name]=FileInput(infile)
if ischar(infile)
    fid=fopen(infile,'r');
else
    disp('input file no exist');
    return;
end
if fid<0
    disp('error while open file');
    return;
end
NodeCoord =[];
NodeWeight = [];
while feof(fid)==0
    temps=fgetl(fid);
    if strcmp(temps,'')
        continue;
    elseif strncmpi('NAME',temps,4)
        k=findstr(temps,':');
        Name=temps(k+1:length(temps));
    elseif strncmpi('DIMENSION',temps,9)
        k=findstr(temps,':');
        d=temps(k+1:length(temps));
        Dimension=str2double(d); %str2num
    elseif strncmpi('EDGE_WEIGHT_SECTION',temps,19)
        formatstr = [];
        for i=1:Dimension
            formatstr = [formatstr,'%g '];
        end
        NodeWeight=fscanf(fid,formatstr,[Dimension,Dimension]);
        NodeWeight=NodeWeight';
    elseif strncmpi('NODE_COORD_SECTION',temps,18) ||
strncmpi('DISPLAY_DATA_SECTION',temps,20)
        NodeCoord=fscanf(fid,'%g %g %g',[3 Dimension]);
        NodeCoord=NodeCoord';
```

```
    end
end
fclose(fid);
%% ---------------------------------------------------------------

function [GBTour,GBLength,Record] = GetResults(ITime,ANB)
global ACS ACSOption
[IBLength,AntIndex] = min(ACS.lengths);
IBTour = ACS.tours(AntIndex,:);
StdTour = std(ACS.lengths);
if IBLength<=ASOption.GBLength
    ACSOption.GBLength = IBLength;
    ACSOption.GBTour = IBTour;
    ACSOption.OptITime = ITime;
end
GBTour = ACSOption.GBTour';
GBLength = ACSOption.GBLength;
Record = [GBLength,ANB,IBTour,IBLength,StdTour]';
%% ---------------------------------------------------------------

function GlobleRefreshPheromone()
global ACSOption Problem
for i=1:ACSOption.n
       Problem.tau(ACSOption.GBTour(i),ACSOption.GBTour(i+1))= ...
       (1-ACSOption.alpha) *Problem.tau(ACSOption.GBTour(i), ...
       ACSOption.GBTour(i+1))+ ACSOption.alpha/ACSOption.GBLength;
if Problem.symmetry
  Problem.tau(ACSOption.GBTour(i+1),ACSOption.GBTour(i))= ...
  Problem.tau(ACSOption.GBTour(i),ACSOption.GBTour(i+1));
end
end
%% ---------------------------------------------------------------

function ACS = InitACS()
global ACSOption
AntTours = zeros(ACSOption.m,ACSOption.n+1);
ToursLength = zeros(ACSOption.m,1);
ACS = struct('tours',AntTours,'lengths',ToursLength);
%% ---------------------------------------------------------------

function ACSOption = InitParameter(CityMatrix,
WeightMatrix,AntNum,beta,rho,MaxITime)
if isempty(CityMatrix)
    Nodes = WeightMatrix;
else
    Nodes = CityMatrix;
end
ACSOption.n = length(Nodes(:,1));
ACSOption.m = AntNum;
ACSOption.alpha = alpha;
ACSOption.beta = beta;
ACSOption.rho = rho;
ACSOption.q0 = 0.9;
ACSOption.tau0 = 7e-005;
ACSOption.C = ASOption.tau0 ;
ACSOption.MaxITime = MaxITime;
ACSOption.OptITime = 1;
ACSOption.Lbs = Inf;
ACSOption.lambda = 0.15;
ACSOption.ANBmin = 2;
ACSOption.GBLength = inf;
```

```
ACSOption.GBTour = zeros(length(Nodes(:,1))+1,1);
ACSOption.DispInterval = 10;
rand('state',sum(100*clock));
%% ---------------------------------------------------------------

function Problem = InitProblem(CityMatrix,WeightMatrix)
global ACSOption
if isempty(CityMatrix)
    Nodes = WeightMatrix;
    n = length(Nodes(:,1));
    Distances = WeightMatrix;
    SymmetryFlag = false;
else
   Nodes =  CityMatrix;
   n = length(Nodes(:,1));
   Distances = CalculateDistance(Nodes);
   SymmetryFlag = true;
end
MatrixTau = (ones(n,n)-eye(n,n))*ACSOption.C;
Problem = struct('nodes',Nodes,'dis',Distances, ...
        'tau',MatrixTau,'symmetry',SymmetryFlag);
%% ---------------------------------------------------------------

function InitStartPoint()
global ACS ACSOption
ACS.tours = zeros(ACSOption.m,ACSOption.n+1);
rand('state',sum(100*clock));
ACS.tours(:,1) = randint(ACSOption.m,1,[1,ACSOption.n]);
ACS.lengths = zeros(ACSOption.m,1);
%% ---------------------------------------------------------------

function LocalUpdatePheromone(step_i, ant_k, nextnode)
global ACS ACSOption Problem
CurrentNode = ACS.tours(ant_k, step_i-1);
Problem.tau(CurrentNode,nextnode) = (1-ACSOption.rho) ...
   *Problem.tau(CurrentNode,nextnode)+ ACSOption.rho ...
   * ACSOption.tau0;
if Problem.symmetry

Problem.tau(nextnode,CurrentNode)=Problem.tau(CurrentNode,nextnode);
end
%% ---------------------------------------------------------------

function RefreshTabu(step_i,ant_k,nextnode)
global ACS
ACS.tours(ant_k,step_i) = nextnode;
%% ---------------------------------------------------------------

function Select = Roulette(P,num)
m = length(P);
flag = (1-sum(P)<=1e-5);
Select = zeros(1,num);
rand('state',sum(100*clock));
r = rand(1,num);
for i=1:num
    sumP = 0;
    j = ceil(m*rand);
    while (sumP<r(i)) && flag
        sumP = sumP + P(mod(j-1,m)+1);
        j = j+1;
    end
```

```
    Select(i) = mod(j-2,m)+1;
end
%% ---------------------------------------------------------------

function ShowIterativeCourse(IBTour,ITime,hmovie)
global Problem ASOption
num = length(IBTour);
if mod(ITime,ASOption.DispInterval)==0
    title(get(hmovie,'Parent'),['ITime = ',num2str(ITime)]);
    NodeCoord = Problem.nodes;
    xd=[];yd=[];
    for i=1:num
        xd(i)=NodeCoord(IBTour(i),2);
        yd(i)=NodeCoord(IBTour(i),3);
    end
    set(hmovie,'XData',xd,'YData',yd);
    pause(0.01);
end
%% ---------------------------------------------------------------

function flag = Terminate(ITime,ANB)
global ACSOption
flag = false;
if ITime>=ACSOption.MaxITime || ANB<=ASOption.ANBmin
    flag = true;
end
%% ---------------------------------------------------------------
```

Appendices D Source Code for WDS-PT

```
%New York Water Tunnel Problem
function [best_iter fo_best Solution]=wds_aco_nytp()
clc
clear all
close all

loadlibrary('PTD', 'PTD2_2.h');
%Dummy variables
Msg = '123'; dia = 0.0; rough = 0.0;
%Open PTD toolkit

Err = calllib('PTD', 'PTopenD', 'NewYork.inp', 'NewYork.dat');
if(Err < 0)
    disp('unable to Open PTD toolkit');
    [Err, Msg] = calllib('PTD', 'PTgeterrmessage', Msg, 80);
    disp(Msg);
    return;
end

%Load data
NetworkData();
load('VariableBounds.mat');
%define number of variables
NNP = length(LBnewPipes);          %Number of new pipes
NOP = length(LBoldPipes);          %Number of old pipes

%define anonymous objective function and number of variables
objfun = @(x)Simulator(x, NNP, NOP);

pipe_length = [11600 19800 7300 8300 8600 19100 9600 12500 9600 11200
14500 12200 24100 21100 15500 26400 31200 24000 14400 38400 26400];
pipe_cost = [21.0 93.5 134.0 176.0 221.0 267.0 316.0 365.0 417.0 469.0
522.0 577.0 632.0 689.0 746.0 804.0];

node_size = length(pipe_length);
pipe_type = length(pipe_cost);

fglobal = 1e8; %
global_solution = 16*ones(1,21);

% Ant Colony Optimization
% Initialize parameter
nAnts =84;
alpha = 1;
beta = 0.45;
rho = 0.98;
Q =2.9e8;

eta=[]; %heuristic information
```

```
for i=1:node_size
    for j=1:pipe_type
      eta(j,i) = 1/(pipe_length(i)*pipe_cost(j));
    end
end

% MMAS intialization
pbest = 0.05;
NOavg = pipe_type;
n = node_size; %the number of the right decision

%algorithm parameter
max_iter = 1000;

%Initialize pheromone
t0 = 140;
tmax = t0;
tmin = 0;
ph_l=t0*ones(pipe_type, node_size);
stag  = 0;
global_best = [];

for iter=1:max_iter
x = []; y = []; P = []; fcost = [];
% construction phase

for k=1:nAnts
  for i= 1:node_size  %calculate probability for each of node
     prob =[];
     for j=1:pipe_type
        prob(j) = ph_l(j,i)^alpha * eta(j,i)^beta ; %calculate the
probabiliy
     end
     P(:,i) = prob/sum(prob); %normalised probability
      if rand > 0.9 %eplison
         [maxP indP] = max(P(:,i));
         np_ind(k,i) = indP-1;
     else
         np_ind(k,i)= Roulette(P(:,i),1)-1
     end
  end

%evaluate the objective function
  [x(k), y(k)] = objfun(np_ind(k,:));
  z(k,:) = [k x(k) y(k)];

end

w=eval_rank(z(:,1)',z(:,2)',z(:,3)');

fbest(iter) = w(1,2);
elite_ind(iter) = w(1,1);
best_solution(iter,:) = np_ind(elite_ind(iter),:);

if w(1,3) == 0
    if fglobal > w(1,2);
       fglobal = w(1,2);
       global_solution = np_ind(elite_ind(iter),:);
    end
```

```
else
end

rank = z(2:end,:);
fcost = 1./rank(:,2);
cost_reference = mean(fcost);
cost_frame = fcost - cost_reference;
valuecost = valuefunction(0.8, 0.8, 2.7, cost_frame');
prob = 1 - rank(:,3)/max(rank(:,3));
probweight = weight_prob(prob',0.6);
prospect = valuecost' .* probweight';

rank_prospect = [rank prospect];
rank_prospect = flipud(sortrows(rank_prospect,4));

%trial pheromone update phase
ph_l = ph_l * rho; %evaporate all pheromone for all nodes after all
ants completed the tour

dtaubest = Q/fbest(iter);
dtauglobal = Q/fglobal;

mdata = 3;

for m=1:mdata
 for i=1:node_size
   ph_l(np_ind(w(m,1),i)+1,i)= ph_l(np_ind(w(m,1),i)+1,i)+ (3 -
m)*Q/(2*w(m,2));
 end
end

ph_l(find(ph_l > tmax)) = tmax;
ph_l(find(ph_l < tmin)) = tmin;
if not(isempty(find(ph_l == Inf)))
    xxx = 1;
end

tmax = (1/(1-rho))*Q/fglobal;
tmin = tmax*(1 - pbest^(1/n))/((NOavg - 1) * pbest ^ (1/n));
if tmin > tmax
   tmin = tmax;
end

global_best(iter) = fglobal;
iter
%stopping criteria
if iter > 1
  dis = fbest(iter-1) - fbest(iter);
  if dis <=1e03
    stag = stag + 1;
  else
    stag = 0;
  end
  if stag > 50
   break
  end
end

end %end of iteration
```

```
[fbest_global best_iter] = min(fbest)
solution = best_solution(best_iter,:)
[a, b] = objfun(solution)
corverge_solution = best_solution( length(fbest),:)
figure(1)
plot(1:iter, fbest, 1:iter, fglobal);

title('Best fitness at the each iteration');
ylabel('fitness');
xlabel('iteration');

figure(2)
plot(1:iter, global_best);
text(iter/2,fglobal + 3e7, ['global best w/o PT =',
num2str(fglobal)]);
ylabel('fitness');
xlabel('iteration');

%Close EPANET toolkit
calllib('PTD', 'PTcloseD');

%Unload EPANET DLL.
%unloadlibrary('PTD');

%Hanoi Network Problem
function [best_iter fo_best Solution]=wds_aco_hnp()
clear all
close all

loadlibrary('PTD', 'PTD2_2.h');

%Dummy variables
Msg = '123'; dia = 0.0; rough = 0.0;

%Open PTD toolkit
Err = calllib('PTD', 'PTopenD', 'Hanoi.inp', 'Hanoi.dat');
if(Err < 0)
    disp('unable to Open PTD toolkit');
    [Err, Msg] = calllib('PTD', 'PTgeterrmessage', Msg, 80);
    disp(Msg);
    return;
end

%Load data
NetworkData();
load('VariableBounds.mat');

%define number of variables
NNP = length(LBnewPipes);          %Number of new pipes
NOP = length(LBoldPipes);          %Number of old pipes

LB = [LBnewPipes, LBoldPipes];
UB = [UBnewPipes, UBoldPipes];

%define anonymous objective function and number of variables
objfun = @(x)Simulator(x, NNP, NOP);
Nvars = NNP+NOP;
```

```
pipe_length = [100 1350 900 1150 1450 450 850 850 800 950 1200 350 800
500 550 2730 ...
               1750 800 400 2200 1500 500 2650 1230 1300 850 300 750
1500 2000 1600 150 860 950];
pipe_cost = [45.73 70.40 98.39 129.33 180.75 278.28];


node_size = 34;
pipe_type = 6;

fglobal = 10969798; %10.969.798
global_solution = 6*ones(1,34);

nAnts =83;
alpha = 1;
beta = 0.25;
rho = 0.98;
Q =1.1e07;
penalty = 1e7;


eta=[]; %heuristic information
for i=1:node_size
    for j=1:pipe_type
      eta(j,i) = 1/(pipe_length(i)*pipe_cost(j));
    end
end

% MMAS intialization
pbest = 0.9;
NOavg = pipe_type;
n = node_size; %the number of the right decision

%algorithm parameter
max_iter = 700;

%Initialize pheromone
t0 = 26;
tmax = t0;
tmin = 0;
ph_l=t0*ones(pipe_type, node_size);
stag  = 0;
global_best = [];

for iter=1:max_iter
% construction phase
obj = []; y = []; P = []; fcost = [];
np_ind=zeros(nAnts,node_size);
ant_fitness = zeros(nAnts,3);
for k=1:nAnts
   for i= 1:node_size  %calculate probability for each of node
     if rand > 0.9  %epsilon
      [maxP indP] = max(P(:,i));
      np_ind(k,i) = indP;
     else
      np_ind(k,i)= Roulette(P(:,i),1);
     end
   end
%evaluate the objective function
```

```
[obj(k) viol(k)] = objfun(np_ind(k,:));
ant_fitness(k,:) = [k obj(k) viol(k)];
end

%stochastic ranking
sr = ant_fitness;
for j=1:nAnts
for i=1:nAnts-1
    u = rand;
    if or(sr(i,3) == sr(i+1,3), u < 0.05)
        if sr(i,2) > sr(i+1,2)
            temp(1,:)=sr(i,:);
            sr(i,:)=sr(i+1,:);
            sr(i+1,:)=temp(1,:);
        end
    else
        if sr(i,3) > sr(i+1,3)
            temp(1,:)=sr(i,:);
            sr(i,:)=sr(i+1,:);
            sr(i+1,:)=temp(1,:);
        end
    end
end
end

w=sr(1,1);

elite_ind = ant_fitness(w,1);
fbest(iter)= ant_fitness(w,2);
w_viol = ant_fitness(w,3);

if  fbest(iter)< fglobal & w_viol == 0
     fglobal = fbest(iter)
     global_solution = np_ind(elite_ind,:);
end
%-------------------------------------------
%trial pheromone update phase
ph_l = ph_l * rho; %evaporate all pheromone for all nodes after all
ants completed the tour

dtauglobal = Q/fglobal;

for i=1:node_size
    ph_l(np_ind(elite_ind,i),i)= ph_l(np_ind(elite_ind,i),i)+
dtauglobal;
end

%Max-Min Update
ph_l(find(ph_l > tmax)) = tmax;
ph_l(find(ph_l < tmin)) = tmin;

tmax = (1/(1-rho))*dtauglobal;
tmin = tmax*(1 - pbest^(1/n))/((NOavg - 1) * pbest ^ (1/n));
if tmin > tmax
   tmin = tmax;
end

global_best(iter) = fglobal;
```

```
%stopping criteria
 if iter > 1
   dis = abs(fbest(iter-1) - fbest(iter));
   if dis <=1e05
     stag = stag + 1;
   else
     stag = 0;
   end
   if stag > 50
    break
   end
end

end %end of iteration

[a, b] = objfun(global_solution)
figure(1)
plot(1:iter, fbest, 1:iter, fglobal);

title('Best fitness at the each iteration');
ylabel('fitness');
xlabel('iteration');

figure(2)
plot(1:iter, global_best);
text(iter/2,fglobal + 3e7, ['global best without PT =',
num2str(fglobal)]);
ylabel('fitness');
xlabel('iteration');

%Close EPANET toolkit
calllib('PTD', 'PTcloseD');

Unload EPANET DLL.
unloadlibrary('PTD');
```

Appendices E Source Code for ACOR and ACOR-PT

```
function  result = acorPTcg();
%Archive table is initialized by uniform random
%
clear all
nVar = 10;
nSize = 50; %size
nAnts = 2;
fopt = 0; %Apriori optimal
q=0.1;

qk=q*nSize;
xi = 0.85;

maxiter = 25000;
errormin = 1e-04; %Stopping criteria
%Paramaeter range
Up = 0.5*ones(1,nVar); %range for dp function
Lo = 1.5*ones(1,nVar);

%Initialize archive table with uniform random and sort the result from
%the lowest objective function to largest one.
S = zeros(nSize,nVar+1,1);

Solution = zeros(nSize,nVar+2,1);

for k=1:nSize
    Srand = zeros(nVar);
    for j = 1:nAnts
      for i=1:nVar
        Srand(j,i) = (Up(i) - Lo(i))* rand(1) + Lo(i); %uniform
distribution
      end
    ffbest(j)=dp(Srand(j,:));  %dp test function
    end
    [fbest kbest] = min(ffbest);
    S(k,:)=[Srand(kbest,:) fbest];
end


%Rank the archive table from the best (the lowest)
S = sortrows(S,nVar+1);
%Select the best one as the best
%Calculate the weight,w
%the parameter q determine which solution will be chosen as a guide to
%the next solution, if q is small, we prefer the higher rank
%qk is the standard deviation
% mean = 1, the best on
w = zeros(1,nSize);

for i=1:nSize
```

```
    w(i) = pdf('Normal',i,1.0,qk);
end

Solution=S;
%end of archive table initialization
stag = 0;
% Iterative process
for iteration = 1: maxiter
%phase one is to choose the candidate base one probability
%the higher the weight the larger probable to be chosen
%value of function of each pheromone

p=w/sum(w);

ref_point = mean(Solution(:,nVar+1));
for i=1:nSize
 pw(i) = weight_prob(p(i),0.6);
 objv(i)= valuefunction(0.8,0.8, 2.25, ref_point-Solution(i, nVar+1));
 prospect(i) = pw(i)*objv(i);
end
[max_prospect ix_prospect]=max(prospect);

selection = ix_prospect;

%phase two, calculate Gi
%first calculate standard deviation
delta_sum =zeros(1,nVar);
for i=1:nVar
 for j=1:nSize
  delta_sum(i) = delta_sum(i) + abs(Solution(j,i)- ...
                 Solution(selection,i)); %selection
 end
  delta(i)=xi /(nSize - 1) * delta_sum(i);
 end
% xi has the same as pheromone evaporation rate. Higher xi, the lower
% convergence speed of algorithm
% do sampling from PDF continuous with mean chosen from phase one and
% standard deviation calculated above
% standard devation * randn(1,) + mean , randn = random normal
generator
Stemp = zeros(nAnts,nVar);

for k=1:nAnts
    for i=1:nVar
       Stemp(k,i) =  delta(i) * randn(1) + Solution(selection,i);
%selection
       if Stemp(k,i)> Up(i)
           Stemp(k,i) = Up(i);
       elseif Stemp(k,i) < Lo(i);
           Stemp(k,i) = Lo(i);
       end
    end
    ffeval(k)  =dp(Stemp(k,:));  %dp test function
end

Ssample = [Stemp ffeval']; %put weight zero

%insert this solution to archive, all solution from ants
Solution_temp = [Solution; Ssample];
```

```
%sort the solution
Solution_temp = sortrows(Solution_temp,nVar+1);
%remove the worst
Solution_temp(nSize+1:nSize+nAnts,:)=[];
Solution = Solution_temp;
best_par(iteration,:) = Solution(1,1:nVar);
best_obj(iteration) = Solution(1,nVar+1);
%check stopping criteria

if iteration > 1
dis = best_obj(iteration-1) - best_obj(iteration);
if dis <=1e-04
    stag = stag + 1;
else
    stag = 0;
end
end

ftest = Solution(1,nVar+1);
if abs(ftest - fopt) < errormin || stag >=5000
    break
end
end
plot(1:iteration,best_obj);
clc
title (['ACOR6   ','best obj = ', num2str(best_obj(iteration))]);
disp('number of function eveluation')
result = nAnts*iteration;

%%-----------------------------------------------------------------

function value = valuefunction(alpha, beta, lambda, xinput)
value =[];
n = length(xinput);
for i=1:n
    if xinput(1,i) >= 0
        value(1,i) = xinput(1,i) ^ alpha;
    else
        value(1,i) = -lambda * (-xinput(1,i))^ beta;
    end
end

function prob = weight_prob(x, gamma)
% weighted the probability
% gamma is weighted parameter
prob=[];
for i=1:length(x)
  if x(i) < 1
   prob(i) = (x(i)^(gamma))/((x(i)^(gamma) + (1-
x(i))^(gamma))^(1/gamma));
   %prob(i) = (x(i)^(1/gamma))/((x(i)^(1/gamma) + (1-
x(i))^(1/gamma))^(1/gamma));
  else
   prob(i) = 1.0;
  end
end

%%-----------------------------------------------------------------
%List of funtions
```

```
function y = dp(x)
% Diagonal plane
% n is the number of parameter
n = length(x);
s = 0;
for j = 1: n
    s = s + x(j);
end
y = 1/n * s;
%%---------------------------------------------------------------------

function y = branin(x)
% Branin function
% Matlab Code by A. Hedar (Sep. 29, 2005).
% The number of variables n = 2.
%
y = (x(2)-(5.1/(4*pi^2))*x(1)^2+5*x(1)/pi-6)^2+10*(1-
1/(8*pi))*cos(x(1))+10;
%%---------------------------------------------------------------------

function y = shekel45(x)
%
% Shekel function
% Matlab Code by A. Hedar (Nov. 23, 2005).
% The number of variables n = 4
% The parameter m should be adjusted m = 5,7,10.
% The default value of m = 10.
%
m = 5;
a = ones(10,4);
a(1,:) = 4.0*a(1,:);
a(2,:) = 1.0*a(2,:);
a(3,:) = 8.0*a(3,:);
a(4,:) = 6.0*a(4,:);
for j = 1:2;
   a(5,2*j-1) = 3.0; a(5,2*j) = 7.0;
   a(6,2*j-1) = 2.0; a(6,2*j) = 9.0;
   a(7,j)     = 5.0; a(7,j+2) = 3.0;
   a(8,2*j-1) = 8.0; a(8,2*j) = 1.0;
   a(9,2*j-1) = 6.0; a(9,2*j) = 2.0;
   a(10,2*j-1)= 7.0; a(10,2*j)= 3.6;
end
c(1) = 0.1; c(2) = 0.2; c(3) = 0.2; c(4) = 0.4; c(5) = 0.4;
c(6) = 0.6; c(7) = 0.3; c(8) = 0.7; c(9) = 0.5; c(10)= 0.5;
s = 0;
for j = 1:m;
   p = 0;
   for i = 1:4
      p = p+(x(i)-a(j,i))^2;
   end
   s = s+1/(p+c(j));
end
y = -s;
%%---------------------------------------------------------------------

function y = shekel47(x)
%
% Shekel function
% Matlab Code by A. Hedar (Nov. 23, 2005).
% The number of variables n = 4
% The parameter m should be adjusted m = 5,7,10.
```

```
% The default value of m = 10.
%
m = 7;
a = ones(10,4);
a(1,:) = 4.0*a(1,:);
a(2,:) = 1.0*a(2,:);
a(3,:) = 8.0*a(3,:);
a(4,:) = 6.0*a(4,:);
for j = 1:2;
   a(5,2*j-1) = 3.0; a(5,2*j) = 7.0;
   a(6,2*j-1) = 2.0; a(6,2*j) = 9.0;
   a(7,j)     = 5.0; a(7,j+2) = 3.0;
   a(8,2*j-1) = 8.0; a(8,2*j) = 1.0;
   a(9,2*j-1) = 6.0; a(9,2*j) = 2.0;
   a(10,2*j-1)= 7.0; a(10,2*j)= 3.6;
end
c(1) = 0.1; c(2) = 0.2; c(3) = 0.2; c(4) = 0.4; c(5) = 0.4;
c(6) = 0.6; c(7) = 0.3; c(8) = 0.7; c(9) = 0.5; c(10)= 0.5;
s = 0;
for j = 1:m;
   p = 0;
   for i = 1:4
      p = p+(x(i)-a(j,i))^2;
   end
   s = s+1/(p+c(j));
end
y = -s;
%%---------------------------------------------------------------------

function y = shekel410(x)
%
% Shekel function
% Matlab Code by A. Hedar (Nov. 23, 2005).
% The number of variables n = 4
% The parameter m should be adjusted m = 5,7,10.
% The default value of m = 10.
%
m = 10;
a = ones(10,4);
a(1,:) = 4.0*a(1,:);
a(2,:) = 1.0*a(2,:);
a(3,:) = 8.0*a(3,:);
a(4,:) = 6.0*a(4,:);
for j = 1:2;
   a(5,2*j-1) = 3.0; a(5,2*j) = 7.0;
   a(6,2*j-1) = 2.0; a(6,2*j) = 9.0;
   a(7,j)     = 5.0; a(7,j+2) = 3.0;
   a(8,2*j-1) = 8.0; a(8,2*j) = 1.0;
   a(9,2*j-1) = 6.0; a(9,2*j) = 2.0;
   a(10,2*j-1)= 7.0; a(10,2*j)= 3.6;
end
c(1) = 0.1; c(2) = 0.2; c(3) = 0.2; c(4) = 0.4; c(5) = 0.4;
c(6) = 0.6; c(7) = 0.3; c(8) = 0.7; c(9) = 0.5; c(10)= 0.5;
s = 0;
for j = 1:m;
   p = 0;
   for i = 1:4
      p = p+(x(i)-a(j,i))^2;
   end
   s = s+1/(p+c(j));
end
```

```
y = -s;
%%---------------------------------------------------------------------

function y = sphere(x)
%
% Sphere function
% Matlab Code by A. Hedar (Nov. 23, 2005).
% The number of variables n should be adjusted below.
% The default value of n = 30.
%
%n = 30;
n = length(x);
s = 0;
for j = 1:n
    s = s+x(j)^2;
end
y = s;
%%---------------------------------------------------------------------

function y = hart3(x)
%
% Hartmann function
% Matlab Code by A. Hedar (Sep. 29, 2005).
% The number of variables n = 3.
%
a(:,2)=10.0*ones(4,1);
for j=1:2;
   a(2*j-1,1)=3.0; a(2*j,1)=0.1;
   a(2*j-1,3)=30.0; a(2*j,3)=35.0;
end
c(1)=1.0;c(2)=1.2;c(3)=3.0;c(4)=3.2;
p(1,1)=0.36890;p(1,2)=0.11700;p(1,3)=0.26730;
p(2,1)=0.46990;p(2,2)=0.43870;p(2,3)=0.74700;
p(3,1)=0.10910;p(3,2)=0.87320;p(3,3)=0.55470;
p(4,1)=0.03815;p(4,2)=0.57430;p(4,3)=0.88280;
s = 0;
for i=1:4;
   sm=0;
   for j=1:3;
      sm=sm+a(i,j)*(x(j)-p(i,j))^2;
   end
   s=s+c(i)*exp(-sm);
end
y = -s;
%%---------------------------------------------------------------------

function y = hart6(x)
%
% Hartmann function
% Matlab Code by A. Hedar (Sep. 29, 2005).
% The number of variables n = 6.
%

a(1,1)=10.0;    a(1,2)=3.0;     a(1,3)=17.0;    a(1,4)=3.5;
a(1,5)=1.7;     a(1,6)=8.0;
a(2,1)=0.05;    a(2,2)=10.0;    a(2,3)=17.0;    a(2,4)=0.1;
a(2,5)=8.0;     a(2,6)=14.0;
a(3,1)=3.0;     a(3,2)=3.5;     a(3,3)=1.7;     a(3,4)=10.0;
a(3,5)=17.0;    a(3,6)=8.0;
```

```
a(4,1)=17.0;    a(4,2)=8.0;     a(4,3)=0.05;    a(4,4)=10.0;
a(4,5)=0.1;     a(4,6)=14.0;
c(1)=1.0;c(2)=1.2;c(3)=3.0;c(4)=3.2;
p(1,1)=0.1312;  p(1,2)=0.1696;  p(1,3)=0.5569;  p(1,4)=0.0124;
p(1,5)=0.8283;  p(1,6)=0.5886;
p(2,1)=0.2329;  p(2,2)=0.4135;  p(2,3)=0.8307;  p(2,4)=0.3736;
p(2,5)=0.1004;  p(2,6)=0.9991;
p(3,1)=0.2348;  p(3,2)=0.1451;  p(3,3)=0.3522;  p(3,4)=0.2883;
p(3,5)=0.3047;  p(3,6)=0.6650;
p(4,1)=0.4047;  p(4,2)=0.8828;  p(4,3)=0.8732;  p(4,4)=0.5743;
p(4,5)=0.1091;  p(4,6)=0.0381;
s = 0;
for i=1:4;
   sm=0;
   for j=1:6;
      sm=sm+a(i,j)*(x(j)-p(i,j))^2;
   end
   s=s+c(i)*exp(-sm);
end
y = -s;
%%---------------------------------------------------------------------

function y = gold(x)
%
% Goldstein and Price function
% Matlab Code by A. Hedar (Sep. 29, 2005).
% The number of variables n = 2.
%
a = 1+(x(1)+x(2)+1)^2*(19-14*x(1)+3*x(1)^2-
14*x(2)+6*x(1)*x(2)+3*x(2)^2);
b = 30+(2*x(1)-3*x(2))^2*(18-32*x(1)+12*x(1)^2+48*x(2)-
36*x(1)*x(2)+27*x(2)^2);
y = a*b;
%%---------------------------------------------------------------------

function y = rosen(x)
%
% Rosenbrock function
% Matlab Code by A. Hedar (Nov. 23, 2005).
% The number of variables n should be adjusted below.
% The default value of n = 2.
%
n=length(x);
sum = 0;
for j = 1:n-1;
    sum = sum+100*(x(j)^2 - x(j+1))^2+(x(j)-1)^2;
end
y = sum;
%%---------------------------------------------------------------------

function y = zakh(x)
%
% Zakharov function
% Matlab Code by A. Hedar (Nov. 23, 2005).
% The number of variables n should be adjusted below.
% The default value of n = 2.
%
n = 2;
s1 = 0;
s2 = 0;
for j = 1:n;
```

```
    s1 = s1+x(j)^2;
    s2 = s2+0.5*j*x(j);
end
y = s1+s2^2+s2^4;
%%---------------------------------------------------------------------


function y = griewank(x)
%
% Griewank function
% Matlab Code by A. Hedar (Sep. 29, 2005).
% The number of variables n should be adjusted below.
% The default value of n =2.
%
n = length(x);
fr = 4000;
s = 0;
p = 1;
for j = 1:n; s = s+x(j)^2; end
for j = 1:n; p = p*cos(x(j)/sqrt(j)); end
y = s/fr-p+1;
```

References

Allais, M. (1979). *The so-called Allais paradox and rational decisions under uncertainty*. Springer Netherlands.

Beni, G., & Wang, J. (1993). Swarm Intelligence in Cellular Robotic Systems. In *Robots and Biological Systems: Towards a New Bionics?* (pp. 703-712). Springer Berlin Heidelberg.

Bilchev, G., & Parmee, I. (1995). The ant colony metaphor for searching continuous design spaces. In *Evolutionary Computing* (pp. 25-39). Springer Berlin Heidelberg.

Blum, C. (2005). Ant colony optimization: Introduction and recent trends. *Physics of Life reviews*, *2*(4), 353-373.

Blum, C., & Dorigo, M. (2004). The hyper-cube framework for ant colony optimization. *IEEE Trans Syst Man Cybernet Part B, 34*(2), 1161-72.

Brand, M., Masuda, M., Wehner, N., & Yu, X.-H. (2010). Ant colony optimization algorithm for robot path planning. *Computer Design and Applications (ICCDA), 2010 International Conference on* (pp. 436-440). IEEE.

Bullnheimer, B., Hartl, R. F., & Strauß, C. (1999). A new rank based version of the Ant System: A computational study. *Central European Journal Operations Research Economics*, *7*(1), 25-38.

Camazine, S., Deneubourg, J. L., Franks, N. R., Sneyd, J., Theraulaz, G., & Bonabeau, E. (2003). *Self-organization in biological systems*. Princeton University Press.

Camerer, C. F. (1989). An experimental test of several generalized utility theories. *Journal of Risk and uncertainty*, *2*(1), 61-104.

Christodoulou, S. E., & Ellinas, G. (2010). Pipe routing through ant colony optimization. *Journal of Infrastructure Systems*, *16*(2), 149-159.

Coello Coello, C. A. (2002). Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer methods in applied mechanics and engineering*, *191*(11), 1245-1287.

Cunha, M. D., & Sousa, J. (1999). Water distribution network design optimization: simulated annealing approach. *Journal of Water Resources Planning and Management, 125*(4), 215-221.

Deb, K., & Padhye, N. (2010). Development of efficient particle swarm optimizers by using concepts from evolutionary algorithms. *Proceedings of the 12th annual conference on Genetic and evolutionary computation* (pp. 55-62). ACM.

Deneubourg, J.-L., Aron, S., Goss, S., & Pasteels, J. (1990). The self-organizing exploratory pattern of the argentine ant. *Journal of insect behavior, 3*(2), 159-168.

Dorigo, M., & Gambardella, L. M. (1997). Ant colonies for the travelling salesman problem. *BioSystems, 43*(2), 73--82.

Dorigo, M., & Stützle, T. (2003). The ant colony optimization metaheuristic: Algorithms, applications, and advances. *Handbook of metaheuristics*, 250-285.

Dorigo, M., & Stützle, T. (2004). *Ant Colony Optimization.* Cambridge, MA: MIT Press.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 26*(1), 29-41.

Dréo, J., & Siarry, P. (2004). Heterarchy, Continuous interacting ant colony algorithm based on dense. *Future Generation Computer Systems, 20*(5), 841-856.

Engelbrecht, A. P. (2005). *Fundamentals of computational swarm intelligence (Vol. 1).* Chichester: Wiley.

Fishburn, P. C. (1970). *Utility Theory For Decision Making.* John Wiley & Sons, Inc.

Fujiwara, O., & Khang, D. B. (1990). A two-phase decomposition method for optimal design of looped water distribution netwotks. *Water resources research, 26*(4), 539-549.

García, O. C., Triguero, F. H., & Stützle, T. (2002). A review on the ant colony optimization metaheuristic: basis, models and new trends. *Mathware & Soft Computing, 9*(3), 141-175.

Geetha, R., & Srikanth, G. U. (2012). Ant Colony Optimization in Diverse Engineering Applications: an Overview. *International Journal of Computer Application*, 19-25.

Gomes, H. P., Bezerra, S. D., De Carvalho, P. S., & Salvino, M. M. (2009). Optimal dimensioning model of water distribution systems. *Water Sa, 35*(4).

Guo, M., Liu, Y., & Malec, J. (2004). A new Q-learning algorithm based on the metropolis criterion. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics,, 34*(5), 2140-2143. doi:10.1109/TSMCB.2004.832154

Hansson, S. O. (1994). *Decision theory: A brief introduction.* Department of Philosophy and the History of Technology. Royal Institute of Technology. Stockholm.

Herrera, F., Lozano, M., & Molina, D. (2010). Test suite for the special issue of soft computing on scalability of evolutionary algorithms and other metaheuristics for large scale continuous optimization problems. Retrieved from http://sci2s.ugr.es/eamhco/

Hölldobler, B., & Wilson, E. O. (1990). *The ants.* Cambridge, MA: Harvard University Press.

Kahneman, D. (2003). Maps of bounded rationality: Psychology for behavioral economics. *The American economic review, 93*(5), 1449-1475.

Kahneman, D., & Tversky, A. (1979). Prospect theory: An analysis of decision under risk. *Econometrica: Journal of the Econometric Society*, 263-291.

Kahneman, D., & Tversky, A. (2000). *Choices, Values, and Frames.* New York: Cambridge University Press.

Kahneman, D., Slovic, P., & Tversky, A. (1982). *Judgment under uncertainty: Heuristics and biases.* Cambridge Univ Press.

Khichane, M., Albert, P., & Solnon, C. (2008). Integration of ACO in a constraint programming language. *Ant Colony Optimization and Swarm Intelligence*, 84-95.

Leguizamón, G., & Coello, C. (2010). An alternative ACOR algorithm for continuous optimization problems. *Proc of ANTS, 6234*, 48-59.

Li, T., Chen, W., Zheng, X., & Zhang, Z. (2009). An improvement of the ant colony optimization algorithm for solving travelling sales problem (TSP). *5th*

International Conference on Wireless Communications Networking and Mobile Computing.

Liao, T., de Oca, M., Aydn, D., Stützle, T., & Dorigo, M. (2011). An incremental ant colony algorithm with local search for continuous optimization. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2011).* New York.

Liu, L., Dai, Y., & Tao, C. (2011). State Transition Strategy Analysis of Ant Colony Algorithm. *Fourth International Joint Conference on Computational Sciences and Optimization (CSO),* (pp. 969-973). doi:10.1109/CSO.2011.245

López-Ibáñez, M., Prasad, T. D., & Paechter, B. (2008). Ant colony optimization for optimal control of pumps in water distribution networks. *Journal of water resources planning and management, 134*(4), 337-346.

Lozano, M., Herrera, F., & Molina, D. (2011). Special issue on scalability of evolutionary algorithms and other metaheuristics for large scale continuous optimization problems. *Soft Comput.*

Maier, H., Simpson, A., Zecchin, A., Foong, W., Phang, K., Seah, H., & Tan, C. (2003). Ant colony optimization for design of water distribution systems. *Journal of water resources planning and management, 129*(3), 200-209.

McDermott, R. (2001). *Risk-taking in international politics: Prospect theory in American foreign policy.* University of Michigan Press.

Monmarché, N., Venturini, G., & Slimane, M. (2000). On how Pachycondyla apicalis ants suggest a new search algorithm. *Future Generation Computer Systems, 16,* 937-946.

Nobahari, H., & Pourtakdoust, S. H. (2005, February). Optimization of fuzzy rule bases using continuous ant colony system. *In Proceedings of the First International Conference on Modeling, Simulation and Applied Optimization.*

Rini, D. P., & Shamsuddin, S. M. (2011). Particle swarm optimization: technique, system and challenges. *International Journal of Applied Information Systems, 1,* 33-45.

Runarsson, T. P., & Yao, X. (2000). Stochastic ranking for constrained evolutionary optimization. *Evolutionary Computation, IEEE Transactions on, 4*(3), 284-294.

Serugendo, G. D., Gleizes, M. P., & Karageorgos, A. (2011). *Self-organising Sofware: From Natural to Artificial Adaption.* Heidelberg: Springer.

Socha, K., & Dorigo, M. (2008). Ant colony optimization for continuous domains. *European Journal of Operational Research, 185*(3), 1155--1173.

Stützle, T., & Hoos, H. (2000). MAX--MIN ant system. *Future Generation Computer Systems, 16*, 889-914.

Trepel, C., Fox, C. R., & Poldrack, R. A. (2005). Prospect theory on the brain? Toward a cognitive neuroscience of decision under risk. *Cognitive Brain Research, 23*(1), 34-50.

Tversky, A., & Kahneman, D. (1992). Advances in prospect theory: Cumulative representation of uncertainty. *Journal of Risk and uncertainty, 5*(4), 297-323.

Wakker, P. P. (2010). *Prospect theory: For risk and ambiguity.* Cambridge: Cambridge University Press.

Zecchin, A. a., Simpson, A., Leonard, M., & Nixon, J. (2007). Ant colony optimization applied to water distribution system design: comparative study of five algorithms. *Journal of Water Resources Planning and Management, 133*(1), 87-92.

Zheng, S., Ge, M., Li, C., Wang, C., & Xue, A. (2010 July). New transition probability for Ant Colony Optimization: global random-proportional rule. In *Proceeding of 8th World Congress on Intelligent Control and Automation (WCICA), 2010* (pp. 2698-2702). IEEE.

Bibliography

Afshar, M. H., Ketabchi, H., & Rasa, E. (2006). Elitist continuous ant colony optimization algorithm: application to reservoir operation problems. *Int J Civil Eng, 4*(3), 274-285.

Ahmed, H., & Glasgow, J. (2012). *Swarm Intelligence: Concepts, Models and Applications.* Technical Report, 2012-585: Queen's University, Kingston, Ontario, Canada K7L3N6.

Al-Nowaihi, A., & Dhami, S. (2010). Probability weighting functions. *Wiley Encyclopedia of Operations Research and Management Science*.

Andriotti, G. K. (2009). Non-Rational Discrete Choice Based On Q-Learning And The Prospect Theory.

Baker, A., Bittner, T., Makrigeorgis, C., Johnson, G., & Haefner, J. (2010). Teaching Prospect Theory with the Deal or No Deal Game Show. *Teaching Statistics,, 32*(3), 81-87.

Bernoulli, D. (1954). Exposition of a new theory on the measurement of risk. *Econometrica: Journal of the Econometric Society*, 23-36.

Birattari, M., Pellegrini, P., & Dorigo, M. (2007). On the invariance of ant colony optimization. *Evolutionary Computation IEEE Transactions on, 11*(6), 732-742.

Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm intelligence: from natural to artificial systems (No. 1).* OUP USA.

Brucker, P. (2007). *Scheduling Algorithms* (Fifth ed.). Springer.

Buniyamin, N., Sariff, N., Wan Ngah, W. A., & Mohamad, Z. (2011). Robot global path planning overview and a variation of ant colony system algorithm. *International journal of mathematics and computers in simulation, 5*(1), 9-16.

Cochran, A. (2001). *Prospect Theory & Customer Choice.*

Dorigo, M. (2006). Ant Colony Optimization and Swarm Intelligence. *: 5th International Workshop, ANTS 2006, Brussels, Belgium, September 4-7, 2006, Proceedings (Vol. 4150).* Springer-Verlag New York Incorporated.

Dorigo, M., & Gambardella, L. M. (1996). A study of some properties of Ant-Q. *Parallel Problem Solving from Nature—PPSN IV*, 656-665.

Edwards, K. D. (1996). Prospect theory: A literature review. *International Review of Financial Analysis, 5*(1), 19-38.

Gaertner, D., & Clark, K. (2005). On optimal parameters for ant colony optimization algorithms. *Proceedings of the 2005 International Conference on Artificial Intelligence, 1*, pp. 83-89.

Gambardella, L. M., & Dorigo, M. (1995, July). Ant-Q: A reinforcement learning approach to the traveling salesman problem. *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE* (pp. 252-260). MORGAN KAUFMANN PUBLISHERS, INC.

Gomez, O., & Baran, B. (2004). Relationship between genetic algorithms and ant colony optimization algorithms. *Proceedings of CLEI2004. Latin-American Conference on Informatics*, (pp. 766-776).

Gong, D., & Ruan, X. (2004, June). A hybrid approach of GA and ACO for TSP. *Intelligent Control and Automation, 2004. WCICA 2004. Fifth World Congress on. 3*, pp. 2068-2072. IEEE.

Gonzalez, R., & Wu, G. (1999). On the shape of the probability weighting function. *Cognitive psychology, 38*(1), 129-166.

Grant, S., & Van Zandt, T. (2007). *Expected Utility Theory.* INSEAD Business School Research Paper, (2007/71).

Iima, H., Kuroe, Y., & Matsuda, S. (2010, October). Swarm reinforcement learning method based on ant colony optimization. *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on* (pp. 1726-1733). IEEE.

Jonker, R., & Volgenant, T. (1983). Transforming asymmetric into symmetric traveling salesman problems. *Operations Research Letters, 2*(4), 161-163.

Kahneman, D. &. (2000). *Choices, values, and frames.* Cambridge, University Press.

Kahneman, D. (2003). Maps of bounded rationality: Psychology for behavioral economics. *The American economic review, 93*(5), 1449-1475.

Kennedy, J. F., Kennedy, J., & Eberhart, R. C. (2001). *Swarm intelligence.* Morgan Kaufmann Pub.

Kern, S., Müller, S. D., Hansen, N., Büche, D., Ocenasek, J., & Koumoutsakos, P. (2004). Learning probability distributions in continuous evolutionary algorithms–a comparative review. *Natural Computing, 3*(1), 77-112.

Khichane, M., Albert, P., & Solnon, C. (2008). Cp with aco. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 328-332.

KomalChadha, N. H. (2011)). Review Of Ant Colony Optimization Algorithms OnVehicle Routing Problems And Introduction ToEstimation-Based ACO. *Proceedings of International Conference on Environment Science and Engineering (ICESE 2011).*

Lee, J. W., Lee, D. H., & Lee, J. J. (2011, May). Global path planning using improved ant colony optimization algorithm through bilateral cooperative exploration. *Digital Ecosystems and Technologies Conference (DEST), 2011 Proceedings of the 5th* (pp. 109-113). IEEE.

Levy, J. (1992). An introduction to prospect theory. Avoiding Losses/Taking Risks: Prospect Theory and International Conflict. (B. Farnham, Ed.) *Political Psychology Special Issue: Prospect Theory and Political Psychology, 13*(2), 171-186.

Li, T., Chen, W., Zheng, X., & Zhang, Z. (2009, September). An improvement of the ant colony optimization algorithm for solving travelling salesman problem (TSP). *Wireless Communications, Networking and Mobile Computing WiCom'09. 5th International Conference* (pp. 1-3). IEEE.

Liberatore, S., Sechi, G. M., & Zuddas, P. (2003). Water distribution systems optimization by metaheuristic approach. *Advances in supply management, Lisse,*, 265-272.

Liu, J. L. (2005). Rank-based ant colony optimization applied to dynamic traveling salesman problems. *Engineering Optimization, 37*(8), 831-847.

Mullen, R. J., Monekosso, D., Barman, S., & Remagnino, P. (2009). A review of ant algorithms. *Expert Systems with Applications, 36*(6), 9608-9617.

Nilsson, C. (2003). Heuristics for the traveling salesman problem. (pp. 1-6). Linkoping University.

Nobahari, H., & Pourtakdoust, S. H. (2005). Optimal fuzzy CLOS guidance law design using ant colony optimization. *Stochastic Algorithms: Foundations and Applications*, 95-106.

Ostfeld, A. (Ed.). (2011). *Ant Colony Optimization - Methods and Applications.* InTech.

Ponnambalam, S. G., Jawahar, N., & Girish, B. S. (n.d.). An Ant Colony Optimization algorithm for Flexible Job shop scheduling problem.

Randall, M., & Lewis, A. (2010, December). . Modifications and additions to ant colony optimisation to solve the set partitioning problem. *e-Science Workshops, 2010 Sixth IEEE International Conference on* (pp. 110-116). IEEE.

Ridge, E., Kudenko, D., & Kazakov, D. (2006, July). A Study of Concurrency in the Ant Colony System Algorithm. *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on* (pp. 662-1669). IEEE.

Runarsson, T. P., & Yao, X. (2000). Stochastic ranking for constrained evolutionary optimization. *Evolutionary Computation, IEEE Transactions on, 4*(3), 284-294.

Sarigiannidis, G. (2005). Discussion on the implementation of the Ant Colony System for the Traveling Salesman Problem. Delft University of Technology Electrical Engineering. *Mathematics and Computer Science Department of Information Systems and Algorithm*.

Selvi, V., & Umarani, R. (2010). Comparative analysis of ant colony and particle swarm optimization techniques. *International Journal of Computer Applications, 5*(4).

Siefert, W. T., & Smith, E. D. (2011, March). Cognitive biases in engineering decision making. *In Aerospace Conference, 2011 IEEE* (pp. 1-10). IEEE.

Socha, K., & Dorigo, M. (2007). *Ant colony optimization for mixed-variable optimization problems.* IRIDIA. No. TR/IRIDIA/2007, 19.

Tversky, A., & Kahneman, D. (1981). The framing of decisions and the psychology of choice. *Science, 211*(4481), 453-458.

van Ast, J. M., Babuška, R., & De Schutter, B. (2010). Ant colony learning algorithm for optimal control. *Interactive Collaborative Information Systems*, 155-182.

Wu, G., Zhang, J., & Gonzalez, R. (2004). Decision under risk. In D. J. Koehler, & N. Harvey (Eds.), *Blackwell handbook of judgment and decision making* (pp. 399-423).

Zhang, Y., Pellon, M., & Coleman, P. (2007, April). Decision Under Risk in Multi-Agent Systems. . *In System of Systems Engineering, 2007. SoSE'07. IEEE International Conference on* (pp. 1-6). IEEE.