Investigating the Application of Opposition-Based Ideas to Ant Algorithms

by

Alice Ralickas Malisia

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2007

Alice R. Malisia

Abstract

Opposition-based learning (OBL) was recently proposed to extend different machine learning algorithms. The main idea of OBL is to consider opposite estimates, actions or states as an attempt to increase the coverage of the solution space and to reduce exploration time. OBL has already been applied to reinforcement learning, neural networks and genetic algorithms. This thesis explores the application of OBL to ant algorithms. Ant algorithms are based on the trail laying and following behaviour of ants. They have been successfully applied to many complex optimization problems. However, like any other technique, they can benefit from performance improvements. Thus, this work was motivated by the idea of developing more complex pheromone and path selection behaviour for the algorithm using the concept of opposition.

This work proposes opposition-based extensions to the construction and update phases of the ant algorithm. The modifications that focus on the solution construction include three direct and two indirect methods. The three direct methods work by pairing the ants and synchronizing their path selection. The two other approaches modify the decisions of the ants by using opposite-pheromone content. The extension of the update phase lead to an approach that performs additional pheromone updates using opposite decisions.

Experimental validation was done using two versions of the ant algorithm: the Ant System and the Ant Colony System. The different OBL extensions were applied to the Travelling Salesman Problem (TSP) and to the Grid World Problem (GWP). Results demonstrate that the concept of opposition is not easily applied to the ant algorithm. One pheromone-based method showed performance improvements that were statistically significant for the TSP. The quality of the solutions increased and more optimal solutions were found. The extension to the update phase showed some improvements for the TSP and led to accuracy improvements and a significant speed-up for the GWP. The other extensions showed no clear improvement.

The proposed methods for applying opposition to the ant algorithm have potential, but more investigations are required before ant colony optimization can fully benefit from opposition. Most importantly, fundamental theoretical work with graphs, specifically, clearly defining opposite paths or opposite path components, is needed. Overall, the results indicate that OBL ideas can be beneficial for ant algorithms.

Acknowledgements

I would like to thank my supervisor, Professor H. R. Tizhoosh, for his valuable support and encouragement, which led to the creation of this thesis. I would also like to extend my thanks to my readers, Professor F. Karray and Professor J. Kofman, for their time and feedback. Finally, I thank the Natural Sciences and Engineering Research Council of Canada for providing the necessary funding for this research.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Optimization problems are commonly faced by industry, the scientific community and are present even in our everyday lives. Better optimization algorithms and methods are constantly being developed to attempt to solve these complex problems. These complex optimization problems can be solved using sophisticated machine learning methods such as genetic algorithms, neural networks, and ant algorithms to name a few [9]. These machine learning algorithms are successful, but they can always benefit from performance improvements. Recently, the concept of Opposition-Based Learning (OBL) was introduced as a new way to extend machine learning algorithms [31–33].

The idea of opposition might be relatively new in the world of optimization, but it is a concept that is existent all around us in nature, society, and other fields. It is a concept that is everywhere: day/night, hot/cold, negative/positive, multiplication/division. Thus, OBL was is inspired from examples in the real world.

Opposition-based learning was introduced recently [31–33], but it has already been succesfully applied in three main types of algorithms: reinforcement learning [25, 26, 31], genetic algorithms [17–20], and neural networks [34, 35]. Given the success and the possibility of exploring new approaches, it was encouraging to extend the work to another class of algorithms: swarm intelligence. One important algorithm in that class is ant colony optimization.

Ant colony optimization is a powerful technique that has been used to solve many complex optimization problems, such as the travelling salesman problem, quadratic assignment problem, vehicle routing, and others [8]. Given the complexity of these problems and the real-world applications, any improvement is encouraged and strongly welcomed. Despite its successes, ant colony optimization is not a perfect algorithm. Like many other optimization techniques, it can remain trapped in a local optima, miss a portion of the solution space or, in some cases, it can be slow to converge. Thus, it is interesting to study and develop more complex behaviour for the ant algorithm.

Essentially, this work involves an initial investigation of the application of OBL ideas to ant colony optimization. It aims at exploring how the opposition framework can be extended to the ant algorithm and it also attempts to evaluate this framework with two different applications.

This thesis proposes two main types of opposition-based extensions to the ant algorithm. The first type involves modifications to the construction phase of the ant algorithm. The second type focuses on the pheromone update phase of the algorithm.

The construction phase modifications include different classes of modifications: direct and indirect. Direct modifications to the construction phase mean that the decisions made by the ants are controlled. The three direct modification algorithms proposed in this thesis are: Synchronous Opposition, Free Opposition and Free Quasi-Opposition. An indirect modification affects the decisions of the ants by altering decision parameters, such as pheromone. The two indirect extensions are: Opposite Pheromone per Node and Opposite Pheromone per Edge.

Extending the update phase of the ant algorithm led to a single type of algorithm that involves additional updates to the pheromone content of opposite decisions. The extension to the update phase is called: Opposite Pheromone Update.

Two different applications were used to evaluate the opposition-based approaches: the Travelling Salesman Problem (TSP) and the Grid World Problem (GWP). All the opposition-based approaches were applied to the TSP, but only the Opposite Pheromone per Node and the Opposite Pheromone Update methods were applied to the GWP. The performance of the proposed OBL algorithms is compared to the performance of the normal ant algorithms. The experiments aim at assessing the validity of the different OBL extensions, as well as establishing the applicability of opposition to ant colony optimization.

The remaining of this thesis is organized as follows. Chapter 2 provides a background of work that has been conducted to improve the performance of ant algorithms. Chapter 3 gives an overview of ant colony optimization. Chapter 4 presents the concept of OBL. Chapter 5 and 6 include the experimental results for the OBL ant algorithms for the travelling salesman problem and the grid world problem respectively. A detailed discussion of the results is included in chapter 7, which is followed by the conclusions and future work in chapter 8.

Chapter 2

Background

The concept of Opposition-Based Learning (OBL) was recently proposed a new extension to learning algorithms [31–33]. The idea underlying OBL is that by considering opposite estimates, actions, weights, *etc.*, one can improve the coverage of the solution space for a particular problem. In turn, the improved coverage can lead to better covergence and/or higher accuracy. A detailed discussion of OBL is included in chapter 4. OBL has already been successfully applied to reinforcement learning [25,26,31], evolutionary algorithms [17–20] and neural networks [34,35]. Thus, there was motivation to investigate the application of opposition to a new algorithm: Ant Colony Optimization (ACO).

Like some other machine intelligence methods, ACO algorithms are based on a phenomenon occurring in nature: the social behaviour of ant colonies [8]. Ants are well-known for their ability to efficiently find the shortest path between their nest and their food source [1]. ACO implementations have been successfully applied to many complex optimization problems, such as the travelling salesman problem [8]. The ant algorithm is described in detail in the following chapter.

Despite being a powerful algorithm, ACO can benefit from performance improvements. On one hand, ACO has many applications and deals with complex optimization problems. Thus, any increase in speed of convergence is beneficial. On the other hand, given the fundamental structure of ACO, the algorithm can sometimes remain trapped in local optima resulting in reduced accuracy. This situation can occur when a certain component is very desirable on its own, but leads to a sub-optimal solution when combined with other components. Consequently, modifications that can help increase the accuracy of the algorithm are also welcomed.

Since the introduction of ACO, researchers have developed multiple versions to improve the performance of the algorithm. The Ant Colony System (ACS) is a commonly used extension of the original ant algorithm [8]. The ACS algorithm has a greedy selection rule, but provides regular pheromone reduction as a measure to decrease desirability of arcs once they are travelled [6]. This prevents all the ants in the colony from generating the same solution. Another successful version of the ant algorithm is the Max-Min Ant System (MMAS) [8, 29]. The MMAS strongly exploits the best tours found, but the MMAS also limits the range of pheromone content values and initializes the pheromone contents at the upper limit. These modifications led to performance improvements.

In addition, work has been conducted to establish more complex pheromone mechanisms, such as multiple pheromone matrices, and complex pheromone updates. These modifications were implemented so ant algorithms could solve more complex problems and to improve the performance of the ACS. For instance, one particular variant of the ant algorithm, known as the Best-Worst Ant System (BWAS) [2], substracts pheromone content based on the results of the worst ant of the colony. The BWAS also uses a form of pheromone mutation based on concepts from evolutionary computation. To solve a bi-criterion vehicle routing problem, Iredi, Merkle and Middendorf proposed a version of the ACS where two different pheromone trail matrices and two heuristic functions are considered simultaneously [11]. Randall and Montgomery proposed the Accumulated Experience Ant Colony as a method to determine the effect of each component on the overall solution quality [21]. In their approach, the pheromone and heuristic values of an edge are weighted.

Schoonderwoerd and his colleagues were one of the first to elude to the concept of an ‘anti-pheromone’, where ants would decrease pheromone contents rather than reinforce them [24]. Montgomery and Randall developed three methods based on the concept of anti-pheromone as an attempt to capture complex pheromone behaviour [15]. In the first method, the pheromone of the elements composing the worst solutions is reduced. Their second alternative combines a pheromone content for the best solution and pheromone content for the worst solution. The ants select edges based on a weighted combination of pheromone and anti-pheromone and the heuristic. Finally, their third approach involves the use of a small number of *explorer ants* that have a reversed preference for the pheromone. Their methods produced better solutions on the smaller TSP problems.

Given these existing extensions, their results, and the potential for performance improvement, there was strong motivation to investigate the application of opposition to ant colony optimization. OBL can potentially lead to a new way of developing more complex pheromone or path selection behaviour for ant algorithms.

Chapter 3

Ant Colony Optimization

3.1 Background

Ant Colony Optimization (ACO) is classified under the general class of algorithms known as Swarm Intelligence (SI). SI reflects the emergence of collective intelligence from a swarm of simple agents. It is generally defined as a structured collection of interacting organisms which cooperate to achieve a greater goal [1,12]. It is possible to have genetic cooperation, as in the case of genetic algorithms, but in SI, it is more of a social interaction. The framework is based on the repeated sampling of solutions to the problem at hand, where each member of the population provides a potential solution. In the case of ACO, the algorithm mimics the social interaction of ants; thus, the population is a colony of ants.

Social behaviour increases the ability of individuals to adapt, as they can cooperate and learn from each other. The main idea of SI algorithms is that organisms of a swarm behave in a distributed manner while exchanging information directly or indirectly. The general characteristics of SI algorithms are [12]:

- a collection of autonomous agents;
- distributed control among the agents;
- agents interact and communicate;
- agents have a stochastic component (usually for decisions) to encourage exploration;

- agents use collective knowledge to make the stochastic decision.

One important element of SI is agent communication. That is the key for the collective intelligence because agents share the information. There are two main types of agent communication: 1) broadcast-like and 2) indirect [1, 12]. In indirect communication, two individuals interact indirectly when one of them modifies the environment and the other responds to the modified environment at a later time. This phenomenon is called *stigmergy*. A classical example of this is the pheromone deposits present in ant colonies.

3.2 Natural metaphor

The ACO algorithm is inspired from the natural behaviour of trail laying and following by ants [1, 8]. When exploring a region, ants are able to find the shortest path between their nest and a food source. They can adapt to changes in the environment and restructure their path around new obstacles. Their optimizing capacity is a result of their ability to communicate indirectly with each other via pheromone, which is a chemical the ants leave behind as they travel. The pheromone deposited by one ant influences the selection of the path by other ants. A high pheromone concentration increases the probability that the path will be selected. Additionally, pheromone evaporates over time. The pheromone deposits work as a form of positive feedback, reinforcing good path choices and guiding the ants to better paths. Fig. 3.1 depicts the behaviour of ants over a period of time when they are presented with alternate paths.

3.3 History

The ACO algorithm was introduced by Marco Dorigo in 1992 in his PhD thesis [3]. It was developed to solve complex discrete combinatorial problems. The first ACO algorithm was the Ant System (AS) [5], which was designed to solve the Travelling Salesman Problem (TSP). The TSP is an optimization problem based on the problem faced by a travelling salesman who, given a starting city, wants to take the shortest trip through a set of customer cities, visiting each city once before returning to the starting point. Mathematically, the TSP involves finding the minimum cost path in a weighted graph. A particular TSP

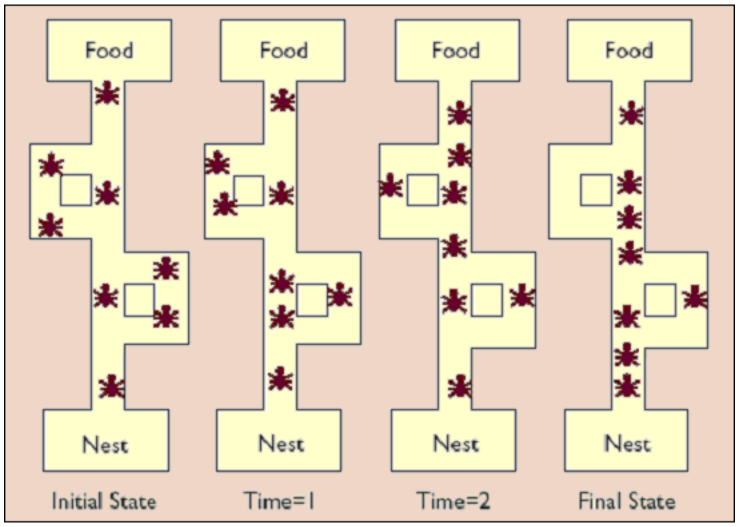

Figure 3.1: Behaviour of ants when they are presented with alternate paths [23].

instance has a specific number of cities (nodes) and arc weights (typically the distance between the cities). The AS algorithm performed well with TSP instances up to about 50 cities, then it generally did not converge to an optimum.

Since the introduction of ACO, researchers have developed multiple versions to improve the performance of the algorithm. The Ant Colony System (ACS) is a popular revised version of ACO [8]. It achieved considerable accuracy improvements [6,8]. Other extensions of the original ACO algorithm include the Best-Worst Ant System [2], the Max-Min Ant System [29], Ant-Q (extends ACO with reinforcement learning), AntNet (dynamic version of the algorithm designed for the vehicle routing problem), and the ACS combined with local search [8].

3.4 The ACO algorithm

When dealing with complex optimization problems, it is generally necessary to use metaheuristics to solve them. Metaheuristics are procedures that use heuristics to seek a near-optimal solution with reasonable computation time [7,8]. The general idea behind a metaheuristic is to create a balance between local improvements and a high-level strategy. They optimize problems through guided search of the solution space [8,27]. In brief, metaheuristics seek optimality while attempting to reduce computation time. Their goal is to search the space efficiently to find near-optimal solutions. ACO is a metaheuristic.

When applied to an optimization problem, the ACO metaheuristic usually involves solution construction on a graph. Ants will move between nodes, sequentially adding edges to their current path until they have visited all nodes. The selection of an edge depends on the pheromone content, represented by values in a $n \times n$ matrix where n is the number of nodes, and the heuristic function values of the available edges. The pheromone content is a form of positive reinforcement to influence future constructions and the heuristic guides the search to potential promising solutions.

ACO has three critical characteristics [27]: positive feedback, distributed computation, and local heuristic. The positive feedback increases the rapidity at which good solutions are found. The distributed computing, embodied by large number of ants working together, avoids premature convergence to suboptimal solutions. Finally, the addition of a local heuristic leads to finding good solutions in the earlier stages of the optimization. Dorigo and Stützle explain the uniqueness of the ACO algorithm [8]: “The ACO uses a population (colony) of ants which construct solutions exploiting a form of indirect memory called artificial pheromones”. The following sections will provide a detailed description of two important versions of ACO, namely the AS and the ACS.

3.4.1 Ant System

As previously mentioned, the AS was the first instance of ACO. The basic steps are:

1. Represent problem as a graph.
2. Initialize the pheromone matrix.
3. Each ant constructs a solution.
4. Evaporate pheromone matrix.
5. Update the pheromone content on the path travelled by each ant.
6. Repeat steps 3 through 5 until the termination criterion is satisfied.

Problem representation

Problems are usually represented in the form of a graph, where solutions are a path along the graph. The ants will travel from node to node constructing a full solution in each iteration.

Initialization

The initialization involves setting all the values of the pheromone matrix to a small value, which is equivalent to dropping an initial amount of pheromone on all edges of the graph. The number of ants, m, is usually smaller than the number of nodes, n ($m < n$). The initial pheromone content, τ_o is usually based on the worst case solution, thus a very small amount [8]. A good value for τ_o is m/C_{nn}, which is the total number of ants, m, divided by the cost of the nearest neighbour path, C_{nn} [8]. The nearest neighbour path cost is the cost of the solution achieved by always moving to the nodes connected by the arc with the lowest weight. Following initialization, the AS has two main phases: solution construction and trail update. The trail update includes evaporation and pheromone update.

Solution construction

The ants are initially distributed randomly among the nodes. Ants travel through the graph adding solution components to partial solutions until they reach a complete solution. They build their solutions stochastically. The selection of the components depends on the pheromone content of the paths and a heuristic value. At each step of construction, ant k selects the next node using a probabilistic action choice rule, which dictates the probability with which ant k will choose to go from current node i to next node j:

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in \mathrm{N}_i^k} [\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}} \qquad \text{if } j \in N_i^k, \tag{3.1}$$

where τ_{ij} represents the pheromone content on the edge connecting city i to city j. The city j is a node that is included in N_i^k, the neighbourhood for ant k given its current location i. The neighbourhood only includes nodes that have not been visited by ant k. If all feasible nodes have been visited, then all neighbours of the current node become available for visit. The constants α and β represent the influence of pheromone content and heuristic respectively. If $\alpha = 0$, the AS simply becomes a greedy algorithm with random starts. Similarly, if $\beta = 0$, it would mean that there is no heuristic bias, which might lead the ants to premature convergence. Experimental results suggest setting $\alpha = 1$ and β from 2 to 5 [8]. Finally, η_{ij} is the heuristic information for going from node i to node j. The heuristic value of an arc is a measure of the cost of extending the current partial solution with that arc (typically the inverse of the weight of the arc). For example, in the TSP, η_{ij} is usually set to $1/d_{ij}$, the reciprocal of the distance between the two nodes. The heuristic should guide the search to more promising solutions. The stochastic component of the algorithm, namely selecting a component based on a probability, leads to exploration of a higher number of solutions because components with lower probability can be selected.

Evaporation

In the AS, when all ants have completed their paths, pheromone is evaporated. Pheromone evaporation involves decreasing the pheromone content of arcs over time. As the number of iterations increases, the pheromone content of unvisited arcs drops. This is an important factor to reduce the chance of premature convergence, as it gives ants a chance to visit arcs that were not visited in the initial iterations. It works as a "forgetting" mechanism. It is important to ensure that local optima are not reinforced [1]. The pheromone evaporation is applied to all arcs following the relation

$$\tau_{ij}^{new} = (1-\rho)\tau_{ij}^{current} \quad 0 < p < 1, \tag{3.2}$$

where τ represents the pheromone content of the arcs and ρ is the evaporation rate. Research indicates that $\rho = 0.5$ is appropriate for the AS [8].

Pheromone update

After the evaporation, the solutions are evaluated and pheromone is deposited relative to the quality of the solution. The ants deposit pheromone on the arcs they visited as follows:

$$\tau_{ij}^{new} = \tau_{ij}^{current} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \tag{3.3}$$

where $\Delta\tau_{ij}^{k}$ is the amount of pheromone ant k contributes to the arc going from node i to node j and m is the total number of ants. The additional pheromone is based on the overall quality of the total path and is defined by

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{C_k} & \text{if arc is in the path of ant } k, \\ 0 & \text{otherwise,} \end{cases} \tag{3.4}$$

where C_k is the total cost of the solution for ant k. In the TSP, it represents the length of the path for ant k. All arcs of one path will receive the same amount of pheromone.

Termination

The algorithm can be terminated after a specific number of cycles, when all ants are travelling the same path (stagnation) or when the quality of the solution has reached a desired value.

Summary

The AS algorithm is summarized in Table 3.1.

Table 3.1: AS Algorithm

```
Initialize pheromone matrix ($\tau = \tau_o$)
Repeat until termination condition is satisfied
  Repeat until solution is constructed (for each ant $k$):
    Pick next node $j$
  Evaporate pheromone
  Apply pheromone trail update
```

3.4.2 Ant Colony System

Another commonly used version of ACO is the ACS. This version differs from the AS algorithm in three aspects [6, 8]: 1) different selection rule for tour construction, 2) trail update only occurs for the best-so-far solution, and 3) local pheromone removal occurs each time an ant visits a node.

Selection rule

When ants construct their paths in the ACS, they use a selection rule that has a strong emphasis on exploitation of previous experience. An ant k located on node i chooses the next node j using a pseudorandom proportional rule described by

$$j = \begin{cases} \underset{l \in \mathrm{N}_i^k}{\operatorname{argmax}} \left\{ \tau_{il}[\eta_{il}]^{\beta} \right\} & \text{if } q < q_o, \\ J & \text{otherwise.} \end{cases} \tag{3.5}$$

The parameter q is a uniform random number and q_o is the probability that an ant will use learned knowledge. If $q < q_o$, the ant will select the node with the highest product of pheromone content and heuristic function value. Otherwise, it will use J, which is a random variable selected by the probabilistic action rule used in the AS (see Eq. 3.1). Note that if $q_o = 0$, the pseudorandom rule is reduced to the selection rule from the AS. In contrast, if $q_o \approx 1$, the search is highly focused on experience by searching around the best-so-far solution. In general, experimental results indicate that selecting $q_o = 0.9$ leads to good results [8]. This pseudorandom rule is a greedy selection approach that will tend to favour short edges with high pheromone.

Best trail update

The ACS has a pheromone update approach that exploits the best solutions found. The evaporation and deposit of pheromone is only applied to the arcs contained in the current best solution. The update is implemented by

$$\tau_{ij}^{new} = (1-\rho)\tau_{ij}^{current} + \rho(\Delta\tau_{ij}^{bs}) \quad \forall (i,j) \in T^{bs}, \tag{3.6}$$

where $\Delta\tau_{ij}^{bs}$ is additional pheromone, ρ is the global evaporation rate, and T^{bs} is the best-so-far path. Sometimes the best-iteration path is used for smaller problems. The additional pheromone is calculated using the cost of the best-so-far path. Research indicates that $\rho = 0.1$ is an appropriate value for the ACS algorithm [8].

Local update

The ACS includes a local pheromone update to reduce emphasis on exploitation of existing solutions. Immediately after an ant adds an arc to its current path the amount of pheromone on the arc is decreased as follows:

$$\tau_{ij}^{new} = (1-\xi)\tau_{ij}^{current} + \xi\tau_o \quad 0 < \xi < 1, \tag{3.7}$$

where τ_o is the initial amount of pheromone. In the case of the TSP, research indicates that for the ACS, this value should be set to $1/mC_{nn}$, where m is the number of ants, n represents the number of cities and C_{nn} is cost of the nearest neighbour solution. The parameter ξ is the local evaporation rate, which is typically set to 0.1 [8]. This local update works to counterbalance the greedy construction rule by reducing the pheromone on the selected edge, thus making it less desirable to the next ant.

Summary

The general steps of the ACS algorithm are summarized in Table 3.2.

Table 3.2: ACS Algorithm

```
Initialize pheromone matrix ($\tau = \tau_o$)
Repeat until termination condition is satisfied
  Repeat until solution is constructed (for each ant $k$):
    Pick next node $j$
    Apply local pheromone update
  If necessary, update best-so-far solution
  Apply best trail update
```

3.5 Applications

The ACO metaheuristic has mostly been applied to solve common optimization problems such as the travelling salesman problem, quadratic assignment, and vehicle routing [8].

More recently, there have been a wide set of new applications, namely sequential ordering, graph coloring, generalized assignment, multiple knapsack and constraint satisfaction [8], cell planning for mobile computing [30], power systems optimization [27], and the shortest common supersequence problem [14]. The ACO metaheuristic was also applied to dynamic shortest-path problems [7]. A very active area of research is the application of ACO algorithms to telecommunication problems, specifically network routing [4, 8]. While the focus of ACO has been optimization problems, researchers have also begun to apply it to fields such as image processing [13, 16, 36].

Chapter 4

Opposition-Based Learning

Opposition-based learning (OBL) was proposed by Tizhoosh as a possible new way to improve the performance of machine learning algorithms [31–33]. The main idea of OBL is that by considering "opposites", one can increase the coverage of the solution space leading to increased accuracy and/or faster convergence. OBL provides a general strategy that can be tailored to the technique of interest.

4.1 Theory

While the idea of opposition might be new in the area of algorithms, it is prevalent in the world around us: male/female, up/down, day/night, etc. The interplay between opposites apparently provides a state of balance. Then, it is only natural that opposition may be a possibility to improve algorithms.

Whenever one is looking for the solution x of a given problem, one usually makes an estimate $\hat{x}$. This estimate is not the exact solution and can be based on experience, a heuristic or a totally random guess. Sometimes, the estimate $\hat{x}$ may be sufficient, as it may have reasonable fitness and accuracy. In most cases, the initial guess $\hat{x}$ is not satisfactory; thus, the estimated value must be modified to move closer to the optimal value.

In machine learning algorithms, learning usually begins at a random point and then moves towards better solutions. For example, the weights of a neural network are initialized randomly, the parameter population in genetic algorithms is configured randomly and the action policy of reinforcement agents is initially based on randomness. The random guess, if not far away from the optimal solution, can result in a fast convergence. However, it is natural to state that when beginning with a random guess that is very far from the existing solution, then the approximation, search or optimization will take considerably more time, or can even become intractable. Of course, in the absence of any *a priori* knowledge, it is not possible to make the best initial guess. Logically, the algorithm should be looking in all directions simultaneously, or more concretely, in the opposite direction. Consequently, if the algorithm is searching for x, and one agrees that searching in the opposite direction could be beneficial, then calculating the *opposite number*, $\breve{x}$, is the first step.

Definition (Opposite Number) - Let $x \in \Re$ be a real number defined on a certain interval: $x \in [a, b]$. The *opposite number*, $\breve{x}$, is defined as follows:

$$\breve{x} = a + b - x. \tag{4.1}$$

Analogously, the opposite number in a multidimensional case can be defined.

Definition (Type-I Opposition) - Let $P(x_1, x_2, \ldots, x_n)$ be a point in a n-dimensional coordinate system with $x_1, x_2, \ldots, x_n \in \Re$ and $x_i \in [a_i, b_i]$ $\forall i \in [1, n]$. The opposite point $\breve{P}$ is completely defined by its coordinates $(\breve{x}_1, \breve{x}_2, \ldots, \breve{x}_n)$ where

$$\breve{x}_i = a_i + b_i - x_i. \tag{4.2}$$

Type-I Opposition is applied to candidate solutions, P. If the points P and $\breve{P}$ are candidate solutions for a function $f(x_1, x_2, \ldots, x_n)$ then it is not always possible to ensure that $f(x)$ will be the mathematical opposite of $f(\breve{x}_1, \breve{x}_2, \ldots, \breve{x}_n)$. The type-I opposite will deliver an opposite output only for linear or quasi-linear functions. Thus, a type-II opposite was defined.

Definition (Type-II Opposition) Let $y = f(x_1, x_2, \ldots, x_n) \in \Re$ be an arbitrary function with $y \in [y_{min}, y_{max}]$. For every point $P = (x_1, x_2, \ldots, x_n)$, the opposite point $\breve{P} = (\breve{x}_1, \breve{x}_2, \ldots, \breve{x}_n)$ is defined by

$$\breve{x} = \{x | \breve{y} = y_{\min} + y_{\max} - y\}. \tag{4.3}$$

This definition assumes that the function $f(x)$ is not known, but y_{min} and y_{max} are given or can be estimated. A type-II opposite is difficult to calculate as it generally involves some

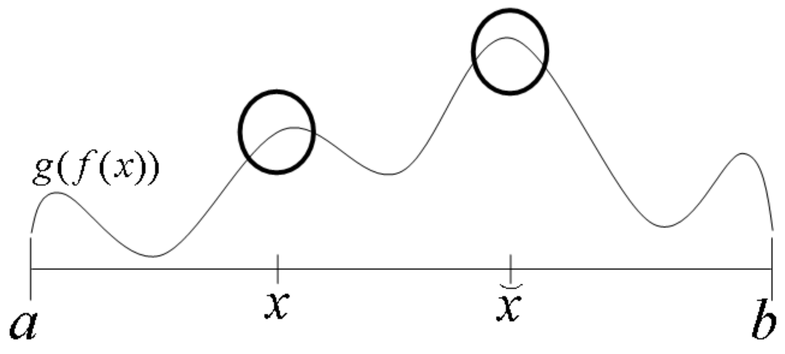

Figure 4.1: Illustration of one-dimensional opposition.

a priori knowledge about the output function. Consequently, type-I opposites are used as an approximation for type-II opposition. The general opposition scheme for learning can now be concretized.

Opposition-Based Learning - Let $f(x)$ be the function in focus and $g(\cdot)$, the be the proper fitness evaluation function. If $x \in [a, b]$ is an initial (random) guess and $\breve{x}$ is its opposite value, then in every iteration $f(x)$ and $f(\breve{x})$ are calculated. The learning continues with x if $g(f(x)) \geq g(f(\breve{x}))$, otherwise it continues with $\breve{x}$.

OBL is illustrated in Fig. 4.1 for the one-dimensional case. For example, given the current guess x, considering its mathematical opposite (as defined in equation 4.1) can lead to a better area in the solution space. The consideration of the mathematical opposite provides a better coverage than a simple second random guess would achieve [17, 19]. In addition, one can note that, in this case, x and $\breve{x}$ have a one-to-one correspondence.

4.2 Existing OBL extensions

In order to fit in the OBL scheme, one must relate the term *opposite* to the specific algorithm. The general OBL idea can be applied in many different ways. OBL has been successfully applied to reinforcement learning [25, 26, 31], differential evolution [17–20], and neural networks [34, 35].

In reinforcement learning, opposition is used to accelerate the learning process by performing additional updates of opposite actions [25, 26, 31]. The work was conducted using the Q-learning algorithm. The main idea is that when one action of the agent is rewarded, the opposite action is simultaneously punished. Results indicate that the OBL extension leads to an improvement in the learning speed, as well as a better success rate. The best results occur when there is a diminishing consideration of the opposite actions.

In neural networks, OBL was applied to a typical multi-layer perception network by incorporating opposite transfer functions [34,35]. The use of the opposite transfer functions led to an improved search of the weight space. Results showed significant improvement over standard backpropagation learning. The accuracy was comparable, but convergence was much faster.

In differential evolution, which is a type of genetic algorithm, OBL was implemented during population initialization and used for generation jumping. Generation jumping means that, based on a jumping rate, a new population is generated using opposition instead of the regular genetic operators. This new population is a mathematically opposite population. Then, the fitness of the original and opposite populations are calculated and only the best individuals are kept for further optimization [17–20]. The boundaries for opposite generation are dynamic and decrease as the population becomes more concentrated. The opposites generated within the OBL scheme are used to efficiently generate guesses that have a chance of having a better fitness than simply random guesses. In this approach, opposition is a way to reach far points in the solution space, which may have better fitness. The results indicate that the OBL algorithms accelerate the convergence.

4.3 Opposition and ACO

In the case of ACO, the application of opposition is not as straightforward as mapping between two estimates. ACO usually optimizes combinatorial problems, like TSP instances. Thus, the opposite of solutions and partial components of the solutions are not clearly defined. Even if only one element of the solution is changed, it generates a whole set of new solutions. Moreover, simply taking the opposite of every component of the solution might not necessarily lead to a plausible solution. If one refers back to the one-dimensional example (see Fig. 4.1), one can see that it is not clear how one can generate an opposite solution, mainly because of the combinatorial aspect of the applications associated with

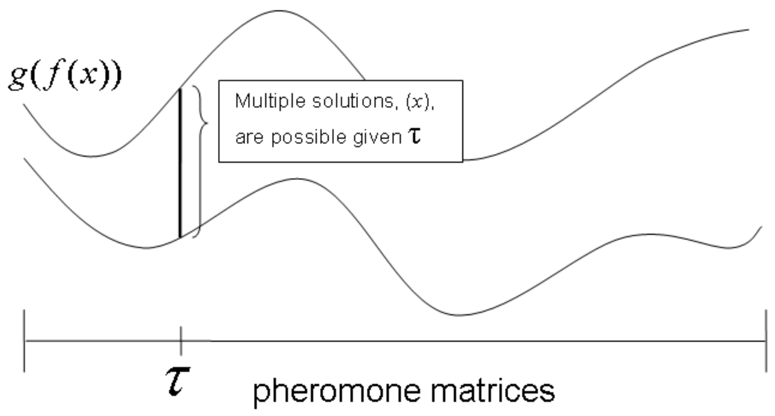

Figure 4.2: Illustration of pheromone matrix opposition.

ACO. To fit in the OBL scheme, the term *opposite* must be related to ant algorithms.

The concept of opposition [31,33] which was described in section 4.1 serves as a starting point for he proposed extensions. The main idea was to think of opposition as a way of increasing the coverage of the solution space. The goal was to attempt to extend the idea of opposite estimates illustrated in Fig. 4.1. Ant algorithms do not work by evolving solutions; instead, at each iteration, new solutions are created based on the pheromone matrix. It is the pheromone matrix that changes as the algorithm progresses.

In algorithms that work with complete solutions, such as genetic algorithms, one can generate an opposite candidate solution and replace the current candidate solution. Then the evolution proceeds with the opposite candidate solution. In contrast, in the ant algorithm, even if the opposite solution is generated, one needs to find a way to alter the pheromone content since that is what affects solution creation. To move in the solution space, the algorithm has to move in the pheromone space. Thus, instead of looking at a solution candidate and its opposite, the concept illustrated in Fig. 4.1 is extended to involve the pheromone matrix and its opposite. This is illustrated in Fig. 4.2.

One can note that, because of the probabilistic nature of the path selection in ACO, a particular pheromone matrix can lead to an array of solutions. This leads to an array of fitness values and, hence, there is no one-to-one relationship between a particular matrix and fitness that exists in the candidate solution-based opposites. Additionally, an opposite pheromone matrix is not easily defined. In a classical ant algorithm, the pheromone matrices are modified by the ants, which is how the algorithm moves between pheromone matrices and solutions. The idea was to find a way to use opposition to move in the pheromone matrix solution space. It was determined that opposition can be applied in two main parts of the ACO algorithm: 1) the construction phase and/or 2) the update phase.

The construction phase can be modified by affecting the ant's decision. This can be done directly or indirectly. Directly means that opposition is used to control the decision of the ant by restricting and reducing the number of available choices. The indirect modification involves altering the parameters used by the decision, namely the pheromone content.

The modification to the update phase involves altering the way the pheromone is updated. It can be done by making additional updates using other ants. A form of this idea was implemented in the Best-Worst Ant System [2], which uses the worst-ant to remove pheromone. However, there are other ways to affect the update phase. It can also involve the best paths. One possible way is to use opposite components of the solution without necessarily creating an opposite solution.

Consequently, with the proposed OBL modifications, the algorithm is able to move to a new region of pheromone matrices. By changing the decision of the ants or changing the pheromone content used in the decision, one simulates the creation of another pheromone matrix without directly changing the current matrix. In contrast, in the case of opposition-based pheromone updates, the algorithm is actually moving to a new pheromone matrix. It will probably not be the *opposite* pheromone matrix, but it may eventually lead to an area closer to the optimal solution.

The discussed modifications provide a general framework as to how opposition can extend ACO. These ideas were used to design specific OBL algorithms for two applications: the Travelling Salesman Problem and the Grid World Problem. The following two chapters will provide a very detailed description of the OBL algorithms used to solve these two problems. This thesis presents three direct approaches (Synchronous Opposition, Free Opposition, and Free Quasi-Opposition), two indirect approaches (Opposite Pheromone per Node, Opposite Pheromone per Edge), and one method that modified the update phase (Opposite Pheromone Update).

Chapter 5

Travelling Salesman Problem Experiments

The Travelling Salesman Problem (TSP) is a classical optimization problem which has been used for the evaluation of the performance of ant algorithms. This chapter presents experiments on TSP instances comparing the performance of ant colony optimization with opposition-based extensions to the ant algorithm. The implementations use the Ant Colony System (ACS) version of the ant algorithm. The proposed methods include five extensions to the contruction phase and one extension to the update phase of the algorithm.

5.1 Construction phase extensions

The first three approaches, namely Synchronous Opposition, Free Opposition, and Free Quasi-Opposition, directly change the decisions of the ants. They use the idea of paired ants (ant and opposite ant) searching the space. By pairing ants and synchronizing their construction, one can reduce the randomness. The other two methods, Opposite Pheromone per Node and Opposite Pheromone per Edge, follow the indirect approach by modifying decision parameters. They use opposite pheromone values. While the two last approaches resemble the *explorer ants* proposed in [15], they differ in that the entire colony is subject to the possibility of using opposite pheromone content. Also, the opposite pheromone is not always activated. The five versions will be described in detail in the following subsections.

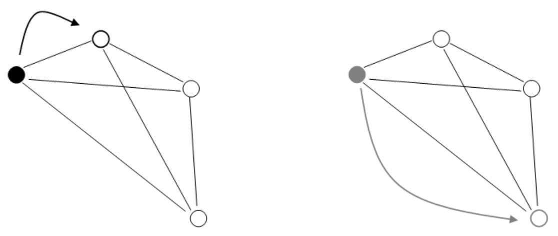

Figure 5.1: Synchronous Opposition: *leading-ant* (left) and *opposite-ant* (right) on the same city.

5.1.1 Synchronous Opposition

The Synchronous Opposition approach is the most rigid in terms of synchronicity. The colony is divided in two and the ants are paired. Each pair follows a similar construction behaviour. The pairs start on a randomly selected city, meaning the two paired ants will start on the same city. The first ant of a pair (*leading-ant*) follows the usual selection rules, but the second ant (*opposite-ant*) picks its next city based on the decision of the *leading-ant*. If the *opposite-ant* was on the same city as the *leading-ant*, it selects the opposite city (see Fig. 5.1). The opposite city is determined by calculating the rank of the city selected by the *leading-ant* and assigning the city with the opposite rank to the *opposite-ant*.

In contrast, if the *opposite-ant* was located on a different city, then it will mimic the decision made by the *leading-ant*. The *opposite-ant* will select a city with the same rank as the one selected by the *leading-ant*. This is illustrated in Fig. 5.2. The rank of a city is based on the combination of pheromone content and heuristic on the edge connecting it to the current city. It is analoguous to the selection probability usually used by ants to select a city. It is important to note that the same rank does not necessarily mean the same city because as the construction progresses, the *leading-ant* and the *opposite-ant* will have visited different cities. This procedure is followed for the entire construction phase.

Through opposition, the *opposite-ant* diverges from its corresponding *leading-ant*, which helps guide the ants into different areas of the solution space. The approach also maintains a constant synchronous relationship between the two ants, which reduces the randomness of the selection process. The synchronous opposition algorithm is included in Table 5.1.

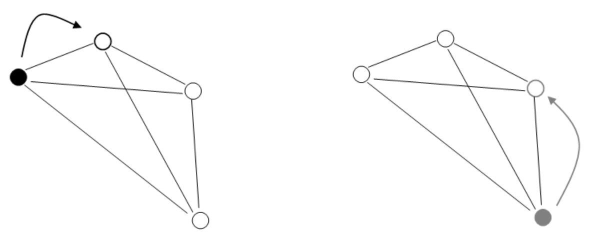

Figure 5.2: Synchronous Opposition: *leading-ant* (left) and *opposite-ant* (right) on different cities.

5.1.2 Free Opposition

The Free Opposition approach retains the opposite selection element from the Synchronous Opposition method. Thus, when the *opposite-ant* is located on the same city as the *leading-ant*, the *opposite-ant* will move to the opposite-ranking city. However, it relaxes its synchronicity aspect, so that if the two ants are not on the same city, the *opposite-ant* will select the next city using the pseudorandom rule (see Eq. 3.5). This method was implemented to examine the effect of removing the rigid synchronicity between the two ants. Nevertheless, since the ACS has a greedy selection process, the *leading-ant* and the *opposite-ant* will both often select the highest ranking city. Table 5.1 includes the pseudocode for the Free Opposition extension of the ACS algorithm.

5.1.3 Free Quasi-Opposition

The third synchronous algorithm is similar to the Free Opposition extension. This extension also has a relaxed synchronocity so that when the *leading-ant* and the *opposite-ant* are located on different cities, they will both use the pseudorandom rule (see Eq. (3.5)) to select their next city. In constrast, the selection of an opposite city is relaxed. Thus, in the case when the two ants are on the same city, the *opposite-ant* also uses the pseudorandom rule, except that the city that was selected by the *leading-ant* is removed from the possible choices. This extension tries to increase the exploration of the solution space by restricting some of the choices by the *opposite-ants*. However, in contrast to the two previous methods, the *opposite-ant* is still able to select highly ranked edges when it is on the same city as the *leading-ant*. Table 5.1 describes the Free Quasi-Opposition extension.

Table 5.1: Direct Modification Algorithms

```
Initialize pheromone matrix ($\tau = \tau_o$)
Repeat until termination condition is satisfied
  Repeat until solution is constructed (for each ant $k$)
    IF ant $k$ is leading-ant
      Pick next city $j$ using pseudorandom rule
    ELSE
      IF opposite-ant is on SAME city as leading-ant
        Synchronous Opposition: Pick opposite-rank city
        Free Opposition: Pick opposite-rank city
        Free Quasi-Opposition: Pseudorandom rule (exclude leading-ant city)
      ELSE
        Synchronous Opposition: Pick same-rank city
        Free Opposition: Pick next city $j$ using pseudorandom rule
        Free Quasi-Opposition: Pick next city $j$ using pseudorandom rule
    Apply local pheromone update
  If necessary, update best-so-far solution
  Apply best trail update
```

5.1.4 Opposite Pheromone per Node

The Opposite Pheromone per Node (OPN) extension to the ACS is a direct modification to the pheromone value used by the ants to make their selection. Basically, there is an *opposite rate*, $\breve{\lambda}_o$, that determines the rate at which opposite pheromone will be used in the construction step. Every time an ant k has to select a city from the available cities, the pheromone content used for its decision will depend on the value of a uniform random number, $\breve{\lambda}$. If $\breve{\lambda} < \breve{\lambda}_o$, then the ant selects its next city j using the opposite pheromone content, $\breve{\tau}$, as follows:

$$j = \begin{cases} \underset{l \in \mathrm{N}_i^k}{\operatorname{argmax}} \left\{ \breve{\tau}_{il} [\eta_{il}]^\beta \right\} & \text{if } q < q_o, \\ J & \text{otherwise,} \end{cases} \tag{5.1}$$

$$p_{ij}^k = \frac{[\breve{\tau}_{ij}][\eta_{ij}]^\beta}{\sum_{l \in \mathrm{N}_i^k} [\breve{\tau}_{il}][\eta_{il}]^\beta} \text{if } j \in N_i^k, \tag{5.2}$$

$$\breve{\tau} = \tau_o + \frac{1}{L_{bs}} - \tau. \tag{5.3}$$

The parameter τ_o represents the initial pheromone deposit and L_{bs} is the length of the best-so-far path. These values are used to determine the opposite pheromone content because they bound the possible pheromone deposit. Given the governing equations of the ACS, the pheromone content is bounded by the initial pheromone deposit and the global optimal value [8]. Furthermore, the pheromone of all the available edges will be modified. In the other case, when $\breve{\lambda} \geq \breve{\lambda}_o$, the ant will select the next city using the original pheromone content. Fig. 5.3 illustrates the selection when the opposite pheromone is used. The local update and best trail update steps were not modified.

This pheromone-centred extension differs from the *explorer ants* method proposed by Montgomery and Randall [15]. In their approach, only a small portion of the colony used the anti-pheromone. Additionally, these *explorer ants* always used the anti-pheromone in their selection. In the method proposed in this work, all ants have the opportunity to use the opposite, and the affected decisions vary from ant to ant and from iteration to iteration. Table 5.2 describes the OPN extension of the ACS.

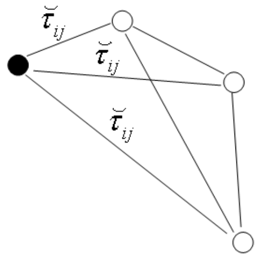

Figure 5.3: OPN: Pheromone content of all outgoing edges of a node depends on $\breve{\lambda}$.

Table 5.2: Opposite Pheromone per Node Algorithm for the TSP

```
Initialize pheromone matrix ($\tau = \tau_o$)
**Repeat** until termination condition is satisfied
  **Repeat** until solution is constructed (for each ant $k$):
    IF $\breve{\lambda} < \breve{\lambda}_o$
      Calculate opposite pheromone values, $\breve{\tau} = \tau_o + \frac{1}{L_{bs}} - \tau$
      Pick next city $j$ (pseudorandom rule with $\breve{\tau}$)
    ELSE
      Pick next city $j$ (pseudorandom rule with $\tau$)
    Apply local pheromone update on selected edge
  If necessary, update best-so-far solution
  Apply best trail update
```

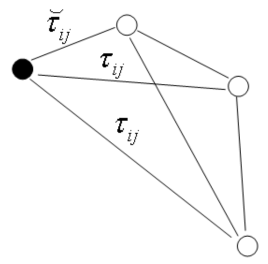

Figure 5.4: OPE: Pheromone content of individual edges j depends on $\breve{\lambda}_j$.

5.1.5 Opposite Pheromone per Edge

The second pheromone extension, Opposite Pheromone per Edge (OPE), is a modification of the OPN method. With this approach, the ants also have the possibility to use the opposite pheromone value to make their decision. However, the *opposite rate*, $\breve{\lambda}_o$, is applied to each individual edge of the decision instead of applying it to all the edges j connected to the current city i. This is depicted in Fig. 5.4. Table 5.3 describes the OPE extension of the ACS. The ants use the pseudorandom selection rule (see Eq. 3.5) to make their decision. The pheromone of each edge is determined by

$$\tau_{ij} = \begin{cases} \breve{\tau}_{ij} = \tau_o + \frac{1}{L_{bs}} - \tau_{ij} & \text{if } \breve{\lambda}_j < \breve{\lambda}_o, \\ \tau_{ij} & \text{otherwise.} \end{cases} \tag{5.4}$$

5.2 Update phase extension

Besides directly and indirectly modifying the construction phase, it was also important to experiment with the other type of OBL extension to the ant algorithm, namely modifying the update phase. Thus, a version of the algorithm by applying OBL to the update phase of the ant algorithm was also implemented.

5.2.1 Opposite Pheromone Update

The Opposite Pheromone Update (OPU) extension aims at performing additional updates using opposition information. The extension is applied during the best trail update phase

Table 5.3: Opposite Pheromone per Edge Algorithm

```
Initialize pheromone matrix ($\tau = \tau_o$)
**Repeat** until termination condition is satisfied
  **Repeat** until solution is constructed (for each ant $k$):
    FOR each available city $j$
      IF $\breve{\lambda} < \breve{\lambda}_o$
      Use opposite pheromone for edge from nodes $i$ to $j$, $\tau_{ij} = \tau_o + \frac{1}{L_{bs}} - \tau_{ij}$
    END
      Pick next city $j$ (using appropriate pheromone values $\tau_{ij}$)
    Apply local pheromone update on selected edge
  If necessary, update best-so-far solution
  Apply best trail update
```

of the ACS. As pheromone is added to the edges of the best path, a proportionally smaller amount is added to all the other edges. At every node, one outgoing edge will receive the best trail pheromone update. Then, an *opposition-rating*, $\breve{o}$, is calculated for all the other outgoing edges relative to the winning edge. This rating is used to determine the amount of pheromone to add to the other edge. It is calculated using the heuristic function values:

$$\breve{o}_{ij} = \frac{\left|\eta_{ij} - \eta_i^{bs}\right|}{\eta^{\max} - \eta^{\min}}, \tag{5.5}$$

where η_{ij} represents the heuristic function value for the edge going from city i to city j, and η_i^{bs} is the value for the edge outgoing from city i included present in the best path. The values $\eta^{\max}$ and $\eta^{\min}$ are the maximum and minimum heuristic values of the graph. They are used to normalize the *opposition-rating* of the edges. Once the *opposition-rating* is determined, the pheromone content of the edges that are not part of the best path is updated as follows:

$$\tau_{ij}^{new} = (1-\rho)\tau_{ij}^{current} + \rho(\Delta\tau_{ij}^{bs})\frac{(1-\breve{o}_{ij})}{W} \quad \forall (i,j) \notin T^{bs}, \tag{5.6}$$

where $\breve{o}_{ij}$ is the *opposition-rating* for the edge and W is a weight to modulate the effect of the additional pheromone. This parameter must be relatively high because the additional pheromone must not overpower the amount deposited for the best trail update. This additional pheromone is simply trying to guide the learning in the right direction. Thus, if the edge has a high $\breve{o}_{ij}$ (for instance, a long edge compared to a short edge in the best path), then $\Delta\tau_{ij}^{bs}$ will be multiplied by a smaller number. For edges that have a similar length to the selected edge, the factor will be higher. Note that the pheromone for the edges in the best path are updated normally (see Eq. 3.6).

In the ACS, only the best ant performs updates. Thus, only some edges benefit from the experience of the ants. Adding pheromone to all the edges helps extend the learning to the entire graph, while maintaining the relation to the best available path. This extension was devised with a pheromone increase because the ACS adds pheromone only to the best path. Removing pheromone would likely lead to extremely low levels of pheromone. This additional operation can be performed during the entire run or for a specific number of iterations. However, it is expected that it would be best if it was only performed for a certain number of iterations, as it mostly helps in the early stages of the algorithm. Table 5.4 summarizes the OPU method.

5.3 Experimental setup

All algorithms were compared to the ACS algorithm on 4 symmetric TSP instances of geographical nature, namely eil51, eil76, kroA100 and d198 [22]. Table 5.5 provides more details about each instance. All algorithms were coded in the C language based on the code developed by Stützle [28] except for Synchronous Opposition and Free Opposition, which were implemented in MATLAB. The algorithms solved the instances using real-valued distances.

The parameters of the algorithms were all set to the same values, namely $\beta = 2$, $p = 0.1$, $\xi = 0.1$, $m = 10$, and $q_o = 0.9$. These values were selected based on other research done using ACS and TSP [6, 15]. The algorithms completed 100 trials for the three smaller instances and 70 trials for the 198-city problem. Each trial was terminated after 5000 iterations or if the optimal solution was found.

Table 5.4: Opposite Pheromone Update Algorithm for the TSP

```
Initialize pheromone matrix ($\tau = \tau_o$)
**Repeat** until termination condition is satisfied
  Place $m$ ants on random squares
  **Repeat** until solution is constructed (for each ant $k$)
      Pick next direction $j$
      Apply local pheromone update (see Eq. (3.7))
  If necessary, update best-so-far solution
  Apply best trail update
  IF OBL condition is satisfied
      Calculate opposition-rating $\breve{o}$ for all edges
      Apply opposite pheromone addition
```

Table 5.5: Overview of the TSP Instances

Instance	#Cities	Optimal Tour (real-valued)
eil51	51	428.87
eil76	76	544.37
kroA100	100	21285.4
d198	198	15808.7

5.4 Results

The Wilcoxon rank sum (or Mann-Whitney) test was used to compare the results [10]. This test is a non-parametric alternative to the two-sample t-test. It compares two independent samples with a non-normal distribution to assess whether they come from a single distribution. This test performs a statistical comparison of the medians of two samples of unknown distribution. If the result of the test comparing two samples is significant ($p < 0.05$), one can accept the alternative hypothesis that there is a difference between the median of the two samples. A multiple comparison adjustment was not included because the comparisons were only done between two samples at once.

5.4.1 Results for direct modification of ant decision

Synchronous Opposition, Free Opposition and Free Quasi-Opposition are the three algorithms that extended the construction phase of the ACS by directly modifying ant decisions. The performance of each algorithm was evaluated in terms of the quality of the solution and the iteration when the best solution was found.

General results

Table 5.6 summarizes the accuracy results for the different algorithms. The median, minimum, and maximum of the final path length and number of times the optimal path was achieved are reported. Table 5.7 includes results on the iteration number when the final solution of the algorithm was found.

The median length of the solutions for the Synchronous Opposition and Free Opposition algorithms was statistically worse than that of the ACS. However, the Free Opposition approach was still able to find the optimal solutions for the three smaller TSP instances. When comparing the number of iterations needed to achieve their final solution, Synchronous Opposition took significantly more iterations for the 100-city problem and significantly less iterations for the 198-city problem. The Free Opposition method had significantly more iterations in both the 76-city and 100-city instances. These results seem to indicate that the two methods have difficulty converging, as they are unable to

Table 5.6: Accuracy Results of the Direct OBL Extensions for the TSP

Instance	Measure	ACS	SyncOpp	FreeOpp	FreeQuasiOpp
eil51	Median	429.53	435.34 †	433.70 †	430.24
	Min	428.87	428.87	428.87	428.87
	Max	441.09	449.65	446.54	445.81
	#Opt	7	0	1	5
eil76	Median	552.84	561.53 †	561.38 †	553.83
	Min	544.37	548.70	544.37	545.39
	Max	567.57	576.94	573.99	563.95
	#Opt	1	0	1	0
kroA100	Median	21428	21755.1 †	21708.8 †	21472.7
	Min	21285.4	21316.4	21285.4	21285.47
	Max	22455.4	22054.2	22685.3	22584.3
	#Opt	10	0	2	7
d198	Median	16141.98	16781 †	16712.5 †	**16127.3**
	Min	15901.1	16328.6	16266.1	15955.9
	Max	16442.6	17943.3	17523.2	17062.5
	#Opt	0	0	0	0

Bold values indicate a better result than the ACS algorithm.

† Difference with the ACS median is significant ($p < 0.05$).

Table 5.7: Median Final Iteration of the ACS and the Direct OBL Extensions

Instance	ACS	SyncOpp	FreeOpp	FreeQuasiOpp
eil51	2665.5	1901	2910	2181
eil76	3438	2986	3661 †	3582
kroA100	2454	3003 †	3816 †	2312
d198	4543	4343 †	4718	4569

† Difference with the ACS median is significant ($p < 0.05$).

find the optimal solution for the smaller city instances even with a larger number of iterations. When considering the 198-city problem, which is a more complex problem, the smaller number of iterations is an indication that the Synchronous Opposition algorithm has difficulty improving and remains trapped in a local optima.

The Free Quasi-Opposition extension to the ACS was more successful than the other two direct modification approaches. Its performance is equivalent to the normal ACS with no significant differences in their solution quality. There were also no significant differences in the number of iterations required to achieve the final solution. These results suggest that this extension did not have enough impact on the algorithm. However, the Free Quasi-Opposition algorithm was able to find a comparable number of optimal solutions.

Ant contribution in the algorithms

It was interesting to compare the level of contribution of the *leading-ants* and *opposite-ants*. The contribution was calculated as the number of times the ant's solution updated the best-so-far solution. Table 5.8 provides the relative contribution over all the trials. The results show that in the Synchronous Opposition and Free Opposition algorithms, the *opposite-ants* contribute less than 2% of the time to the update of the best solution. Thus, their only contribution is through the local update of pheromone. This may be one reason why the algorithms did not perform as well as expected; even when the *opposite-ants* are discovering new potential paths, they are not receiving any additional pheromone. Instead, the local update reduces the pheromone content to discourage other ants to take

Table 5.8: Contribution of the Opposite-Ants for the Direct OBL Algorithms (in %)

Instance	SyncOpp	FreeOpp	FreeQuasiOpp
eil51	0.972	1.760	48.287
eil76	0.274	0.462	45.713
kroA100	0.074	0.182	46.505
d198	0.294	1.441	45.610

those paths. When comparing Free Opposition with Synchronous Opposition, it seems that the reduced synchronicity in Free Opposition helps the *opposite-ants* find better solutions. This is also supported by the performance results. The Free Quasi-Opposition algorithm receives similar contribution from the two types of ants.

5.4.2 Results for indirect modification of ant decision

The OPN and OPE extensions also modify the construction phase of the ants, but they do so indirectly by altering the pheromone values used in the decision. The *opposite rate*, $\breve{\lambda}_o$, can be fixed or variable. The variable rates can be increasing or decreasing over time. The increasing rate is determined by

$$\breve{\lambda}_o = \breve{\lambda}_{final} \times \frac{n_I^{curr}}{n_I^{max}}, \tag{5.7}$$

and the decreasing rate is calculated as follows:

$$\breve{\lambda}_o = \breve{\lambda}_{init} \times \frac{n_I^{max} - n_I^{curr}}{n_I^{max}}. \tag{5.8}$$

In the case of the OPN algorithm, the *opposite rate*, $\breve{\lambda}_o$, was set to fixed rates 0.01, 0.05, 0.1 and 0.3, to variable rates increasing from 0 to 0.05 and 0.1 and to rates ecreasing from 0.05 and 0.1 to 0. For the OPE algorithm, the rates were fixed at 0.001, increased from 0 to 0.001 and 0.01, and decreased from 0.001 and 0.01 to 0.

Accuracy results

The accuracy performance of the algorithms was evaluated in terms of the median final path length, the median accuracy, the median accuracy difference with the ACS, the number of times the optimal solution was found, and the median total computational time. The accuracy was calculated by

$$A = 2 - (\frac{L^{bs}}{L^{opt}}) \times 100\%. \tag{5.9}$$

Additionally, the median accuracy difference between ACS and the OBL algorithms was quantified as follows:

$$\bar{A}_{diff}(\%) = \left(\frac{\bar{A}^{OBL}}{\bar{A}^{ACS}} - 1 \right) \times 100\%. \tag{5.10}$$

The median accuracy and the median total computational time for the proposed algorithms were compared to the ACS. The accuracy performance results for the OPN algorithm with fixed rates and variables rates are reported in Tables 5.9 and 5.10 respectively. The results for the OPE algorithm are summarized in Table 5.11.

In general, the OPN and the OPE algorithms achieved better results than the direct approches from the previous section. The OPN method with fixed $\breve{\lambda}_o$ achieved interesting results in all cases except for the 198-city instance and when $\breve{\lambda}_o = 0.3$.

When $\breve{\lambda}_o = 0.01$, the median accuracy was better than the ACS in all cases but the differences were not statistically significant. For the 76-city case, the improvement was slighty significant ($p < 0.055$). At $\breve{\lambda}_o = 0.05$, the accuracy results are the best. The algorithm performed better than the normal ACS for the three smaller instances and the median differences were statistically significant. For $\breve{\lambda}_o = 0.01$, only the 50- and 76-city instances had statistically better accuracy than the ACS. Finally, setting $\breve{\lambda}_o = 0.3$ resulted in worse results for all test instances. The three OPN algorithms with the lower $\breve{\lambda}_o$ values achieved a number of optimal solutions comparable to the ACS. Overall, OPN with $\breve{\lambda}_o = 0.05$ found the most optimal solutions (20). The computational time was however higher for all the OPN extensions.

The results indicate that the use of opposite pheromone content for some decisions can improve the accuracy of the solutions. It is noted that a very low value for the opposition-rate will improve the results but not significantly. The results did improve when the

Table 5.9: Accuracy Results of OPN for the TSP with Fixed $\breve{\lambda}_o$

Instance	Measure	ACS	OPN(0.01)	OPN(0.05)	OPN(0.1)	OPN(0.3)
eil51	Median	429.53	429.48	429.12 †	429.48 †	431.57 †
	$\bar{A}$	99.85	99.86	99.94	99.84	99.37
	$\bar{A}_{diff}$(%)	–	**0.011**	**0.096**	**0.011**	-0.476
	#Opt	7	3	4	2	1
	t(s)	4	4.4 †	5.4 †	6.2 †	10.1 †
eil76	Median	552.84	551.77	551.32 †	550.85 †	558.61 †
	$\bar{A}$	98.44	98.64	98.72	98.81	97.34
	$\bar{A}_{diff}$(%)	–	**0.20**	**0.28**	**0.372**	-1.07
	#Opt	1	2	4	3	0
	t(s)	7.7	8.4 †	10.3 †	12.453 †	21.3 †
kroA100	Median	21428	21400.3	21388.4 †	21408.4	22015.9 †
	$\bar{A}$	99.33	99.46	99.52	99.42	96.57
	$\bar{A}_{diff}$(%)	–	**0.13**	**0.19**	**0.092**	-2.78
	#Opt	10	11	12	11	0
	t(s)	12.4	13.8 †	16.8 †	20.6 †	35.4 †
d198	Median	16142	16103.3	16287.1 †	16718.5 †	17365.8 †
	$\bar{A}$	97.89	98.14	96.97	94.24	90.15
	$\bar{A}_{diff}$(%)	–	**0.25**	-0.94	-3.73	-7.9
	#Opt	0	0	0	0	0
	t(s)	48.0	52.5 †	65 †	80.0 †	141 †

Bold values indicate a better result than the ACS algorithm.

† Difference with the ACS median is significant ($p < 0.05$).

Table 5.10: Accuracy Results of OPN for the TSP with Increasing and Decreasing $\breve{\lambda}_o$

Instance	Measure	ACS	OPN (increasing $\breve{\lambda}_o$)		OPN (decreasing $\breve{\lambda}_o$)	
			(0→0.05)	(0→0.1)	(0.05→0)	(0.1→0)
eil51	Median	429.53	429.48	429.30 †	429.48 †	429.12 †
	$\bar{A}$	99.85	99.86	99.90	99.86	99.94
	$\bar{A}_{diff}(\%)$	–	**0.01**	**0.05**	**0.01**	**0.096**
	#Opt	7	5	5	4	8
	t(s)	4	4.7 †	5.2 †	4.8 †	5.3 †
eil76	Median	552.84	551.29 †	550.91 †	551.60 †	550.83 †
	$\bar{A}$	98.44	98.73	98.8	98.67	98.81
	$\bar{A}_{diff}(\%)$	–	**0.29**	**0.36**	**0.23**	**0.38**
	#Opt	1	1	1	9	2
	t(s)	7.7	9.1 †	10.3 †	9.3 †	10.4 †
kroA100	Median	21428	21393.2	21383.2 †	21390.8 †	21393 †
	$\bar{A}$	99.33	99.49	99.54	99.51	99.49
	$\bar{A}_{diff}(\%)$	–	**0.16**	**0.21**	**0.18**	**0.165**
	#Opt	10	12	14	9	12
	t(s)	12.4	14.8 †	16.7 †	15.0 †	16.9 †
d198	Median	16142	16221.4 †	16297.2 †	16163.2	16219.5 †
	$\bar{A}$	97.89	97.39	96.91	97.76	97.40
	$\bar{A}_{diff}(\%)$	–	-0.513	-1	-0.137	-0.5
	#Opt	0	0	0	0	0
	t(s)	48.0	56.7†	64.6†	57.1†	64.7†

Bold values indicate a better result than the ACS algorithm.

† Difference with the ACS median is significant ($p < 0.05$).

Table 5.11: Accuracy Results of OPE for the TSP

Instance	Measure	ACS	OPE	OPE(incr.$\breve{\lambda}_o$)		OPE (decr.$\breve{\lambda}_o$)
			($\breve{\lambda}_o = 0.001$)	(0→0.001)	(0→0.01)	(0.001→0)
eil51	Median	429.53	430.24	430.24	431.9†	430.24
	$\bar{A}$	98.85	98.68	99.63	99.30	99.63
	$\bar{A}_{diff}$(%)	–	-0.167	-0.208	-0.54	-0.167
	#Opt	7	5	4	3	4
	t(s)	4 †	8.2 †	10.5 †	10.7	12.1
eil76	Median	552.84	554.66 †	554.06	560.35 †	555.06 †
	$\bar{A}$	98.44	98.11	98.22	97.06	98.03
	$\bar{A}_{diff}$(%)	–	-0.33	-0.23	-1.40	-0.41
	#Opt	1	1	1	0	0
	t(s)	7.7	17.1 †	22.2 †	24.9 †	25.9 †
kroA100	Median	21428	21465.7	21452	21547.4 †	21415.5
	$\bar{A}$	99.33	99.15	99.24	98.77	99.39
	$\bar{A}_{diff}$(%)	–	-0.179	-0.087	-0.565	**0.059**
	#Opt	10	5	5	0	8
	t(s)	12.4	28.8 †	38 †	38.2 †	44.3 †
d198	Median	16142	16596.6 †	16272.4 †	17104.2 †	16230.7 †
	$\bar{A}$	97.89	95.01	97.07	91.81	97.33
	$\bar{A}_{diff}$(%)	–	-2.94	-0.84	-6.22	-0.57
	#Opt	0	0	0	0	0
	t(s)	48.0	113.3 †	148.3 †	149.7	173.4 †

Bold values indicate a better result than the ACS algorithm.

† Difference with the ACS median is significant ($p < 0.05$).

opposition-rate was increased from 0.01 to 0.05 and 0.1. However, the performance was not as good when $\breve{\lambda}_o = 0.1$, suggesting that increasing the opposition-rate too much results in a drop in accuracy. This is supported by the fact that the performance was lower when the opposition-rate was set to 0.3. Overall, there is an indication that $\breve{\lambda}_o = 0.05$ is a reasonable value. Nevertheless, it was important to investigate the effects of varying the opposition-rate during the optimization.

The results for OPN with variable rate were similar to OPN with fixed rate. Increasing the rate from 0 to 0.05 resulted in statistically higher accuracy for the 76-city case. The other two smaller instances also had lower median, but the difference was not statistically significant. Increasing the rate from 0 to 0.1 also led to better results for the smaller instances. The differences were statistically significant. The two OPN versions with decreasing rates led to accuracy that were statistically better than those achieved by the ACS for the three smaller instances.

For the 198-city instance, all OPN versions, except OPN decreasing from 0.05 to 0, led to statistically worse results. Finally, the number of optimal solutions achieved by the OPN extensions with variable rate was in general higher than the ACS. The two cases with decreasing rate had the overall highest number of optimal solutions, namely 22. Thus, it seems that the ant algorithm benefits from the use of opposite pheromone earlier in the optimization. Despite the use of linearly decreasing and increasing rates, the computational time was still higher than for the ACS. However, the OPN algorithm with variable rates was slighty faster compared to fixed rate.

When comparing the fixed rate and the variable rate OPN results for the 198-city case, the paths achieved by increasing to or decreasing from $\breve{\lambda}_o = 0.05$ were statistically better than the fixed $\breve{\lambda}_o$. In the case of the three smaller instances, increasing and decreasing the rate did not generate any statistical differences.

The OPE approach with a fixed rate had no significant difference in solution quality compared to the ACS algorithm for the 51- and 100-city instances. Its performance was statistically worse for the other two cases. This may suggest that the extension has a very small impact on the path construction phase. However, the OPE algorithm was still able to find optimal solutions in the three smaller instances.

Increasing the rate of OPE from 0 to 0.001 led to statistically worse solutions only in the 198-city case. The performance in the other instances was only slightly better than for the ACS. Increasing the rate from 0 to 0.01 led to significant worse solutions in all instances. This may also be an indication that 0.01 is too high for the OPE approach. Decreasing $\breve{\lambda}_o$ from 0.001 to 0 did not lead to improvements in any of the instances. It seems that the use of OPE is not beneficial for the ant algorithm.

Finally, the computational time was statistically higher for all the OPE versions.

Convergence results

To evaluate the convergence rate of the algorithms, a desired accuracy of 95% was set. The number of iterations needed to reach the accuracy was used as the convergence measure since total computational time has already been reported. A speed-up factor, S, was also defined, to compare the median number of iterations of ACS relative to the median number of iterations of the OBL algorithm:

$$S = \left(1 - \frac{\bar{n}_I^{OBL}}{\bar{n}_I^{ACS}}\right) \times 100\% \tag{5.11}$$

Tables 5.12 and 5.13 summarize the convergence results for OPN for fixed and variable $\breve{\lambda}_o$ respectively. Table 5.14 reports the results for the OPE algorithm.

Similar to the accuracy results, OPN with $\breve{\lambda}_o = 0.3$ required statistically more iterations than the ACS to achieve 95%. The other three OPN algorithms had all comparable results with the ACS in the three smaller instances. When $\breve{\lambda}_o = 0.05$ in the 51-city instance, the algorithm achieved a statistically significant improvement of 20%. In the 198-city case, the median number of iterations was 5000 for $\breve{\lambda}_o = 0.3$ and 0.1, meaning that the desired accuracy was not reached in most cases. Overall, the results indicate that fixed rate OPN has a lower convergence rate than the normal ACS.

Results for the OPN algorithms with variable rates show that decreasing the rate from 0.1 to 0 requires more iterations to achieve the desired accuracy. This version of OPN took statistically more iterations in the 76- and 198-city instances. The other three algorithms resulted in better performance than the ACS, but the differences were not statistically significant. When $\breve{\lambda}_o$ was decreased from 0.05 to zero, for the 198-city instance, the number of iterations was statistically higher. In sum, the algorithm provides a small speed-up for the smaller TSP instances, but with no statistically significant differences.

Table 5.12: Convergence of OPN for the TSP with Fixed $\breve{\lambda}_o$

Instance	Measure	ACS	OPN(0.01)	OPN(0.05)	OPN(0.1)	OPN(0.3)
eil51	$\bar{n}_I$	70.5	59.5	56 †	67.5	177 †
	$S(\%)$	–	**15.60**	**20.57** †	**4.26**	-151.1
eil76	$\bar{n}_I$	173.5	175	174.5	196.5	1165 †
	$S(\%)$	–	-0.864	-0.57	-13.2	-571
kroA100	$\bar{n}_I$	251	223	230	251	2223 †
	$S(\%)$	–	**11.16**	**8.37**	0	-785.4
d198	$\bar{n}_I$	966.5	900.5	2000 †	–	–
	$S(\%)$	–	**6.83**	-106.88	–	–

Bold values indicate a better result than the ACS algorithm.

† Difference with the ACS median is significant ($p < 0.05$).

Table 5.13: Convergence of OPN for TSP with Increasing and Decreasing $\breve{\lambda}_o$

Instance	Measure	ACS	OPN (increasing $\breve{\lambda}_o$)		OPN (decreasing $\breve{\lambda}_o$)	
			(0→0.05)	(0→0.1)	(0.05→0)	(0.1→0)
eil51	$\bar{n}_I$	70.5	56	53.5	63	63.5
	$S(\%)$	–	**20.57**	**24.11**	**10.64**	**9.93**
eil76	$\bar{n}_I$	173.5	154	169	169 †	256
	$S(\%)$	–	**11.24**	**2.59**	**2.59**	-47.84
kroA100	$\bar{n}_I$	251	232	229	212	270
	$S(\%)$	–	7.57	**8.76**	**15.54**	**7.57**
d198	$\bar{n}_I$	966.5	995.5	1015	1480 †	2920 †
	$S(\%)$	–	-3.0	-5.02	-53.1	-202

Bold values indicate a better result than the ACS algorithm.

† Difference with the ACS median is significant ($p < 0.05$).

Table 5.14: Convergence of OPE for the TSP

Instance	Measure	ACS	OPE (fixed $\breve{\lambda}_o$)	OPE(incr.$\breve{\lambda}_o$)		OPE (decr. $\breve{\lambda}_o$)
			(0.001)	(0→0.001)	(0→0.01)	(0.001→0)
eil51	$\bar{n}_I$	70.5	82	63.5	67	75
	$S(\%)$	–	-16.3	**9.93**	**4.96**	-6.38
eil76	$\bar{n}_I$	173.5	269 †	172	201.5	289 †
	$S(\%)$	–	-55	**0.864**	-16.1	-66.6
kroA100	$\bar{n}_I$	251	349 †	243.5	215	352.5 †
	$S(\%)$	–	-39.0	**2.99**	**14.3**	-40.4
d198	$\bar{n}_I$	966.5	4997 †	1075	5000 †	2888 †
	$S(\%)$	–	-417	-11.2	–	-198.8

Bold values indicate a better result than the ACS algorithm.

† Difference with the ACS median is significant ($p < 0.05$).

With a fixed rate, the OPE algorithm also had a significantly slower convergence in all instances except for the 51-city case. Increasing the rate from 0 to 0.001 led to faster convergence than the ACS for the three small instances, but the difference was not of statistical significance. This seems to indicate that linearly decreasing the rates can be beneficial. In the 198-city case the number of iterations was only slighty higher. Increasing the rate from 0 to 0.01 led to comparable results to the ACS for the three smaller instances, but the convergence rate was the worse for the 198-city case. The median was 5000, indicating that the desired accuracy was often not reached. This confirms that 0.01 may be too high for the more complex problems. Finally, decreasing from 0.001 to 0 led to significantly worse results in the three larger instances. However, one can notice that for the 198-city case, it was not one of the worse convergence rates. The lower convergence rates for the 198-city instance were achieved with the two linear variations of the 0.001 rate, which is consistent with the accuracy results. Overall, it seems that linearly increasing to the rate to 0.001 is the most beneficial extension.

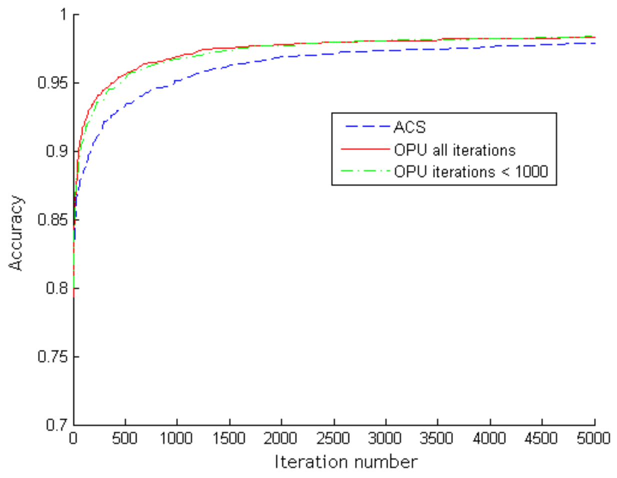

Figure 5.5: Accuracy results of ACS and OPU with $W = 1000$ for the 198-city TSP.

5.4.3 Results for modifying pheromone update

In contrast to the other five extensions, Opposite Pheromone Update (OPU) modifies the update phase of the ant algorithm. It changes the pheromone values after the construction. The algorithm was tested with four combinations of the two settings: 1) OPU applied at all iterations or for $n_I < 1000$ and 2) $W = 100$ or $W = 1000$. The accuracy and convergence performance of the algorithms was evaluated.

Accuracy results

Once again, the accuracy performance of the algorithms was evaluated in terms of the median final path length, the median accuracy relative to the optimal solution, the difference in median accuracy with the ACS, the number of times the optimal solution was reached and the median total computational time. Table 5.15 summarizes the results.

The results of the OPU approach were very interesting. In contrast to OPN, OPU did not affect the accuracy of the small instances, but had an impact on the performance of the 198-city instance. In the 198-city instance, the best results occur when $W = 1000$. Moreover, it did not matter if the updates were applied at every iteration or only for the first 1000 iterations. The OPU accuracy was statistically better in both cases. The only

Table 5.15: Accuracy Results of OPU for the TSP with Increasing and Decreasing $\breve{\lambda}_o$

Instance	Measure	ACS	OPU (∀ iter)		OPU (for iter≤1000)	
			$(W = 100)$	$(W = 1000)$	$(W = 100)$	$(W = 1000)$
eil51	Median	429.53	429.48	432.15 †	430.47	430.60
	$\bar{A}$	99.84	99.86	99.23	99.63	99.6
	$\bar{A}_{diff}(\%)$	–	**0.011**	-0.61	-0.22	-0.25
	#Opt	7	6	1	3	0
	t(s)	4	8.6 †	8.9 †	4.9 †	4.9 †
eil76	Median	552.84	552.81	555.0 †	553.58	552.7
	$\bar{A}$	98.44	98.45	98.04	98.31	98.47
	$\bar{A}_{diff}(\%)$	–	**0.006**	-0.406	-0.137	**0.027**
	#Opt	1	1	1	1	3
	t(s)	7.7	18.8 †	18.2 †	9.8 †	9.7 †
kroA100	Median	21428	21426.4	21478.1	21390.8	21410.7
	$\bar{A}$	99.33	99.34	99.1	99.51	99.41
	$\bar{A}_{diff}(\%)$	–	**0.008**	-0.237	**0.176**	**0.082**
	#Opt	10	6	2	11	9
	t(s)	12.4	34 †	34.1 †	16.7 †	16.8 †
d198	Median	16142	16647.3 †	16083.8 †	16196.8 †	16070.6 †
	$\bar{A}$	97.89	94.7	98.26	97.54	98.34
	$\bar{A}_{diff}(\%)$	–	-3.27	**0.376**	-0.354	**0.461**
	#Opt	0	0	0	0	0
	t(s)	48.0	136.7 †	136.3 †	65.9 †	65.6 †

Bold values indicate a better result than the ACS algorithm.

† Difference with the ACS median is significant ($p < 0.05$).

difference is that the computational time was greatly reduced when OPU application was limited. In contrast, when $W = 100$, the median accuracy was statistically worse compared to the ACS and to OPU with $W = 1000$.

For eil51 and eil76, the accuracy results were comparable to the ACS, except when OPU was applied at every iteration with $W = 1000$. In that case, the median of the TSP solution for the two instances was statistically larger ($p < 0.01$). In the 100-city instances, the differences were not significant, but it seems that better results were achieved with $W = 100$.

The results indicate that $W = 1000$ is an appropriate choice for larger TSP instances, but becomes detrimental as the number of cities decreases. In fact, for all three smaller instances, applying OPU at every iteration with $W = 1000$ leads to results that are statistically worse than the other three OPU cases. Moreover, for the 51- and 100-city instances, the number of optimal solutions achived when $W = 100$ is higher than when $W = 1000$. It appears that a lower weight might be necessary for smaller instances. Fig.5.5 depicts the performance for the 198-city instance for the two cases with $W = 1000$. The accuracy difference and the rapid convergence at the early stages of the optimization can be clearly identified.

The required computational time of the OPU extension was higher with statistical significance for all cases and for all the TSP instances.

Convergence results

Like in the previous experimental results, a desired accuracy of 95% was set in order to quantitatively evaluate the convergence rate of the algorithms. The results were compared using a speed-up factor (see Eq. 5.11) and the Wilcoxon rank sum test. Table 5.16 reports the median iteration and speed-up factor for the different algorithms.

Supporting the accuracy results, the 198-city instance had the best results when the $W = 1000$. The median number of iterations required by the OPU algorithm for both cases were significantly lower than for the ACS. When applied at every iteration, the OPU algorithm achieved a 60.5% speed-up. If applied only for the first 1000 iterations, the speed-up was 51.8%. It is important to realize that the algorithms are computationally slower, as reported in Table 5.15, but their operations lead to results in less iterations. When

Table 5.16: Convergence of OPU for the TSP

Instance	Measure	ACS	OPU ($\forall$ iter)		OPU (for iter$\leq$1000)	
			($W = 100$)	($W = 1000$)	($W = 100$)	($W = 1000$)
eil51	$\bar{n}_I$	70.5	58.5	122 †	52.5	112 †
	$S(\%)$	–	**17.0**	-73.0	**25.5**	-58.7
eil76	$\bar{n}_I$	173.5	164	190.5	184.5	187
	$S(\%)$	–	**5.48**	-9.8	-6.3	-7.78
kroA100	$\bar{n}_I$	251	225	292	234.5	263.5
	$S(\%)$	–	**10.1**	-16.3	**6.57**	-4.98
d198	$\bar{n}_I$	966.5	–	382 †	2318 †	466 †
	$S(\%)$	–	–	**60.5**	-139.9	**51.8**

Bold values indicate a better result than the ACS algorithm.

† Difference with the ACS median is significant ($p < 0.05$).

$W = 100$, for the 198-city case, the median iteration number was significantly higher.

Similar to the previously reported accuracy results, the three smaller instances had better results when $W = 100$. When $W = 1000$, the 51-city instance required significantly more iterations. As for the other instances, the results were not statistically significant, but one can notice a similar trend in the reported iterations. Moreover, while the difference is not statistically significant, the median number of iterations is usually lower than the ACS when $W = 100$ for the three small instances. This suggests that OPU may be beneficial for the smaller instances, but the parameter must be better adjusted.

Compared to the indirect OBL algorithms, it is interesting that OPN is able to improve the accuracy of the three smaller instances, but fails with the 198-city case. In contrast, OPU resulted in significantly better performance for the 198-city instance, but the performance was not as good for the lower instances. Further work is necessary to fully determine the benefits of this approach, the effects of the weight constant, and if opposite updates should be done in a different manner.

Chapter 6

Grid World Problem Experiments

To further assess the benefits of extending ACO with OBL, the OBL versions of the ant algorithm were tested with another application: the Grid World Problem (GWP). The GWP was selected for two main reasons. The first is that the GWP has been used as a benchmark problem for studies involving opposition-based Reinforcement Learning (RL) [25,26,31]. The results were successful with the GWP. It was interesting to see if the success extends to ACO. Secondly, it was important to use a different application than the TSP to generalize the study of the effects of opposition on ACO.

6.1 Description of the grid world problem

The GWP involves a $n \times n$ grid where one square is randomly selected as the goal. The problem is to determine the optimal movement policy for the entire grid (see Fig. 6.1). This means that a direction is assigned to each square of the grid so that, when an agent moves using this grid, it will reach the goal in the smallest number of steps. This is not a typical ACO problem, so the original algorithm had to be customized for this particular problem.

One difference between the implementation proposed in this thesis and the one used in RL is that, in the RL study, the rewards are given based on the location of the goal. In this study, the goal is unknown. Consequently, it was expected that computational time would be higher and accuracy lower than in those experiments.

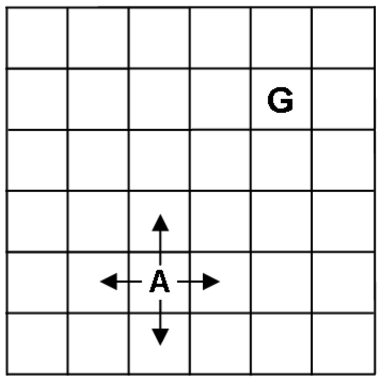

Figure 6.1: The Grid World Problem: the agent A moves in different directions to reach the goal G.

6.2 ACO applied to the GWP

Before extending the ant algorithm with OBL strategies, it was necessary to design an ant algorithm for solving the GWP. The framework of the algorithm is based on the Ant System (AS) version of the ant algorithm.

6.2.1 Representation

The GWP can be represented as a graph, where each square is a node and the ants travel between nodes. However, unlike the TSP, the graph is not fully connected and the ants do not need to pass through all the nodes to have a solution. Given a starting location, the ants only need to find the goal. The path completed by one ant only represents part of the final policy. The path found by an ant is the path that an agent would potentially take from the ant's starting location to the goal. The complete solution, or policy, is determined by the current pheromone matrix.

The pheromone content is associated with the edges connecting the squares in the grid, which is comparable to the edge connecting two cities in a TSP instance. Consequently, each square is associated with four pheromone levels.

6.2.2 Initialization

The initialization involves dropping an initial amount of pheromone throughout the network. However, the initial pheromone deposit, τ_o, will be higher than it is typically suggested for the AS. It was decided to use a higher starting pheromone rate, specifically 1, which is the maximum amount of pheromone that can be deposited by the ants (inverse of the path length). This was to encourage more exploration earlier in the algorithm, so that the ants do not focus too fast on a single direction.

6.2.3 Solution construction

At each iteration, m ants are randomly placed on m squares of the grid. Then, each ant moves from square to square until it finds the goal. The ants can move in four directions (up, down, left or right) to find the goal. The heuristic function was not included in the algorithm. The heuristic is usually used as a way to measure the cost of adding a component to the current path. However, since the location of the goal is unknown, it is not possible to estimate the quality of a decision. To make the problem more realistic, it was important to assume that there is no knowledge about the goal location. The ants select the next direction using the pseudorandom rule from the AS (refer to Eq. 3.5).

6.2.4 Pheromone update

Once the ants complete their path, the pheromone is evaporated and pheromone is deposited in the selected paths.

Evaporation

The evaporation for this algorithm was modified because of the nature of the GWP. In each iteration of the ant algorithm, ants will only visit a certain number of squares. Consequently, only a small number of squares will receive pheromone updates. In many cases, squares will not be visited for many iterations. Typical evaporation would therefore have a strong impact on the algorithm. Consequently, it was determined that the evaporation could not be applied to all the squares in the grid at each iteration because it would out-weight the pheromone deposits. In the proposed algorithm, the evaporation is applied only to squares that have been visited by the m ants during the iteration. The pheromone for a particular square is only evaporated once, even if it was visited by more than one ant. After some testing, the evaporation parameter, ρ, was set to 0.001.

Best trail update

Like in the AS, pheromone is deposited on the path travelled by the ants. Each square is associated with four pheromone quantities, one corresponding to each available direction. The pheromone is deposited on the direction selected by the ants:

$$\tau_{ij}^{new} = \tau_{ij}^{current} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \qquad i = 1, \ldots, n \times n; j = 1, \ldots, 4. \tag{6.1}$$

The amount deposited is calculated as per Eq. 3.4, namely the inverse of the length of the path. Another measure would have been to compare the final length with the optimal length, but that would require knowledge of the location of the goal. Here, the ants keep track of the number of steps they do until they find the goal, but they cannot spatially relate their starting point with the goal. Table 6.1 describes the AS algorithm for the GWP.

Table 6.1: GWP Algorithm

```
Initialize pheromone matrix ($\tau = \tau_o = 1$)
Randomly select goal
Repeat until termination condition is satisfied
  Place $m$ ants at random starting squares
  Repeat until goal is found (for each ant $k$):
    Pick direction $j \rightarrow$ next square
  Evaporate
  Apply best trail update
```

6.2.5 Policy calculation

The final policy is determined based on the pheromone content of each available direction for a square of the grid. The direction with most pheromone becomes the direction of the final policy. During the optimization, the ants provide pheromone information for each direction; they do not provide final solutions. In the TSP, ants will always generate a complete solution to the problem. In the GWP, the ants provide partial solutions, which help build the final pheromone matrix. Also, in the GWP, one pheromone matrix is associated with one policy, but in the case of the TSP, the same pheromone matrix can lead to multiple paths.

6.3 OBL extensions

Two different types of extensions to the AS algorithm were tested: applying OBL to the construction phase and to the update phase.

6.3.1 Opposite Pheromone per Node for the GWP

In this algorithm, the design is the same as for the OPN for the TSP. Every time an ant k has to select the next direction, the pheromone content used for its decision will depend on the value of a random number, $\breve{\lambda}$, and the *opposite-rate*, $\breve{\lambda}_o$. If $\breve{\lambda} < \breve{\lambda}_o$, then the ants will use the opposite pheromone content, $\breve{\tau}$. With the AS, the maximum and minimum pheromone contents are not bounded. Thus, the opposite is calculated using the maximum and minimum pheromone contents of the available directions. The maximum and minimum will vary depending on the current square. This method was designed to help the ants try different paths. Table 6.2 summarizes the OPN extension on the ACS for the GWP.

6.3.2 Opposite Pheromone Update for the GWP

The second algorithm focuses on the update of pheromone content. Additional updates are done on the opposite actions. When an ant completes its path by finding the goal,

Table 6.2: Opposite Pheromone per Node Algorithm for the GWP

```
Initialize pheromone matrix ($\tau = \tau_o = 1$)
Randomly select goal
**Repeat** until termination condition is satisfied
  Place $m$ ants on random squares
  **Repeat** until goal is found (for each ant $k$):
    IF $\breve{\lambda} < \breve{\lambda}_o$
      Calculate opposite pheromone values, $\breve{\tau} = \tau_o + \frac{1}{L_{bs}} - \tau$
      Pick next direction $j$
    ELSE
      Pick next direction $j$ (regular selection rule)
  Evaporate
  Apply best trail update
```

it adds pheromone to every decision along the path. In this extension, the ant will also remove pheromone from the opposite decisions along the path. For example, if, at square i, the ant choses to move "up", then, for that particular square, the pheromone for the "up" direction will increase and it will decrease by the same amount for the "down" direction. The amount of pheromone deposited is calculated using Eq. 3.4 using the length of the path. The following relation describes the update action:

$$\tau_{i\breve{j}} = \tau_{i\breve{j}} - \Delta\tau^k, \quad i = 1 \text{ to } n \times n. \tag{6.2}$$

In the OPU algorithm, the opposite update replaces the evaporation that occurs before the best trail update. Evaporation is used as a way to "forget" bad decisions. Removing pheromone from opposite actions can be a form of evaporation. Also, since the opposite actions will vary in the early iterations, the opposite update will be applied to different directions.

The opposite update can be used during the entire run of the algorithm or it can be used at a decreasing opposite-rate, $\breve{\lambda}_o$. In the case of the variable rate, the type of update

Table 6.3: Opposite Pheromone Update Algorithm for the GWP

```
Initialize pheromone matrix ($\tau = \tau_o = 1$)
Randomly select goal
**Repeat** until termination condition is satisfied
  Place $m$ ants on random squares
  **Repeat** until goal is found (for each ant $k$):
      Pick next direction $j$
    IF $\breve{\lambda} < \breve{\lambda}_o$
      Apply best trail update
      Apply *opposite* update
    ELSE
      Evaporate
      Apply best trail update
```

will depend on the value of a uniform random number, $\breve{\lambda}$. If $\breve{\lambda} < \breve{\lambda}_o$, the opposite update will occur, otherwise, regular evaporation is used. Table 6.3 describes the OPU extension on the ACS.

6.4 Experimental setup

Each algorithm was tested on three different grid sizes, namely 20×20, 50×50 and 100×100. They were implemented in the C programming language, inspired from code developed by Stützle [28]. The algorithms terminated after 10000 iterations. Each algorithm completed 100 trials on each grid set. The parameters of the different algorithms were set to the following values: $\alpha = 1$, $\rho = 0.001$, $\tau_o = 1$, and $m = 10$. In the case of the OPN algorithm, the *opposite rate*, $\breve{\lambda}_o$, was set to 0.01, 0.05 and a linearly decreasing rate from 0.01 to 0.

In the case of the OPU extension, the removal of pheromone was done in two different settings: 1) for the entire trial with $\breve{\lambda}_o = 1$ and 2) linearly decreasing from 1 to 0 (regular evaporation was used the rest of the time).

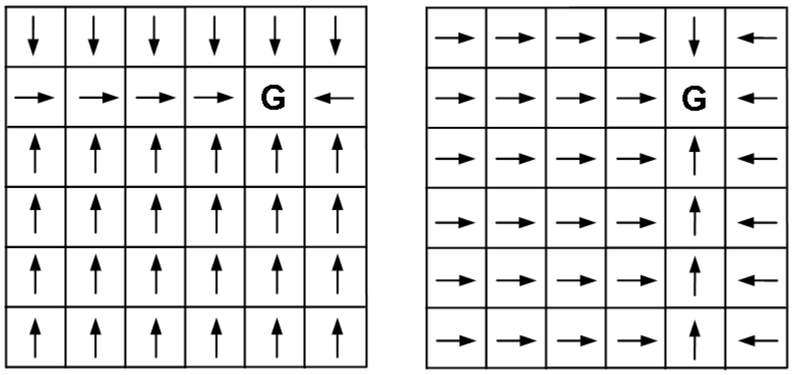

Figure 6.2: Two optimal policies for a 6 × 6 grid encompassing all possibilities.

6.5 Results

The perfomance of each algorithm was evaluated based on the accuracy of the final policy and computation time. The Wilcoxon rank sum (or Mann-Whitney) test was used to compare the medians of the results [10].

6.5.1 Accuracy results

The quality of the policies is determined by comparing them to an optimal policy. This performance measure, which was used in other work with GWP experiments [31], is defined as follows:

$$A_{\pi^*} = \frac{\|(\pi^* \cap \pi_1) \cup (\pi^* \cap \pi_2)\|}{n \times n}, \tag{6.3}$$

where π_1 and π_2 represent the two optimal possibilities for each square given a goal. Basically, for any problem, one can manually generate the possible solutions for each square that would be optimal. Fig. 6.2 illustrates manually generated policies for a 6 × 6 grid.

Like in the TSP experiments, the accuracy difference (in percent) between the AS and the OBL algorithms was also calculated. Tables 6.4 and 6.5 report the overall accuracy results for the OPN and OPU algorithms respectively.

Results show that the OPN algorithm was not as successful with the GWP as it was with the TSP instances. When $\breve{\lambda}_o = 0.01$, the accuracy for all three grid problems was comparable with the AS, but the differences were not statistically significant. With $\breve{\lambda}_o =$

Table 6.4: Accuracy Results of OPN for the GWP

Instance	Measure	AS	OPN(0.01)	OPN(0.05)	OPN(0.01 $\rightarrow$ 0)
20×20	$\bar{A}$	98.0	97.88	97.75	98.0
	$\bar{A}_{diff}(\%)$	–	-0.128	-0.255	0
	$t(s)$	6.06	6.13 †	6.39 †	6.33
50×50	$\bar{A}$	97.1	97.08	96.84 †	97.08
	$\bar{A}_{diff}(\%)$	–	-0.025	-0.268	-0.021
	$t(s)$	39.64	41.4 †	42.98 †	42.53 †
100×100	$\bar{A}$	93.2	93.25	92.575 †	93.22
	$\bar{A}_{diff}(\%)$	–	**0.048**	-0.676	**0.016**
	$t(s)$	190.5	204.3 †	210.3 †	209.3 †

Bold values indicate a better result than the AS algorithm.

† Difference with the AS median is significant ($p < 0.05$).

Table 6.5: Accuracy Results of OPU for the GWP

Instance	Measure	AS	OPU	OPU $\rightarrow$ evap
20×20	$\bar{A}$	98.0	98.25 †	98.5 †
	$\bar{A}_{diff}(\%)$	–	**0.255**	**0.510**
	t(s)	6.06	5.83 †	5.91 †
50×50	$\bar{A}$	97.1	97.84 †	98.0 †
	$\bar{A}_{diff}(\%)$	–	**0.762**	**0.927**
	t(s)	39.64	35.73 †	36.68 †
100×100	$\bar{A}$	93.2	96.67 †	96.165 †
	$\bar{A}_{diff}(\%)$	–	**3.71**	**3.176**
	t(s)	190.5	165.9 †	171.2 †

Bold values indicate a better result than the AS algorithm.

† Difference with the AS median is significant ($p < 0.05$).

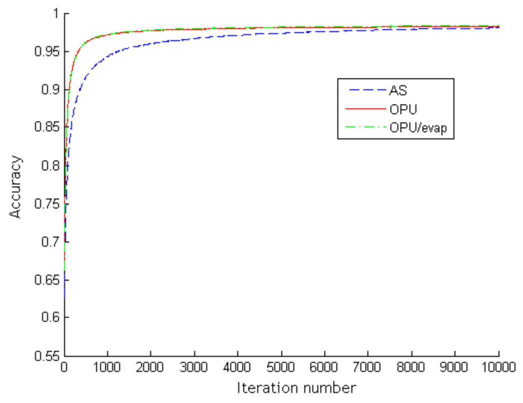

Figure 6.3: Accuracy per iteration for AS and OPU for the GWP (20×20).

0.05, the rate that had the most success in the TSP experiments, in the GWP, the accuracy was lower for all the cases with statistical difference in the two larger grids. Finally, linearly decreasing $\breve{\lambda}_o$ does not have a significant impact on performance. All three OPN implementations were also more computationally expensive.

In contrast to the OPN results, the OPU extensions performed very well. The version of OPU that applies the update at every iteration led to improved accuracy for all grid sizes. The difference is significant for the smaller size ($p < 0.05$) and very significant ($p < 0.01$) for the 50×50 and 100×100 grids. In the 100×100 grid case, the accuracy was improved by 3.7%, which is good considering that the base accuracy is already high. The second OPU alternative, where the rate of application linearly decreases over time, also led to accuracy improvements that were statistically significant. In the 100×100 case, the accuracy improved by 3.1%. The two OPU algorithms were less computationally expensive than the original AS. The differences were statistically significant for all of the cases ($p < 0.01$).

Figures 6.3, 6.4, and 6.5 depict the performance of the algorithms on the 20×20, 50×50 and 100×100 grids respectively. The accuracy difference and the rapid convergence at the early stages of the optimization can be clearly identified.

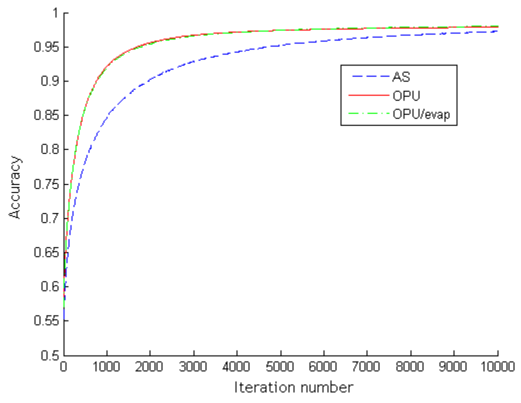

Figure 6.4: Accuracy per iteration for AS and OPU for the GWP (50×50).

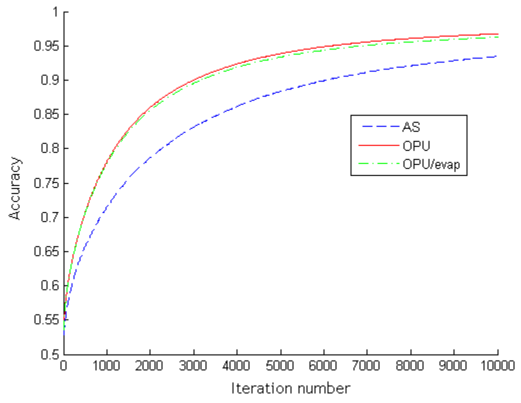

Figure 6.5: Accuracy per iteration for AS and OPU for the GWP (100×100).

Table 6.6: Convergence Results of OPN algorithms for the GWP

Instance	Measure	AS	OPN(0.01)	OPN(0.05)	OPN(0.01→0)
20×20	$\bar{n}_I$	321	349	347	323
	$S(\%)$	–	-8.72	-8.1	-0.62
50×50	$\bar{n}_I$	1885.5	1885.5	2056 †	1978
	$S(\%)$	–	0	-9.04	-4.91
100×100	$\bar{n}_I$	6155	6147	6778.5 †	6193
	$S(\%)$	–	**0.13**	-10.13	**0.62**

Bold values indicate a better result than the AS algorithm.

† Difference with the AS median is significant ($p < 0.05$).

6.5.2 Convergence results

It was interesting to establish a quantitative measure of convergence for the two OPU algorithms. In order to evaluate the convergence rate of the algorithms, a desired accuracy of 90% was set. The Wilcoxon test was used to statistically compare the median of the results. The speed-up factor, S, used in the TSP experiments (see Eq. 5.11), was also used as a comparative measure. Tables 6.6 and 6.7 summarize the convergence results for the OPN and OPU algorithms respectively.

The three OPN algorithms did not lead to improvements in convergence. The results were all comparable with no statistical significance, except for $\check{\lambda}_o = 0.05$. At that rate, the number of iterations was significantly higher for the two larger grid sets.

In contrast, the OPU algorithms were significantly faster than the AS. The OPU version that works at every iteration achieved a speed-up factor of 66%, 60% and 50% for the 20×20, 50×50, and 100×100 grids respectively. The linearly decreasing OPU achieved similar speed-up improvements of 64%, 59.5% and 48%. These results clearly demonstrate that the opposite updates are providing essential information to the ants, helping them achieve the optimal policy faster.

Table 6.7: Convergence Results of OPU for the GWP

Instance	Measure	AS	OPU	OPU/evap
20×20	$\bar{n}_I$	321	108.5 †	115.5 †
	$S(\%)$	–	**66.2**	**64.0**
50×50	$\bar{n}_I$	1885.5	755.5 †	763.0 †
	$S(\%)$	–	**59.9**	**59.5**
100×100	$\bar{n}_I$	6155	3048.5 †	3200.5 †
	$S(\%)$	–	**50.5**	**48.0**

Bold values indicate a better result than the AS algorithm.

† Difference with the AS median is significant ($p < 0.05$).

Chapter 7

Discussion

While the application of OBL to ACO can be challenging, in general, results indicate that the use of opposition can be beneficial. Specifically, results show that the OPN approach, namely using the opposite pheromone for some decisions was beneficial for the TSP. The OPU extension, which involved performing additional updates during the best trail update phase, led to excellent results for the GWP and some interesting performance improvements for the TSP. Nevertheless, there were performance differences among the OBL algorithms and the two applications. Results in both applications showed that OPN does not help with convergence rates. OPN did not lead to any significant convergence improvements. The only exception occurred for the 51-city TSP instance where OPN achieved a 20% speed-up in the number of iterations when the *opposite-rate* was fixed at 0.05.

The OPN method did not perform as well for the GWP. In the TSP, it was clear that the use of opposites was a contributing factor since a higher opposition rate improved the results. Additionally, linearly decreasing the rate at which opposite pheromone was used during the optimization also helped improve accuracy. In contrast, in the GWP, the use of opposite pheromone was not advantageous.

The OPU method applied to the GWP led to accuracy improvements in all grid sizes and convergence speed-ups reaching 66%. It was interesting to see that the performance improvements were relatively similar for all grid sizes. For the TSP, while great benefits came from OPN, it still failed in the larger instance. In contrast, OPU was not as successful with the small instances. However, for the 198-city instance, OPU achieved improvements in accuracy that were statistically significant and a speed-up of 60%. In general, it seems that OPU performs well with the more difficult problems, as it helps the learning process. These results are very encouraging, as they demonstrate that opposition can help improve performance.

The performance of the OPE extension applied to the TSP instances was comparable to that of the ACS, which may suggest that the algorithm does not have much impact. When OPE used a variable rate increasing to or decreasing from 0.001 the accuracy and convergence results seemed slightly better. Perhaps this is an indication that opposition should only be used in a manner that does not interfere with regular learning.

There are a few possible reasons for the differences between the TSP and GWP results. One of them may be that the GWP has less alternatives at each decision step. In the GWP, the ants only have up to four choices at each step, but in the TSP, the number of choices can go up to the total number of cities in the instance. Moreover, there is no middle choice among the alternatives in the GWP: some directions are perfectly correct and the others are incorrect. Thus, using opposite pheromone in the GWP will often lead the ants to pick a direction that is wrong. In contrast, in the TSP, the number of alternatives is much higher and the distinction between what is right or wrong is not as clear. An edge might be longer than the others, but it may still be part of the optimal solution. Consequently, in the TSP, the use of the opposite pheromone might lead to a better solution, as the ants may pick one edge that seems "wrong" and still achieve a good solution.

Another possible reason for the difference is that the TSP was solved using the ACS and the GWP optimization worked with the AS. These are two different ant algorithms based on different strategies. In AS, the solution of every individual ant receives a pheromone update, even if it is not the best solution. In the ACS, only the best-so-far solution is updated. Thus, with the AS, if using OPN led to a bad decision, it will still be somewhat reinforced. In contrast, with the ACS algorithm, only the best solution is reinforced.

Additionally, in the GWP, since the ant algorithm does not use a heuristic function, the pheromone content is the only decision parameter. Therefore, using opposite pheromone will often lead to the worse available solution. In the TSP, when the ants use the opposite pheromone, the heuristic is still available, which balances out the opposite pheromone.

One fundamental difference between the TSP and the GWP is that, in the GWP, the "opposite" is clearly defined. For each square in a grid, there are two sets of opposite pairs: up/down and left/right. Each direction has a unique opposite. Consequently, if one action is good, the bad action can be easily identified. In the TSP, a choice made by the ant at a certain node does not have a clearly defined opposite. Also, a straight mathematical opposite might not even be defined. Simply defining opposites with respect to the length of the edge might not make sense because, in some solutions, you need to take a longer edge to get an overall shorter path. In the GWP, the partial components of the solution are all the perfect components, which may be a reason why OPU, by removing pheromone in rejected directions, is very advantageous for the GWP. In the TSP, the algorithm makes local sacrifices for global success, which may explain why OPN might be helpful for the TSP.

Moreover, in the GWP, the path travelled by the ants from their start point to the goal is *unidirectional*. Thus, it is possible to define an "opposite" path that makes sense. This opposite path would include all the decisions that would bring the ants away from the goal. In the TSP, the solutions are *bidirectional*: going in the opposite direction of the path makes no difference in the final solution. Therefore, defining the "opposite" path is not as straightforward. The combinatorial aspect of the TSP complicates the definition of an opposite path. Changing a single component in the solution brings a new array of possibilities. The partial components of a solution are all dependent.

Another important difference is that in the GWP, the solutions achieved by the ants at the end of iterations are components of the final policy. The ants cover a small part of the grid. In the TSP, the ants are solving the entire problem in every iteration.

The speed-ups achieved with the use of opposite pheromone updates can be explained by the fact that the algorithm is rapidly moving toward the final optimal pheromone matrix. With usual pheromone updates, the algorithm takes very small steps moving towards the final pheromone matrix. In contrast, the opposite pheromone updates allow the algorithm to take very large guided jumps toward the optimal solution, by removing or adding more pheromone in the appropriate regions.

Chapter 8

Conclusions and Future Work

The work of investigating the application of opposition to ACO is just beginning. It would be very important to continue the exploration. Results are encouraging, especially for the OPN and OPU methods. Some fundamental opposition concepts, such as the use of opposite pheromone and performing opposite updates, led to encouraging results in the TSP and the GWP. Thus, opposition is a way that can provide benefits to ant algorithms, but more work is needed to fully develop the OBL framework for ACO. Further investigation of the OPU method will likely lead to excellent results. It is also important to attempt to combine the success of the different algorithms.

Consequently, it would be interesting to fully investigate the extent of the exploration done by each algorithm. While the OPN extension proved successful for the three small TSP instances, more work is required to determine all the benefits of this extension. The results show that pheromone content is a key element in solution creation. Thus, using more complex pheromone behaviour could lead to a better coverage of the search space. Additional investigations in new ways to vary the pheromone rate are also necessary. Results showed that a simple linear variation of the opposition rate led to improvements in the larger TSP instances. Future work should also involve the investigation of new ways of using the pheromone deposits.

Computational expense differences should also be evaluated. Most OBL extensions were computationally expensive, despite the success. Thus, it would definitely be important to optimize the algorithm performance .

Further work is be needed to explore the application of opposition to different versions of the ant algorithm, namely the Max-Min Ant System and the Best-Worst Ant System. Continuing the investigation with the ACS and the AS is also necessary so that performance differences can be clearly understood. It is also possible that applying opposition to ant algorithms will eventually generate a new form of the algorithm, which will be separate from the existing ACO frameworks. There should also be some experiments with the concept of opposition in combination with local search. It would be important to determine if the benefits of opposition complement those achieved through local search.

It was interesting to see that the results were not the same for the TSP and the GWP. While it is true that the GWP is not a direct ACO application, it helped reinforce some of the good results achieved with the TSP. Some of the differences might be attributed to the implementations and the different ACO versions. However, the application is what defines the algorithm that is used. Thus, future work should include more applications of ACO.

Another potential issue, as discussed in the OBL section of this work, was that, in the TSP, pheromone matrices lead to an array of possible solutions. There is no one-to-one relation between the pheromone matrix and a solution. Therefore, it might be important to establish rules on how to generate an actual opposite solution in a graph, so that there can be an exact fitness value. Additionally, it is important to establish how to compute the opposite pheromone matrix. The GWP was a little different from the TSP, in that the pheromone matrix was directly related to a solution, which may be one reason why OPU performed better with the GWP than with the TSP. This work explored opposite pheromone values and opposite updates; however, it did not create a direct relation between two pheromone matrices.

The most important work that needs to be developed is fundamental theoretical work with opposition and graph theory. While the GWP was an application that worked well with opposition, the true nature of ant algorithms are graphs like in the TSP. Thus, it is crucial to establish a strong theoretical base regarding opposition and graphs. As it has already been mentioned, opposition is not clearly defined in TSP, which springs from that fact that opposition is not clearly defined in graphs. Research has established opposite actions, opposite estimates, and opposite transfers functions. Perhaps, the next step is to establish the "opposite graph".

Bibliography

[1] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems.* New York: Oxford University Press, 1999.

[2] O. Cordón, I. F. de Viana, F. Herrera, and L. Moreno, "A New ACO Model Integrating Evolutionary Computation Concepts: The Best-Worst Ant System," in *Proc. of the 2nd Int. Workshop on Ant Algorithms (ANTS2000),* Brussels, Belgium, 2000, pp. 22-29.

[3] M. Dorigo, "Optimization, Learning and Natural Algorithms (in Italian)," PhD dissertation, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.

[4] M. Dorigo, and G. Di Caro, "Ant Colony Optimization: A New Meta-heuristic," in *Proc. of the 1999 Congress on Evolutionary Computation-CEC99,* New Jersey, 1999, vol. 2, pp. 1470-1477.

[5] M. Dorigo, V. Maniezzo, and A. Colorni, "The Ant System: Optimization by a Colony of Cooperating Agents," *IEEE Trans. Systems*, *Man*, and *Cybernetics*, vol. 26, pp. 29-41, 1996.

[6] M. Dorigo, and L. M. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem," *IEEE Transactions On Evolutionary Computation,* vol. 1, no. 1, pp. 53-66, 1997.

[7] M. Dorigo and T. Stützle, "The Ant Colony Optimization Metaheuristic: Algorithm, Applications, and Advances," *Handbook of Metaheuristics.* Fred Glover, Gary A. Kochenberger eds. Boston: Kluwer Academic Publishers, 2003, pp. 55-82.

[8] M. Dorigo and T. Stützle, *Ant Colony Optimization.* Cambridge, Massachusetts: The MIT Press, 2004.

[9] F. Glover and G. A. Kochenberger, Eds., *Handbook of Metaheuristics.* Kluwer Academic Publishers, 2003.

[10] M. Hollander, and D. A. Wolfe, *Nonparametric Statistical Methods.* Wiley, 1973.

[11] S. Iredi, D. Merkle, and M. Midderndorf, "Bi-Criterion Optimization with Multi Colony Ant Algorithms," in *Proc. First Int. Conf. on Evolutionary Multi-Criterion Optimization (EMO'01),* 2001, pp. 359-372.

[12] J. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm Intelligence.* San Mateo, CA: Morgan Kaufman, 2001.

[13] A. R. Malisia and H. R. Tizhoosh, "Image Thresholding Using Ant Colony Optimization," in *Proc. of the Third Canadian Conference on Computer and Robot Vision,* Qubec City, Canada, June 7th-9th, 2006.

[14] R. Michel and M. Middendorf, "An Island Model Based Ant System with Lookahead for the Shortest Supersequence Problem," in *Proc. of the 5th Int. Conf. on Parallel Problem Solving from Nature,* September 27-30, 1998, pp. 692-701.

[15] J. Montgomery and M. Randall, "Anti-Pheromone as a Tool for Better Exploration of Search Spaces," in *Proc. 3rd Int. Workshop on Ant Algorithms (ANTS2002),* Brussels, Belgium, September 2002, pp. 100-110.

[16] S. Ouadfel, M. Batouche and C. Garbay, "Ant Colony System for Image Segmentation Using Markov Random Field," in *Proc. 3rd Int. Workshop on Ant Algorithms (ANTS2002),* Brussels, Belgium, September 2002, pp. 294-295.

[17] S. Rahnamayan, H. R. Tizhoosh, and M. M. Salama, "Opposition-Based Differential Evolution Algorithms," in *Proc. IEEE Congress on Evolutionary Computation,* Vancouver, July 16-21, 2006, pp. 7363-7370.

[18] S. Rahnamayan, H. R. Tizhoosh, and M. M. Salama, "Opposition-based Differential Evolution Algorithms for Optimization of Noisy Problems," in *Proc. IEEE Congress on Evolutionary Computation,* Vancouver, July 16-21, 2006, pp. 6756-6763.

[19] S. Rahnamayan, H. R. Tizhoosh, and M. M. Salama, "A Novel Population Initialization Method for Accelerating Evolutionary Algorithms," *Computers and Mathematics with Applications,* vol. 53, no. 10, pp. 1605-1614, 2007.

[20] S. Rahnamayan, H. R. Tizhoosh and M. M. Salama, "Opposition-Based Differential Evolution (ODE) With Variable Jumping Rate," in *Proc. of IEEE Symposium on Foundations of Computational Intelligence (FOCI'07)*, Hawaii, April 1-5, 2007, pp. 81-88.

[21] M. Randall and J. Montgomery, "The Accumulated Experience Ant Colony for the Travelling Salesman Problem," in *Proc. of Inaugural Workshop on Artificial Life,* Adelaide, Australia, 2001, pp. 79-87.

[22] G. Reinelt, "TSPLIB - A traveling salesman problem library," *ORSA J. Comput.,* vol. 3, pp. 376-384, 1991.

[23] J. Schneider, "Swarm Intelligence: Power in Numbers," *Communications of the ACM,* vol. 45, no. 8, pp. 62-67, 2002.

[24] R. Schoonderwoerd, O. E. Holland, J.L. Bruten, and L.J.M. Rothkrantz, "Ant-Based Load Balancing in Telecommunications Networks," *Adaptive Behavior,* vol. 2, 1996, pp. 169-207.

[25] M. Shokri, H. R. Tizhoosh and M. S. Kamel, "Opposition-Based Qλ Algorithm," in *Proc. IEEE International Joint Conf. on Neural Networks (IJCNN),* Vancouver, July 16-21, 2006, pp. 646-653.

[26] M. Shokri, H. R. Tizhoosh and M. S. Kamel, "Opposition-Based Q(lambda) with Non-Markovian Update," in *Proc. IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007),* Hawaii, April 1-5, 2007, pp. 288-295.

[27] Y. Song and M. R. Irving, "Optimisation techniques for electrical power systems. II. Heuristic optimisation methods," *Power Engineering Journal,* vol. 15, no. 1, pp. 151-160, 2001.

[28] T. Stützle, Ant Colony Optimization, *Public Software,* June 14, 2004. http://iridia.ulb.ac.be/ mdorigo/ACO/aco-code/public-software.html

[29] T. Stützle and H. H. Hoos, "*MAX-MIN* Ant System," *Future Generation Computer Systems,* vol. 16, no. 8, pp. 889-914, 2000.

[30] R. Subrata and A. Y. Zomaya, "A Comparison of Three Artificial Life Techniques for Reporting Cell Planning in Mobile Computing," *IEEE Transactions On Parallel And Distributed Systems,* vol. 14, no. 2, pp. 142-153, February 2003.

[31] H. R. Tizhoosh, "Opposition-Based Learning: A New Scheme for Machine Intelligence," in *Proc. Int. Conf. on Computational Intelligence for Modelling Control and Automation - CIMCA'2005,* Vienna, Austria, 2005, vol. I, pp. 695-701.

[32] H. R. Tizhoosh, "Reinforcement Learning Based on Actions and Opposite Actions," in *Proc. Int. Conf. on Artificial Intelligence and Machine Learning,* 2005.

[33] H. R. Tizhoosh, "Opposition-Based Reinforcement Learning," *Journal of Advanced Computational Intelligence and Intelligence Informatics,* vol. 10, no. 4, pp. 578-585, 2006.

[34] M. Ventresca and H. R. Tizhoosh, "Improving the Convergence of Backpropagation by Opposite Transfer Functions," in *Proc. IEEE International Joint Conf. on Neural Networks (IJCNN),* Vancouver, July 16-21, 2006, pp. 9527-9534.

[35] M. Ventresca and H. R. Tizhoosh, "Opposite Transfer Functions and Backpropagation Through Time," in *Proc. IEEE Symposium on Foundations of Computational Intelligence (FOCI'07),* Hawaii, April 1-5, 2007, pp. 570-577.

[36] X. Zhuang, "Image feature extraction with the perceptual graph based on the ant colony system," in *Proc. of 2004 IEEE International Conference on Systems, Man and Cybernetics,* 2004, vol. 7, pp. 6354-6359.