7

Improving the Exploration Ability of Ant-Based Algorithms

Alice Ralickas Malisia

Department of Systems Design Engineering, Univeristy of Waterloo, Canada
amalisia@alumni.uwaterloo.ca

Summary. The chapter discusses the application of Opposition-Based Optimization (OBO) to ant algorithms. Ant Colony Optimization (ACO) is a powerful optimization technique that has been used to solve many complex problems. Despite its successes, ACO is not a perfect algorithm: it can remain trapped in local optima, miss a portion of the solution space or, in some cases, it can be slow to converge. Thus, we were motivated to improve the accuracy and convergence of the current algorithm by extending it with the concept of OBO. In the case of ACO, the application of opposition can be challenging because ACO usually optimizes using a graph representation of problems, where the opposite of solutions and partial components of the solutions are not clearly defined.

The chapter presents two types of opposition-based extensions to the ant algorithm. The first type, called Opposite Pheromone per Node (OPN), involves a modification to the construction phase of the algorithm which affects the decisions of the ants by altering the pheromone values used in the decision. Basically, there is an opposite rate that determines the frequency at which opposite pheromone will be used in the construction step. The second method, Opposite Pheromone Update (OPU), involves an extension to the update phase of the algorithm that performs additional updates to the pheromone content of opposite decisions. The opposition-based approaches were tested using the Travelling Salesman Problem (TSP) and the Grid World Problem (GWP).

Overall, the application of some fundamental opposition concepts led to encouraging results in the TSP and the GWP. OPN led to some accuracy improvements and OPU demonstrated significant speed-ups. However, further work is necessary to fully evaluate the benefits of opposition. Theoretical work involving the application of opposition to graphs is necessary, specifically in establishing the ‘opposite graph’.

7.1 Introduction

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, 14]. It is possible to have genetic cooperation, as it is the case with 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.

In addition, ACO is a population-based metaheuristic. Metaheuristics are procedures that use heuristics to seek a near-optimal solution with reasonable computation time [8, 9]. 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 [9, 29]. In brief, metaheuristics seek optimality while attempting to reduce computation time.

ACO is a powerful technique that has been used to solve many complex optimization problems, such as the travelling salesman problem [6, 7, 10], quadratic assignment problem [18], vehicle routing [2, 3, 11], and many others [9]. Given the complexity of these problems and their real-world applications, any improvement to the ant algorithm performance is encouraged. Specifically, an increase in accuracy and faster convergence are strongly welcomed. Ant algorithms can become trapped in a local optimum, miss a portion of the solution space or simply be slow to converge because the ants might take a long time to discover and learn to use the best paths. Thus, it is interesting to study and develop more complex behaviour for ant algorithms.

We attempt to improve the accuracy and convergence of ACO by extending the current algorithm with the concept of opposition-based optimization (OBO), which is a subclass opposition-based computing. This chapter provides a general overview how OBO can be applied to ACO. We will discuss two types of opposition-based extensions to the ant algorithm. The first type involves a modification to the construction phase of the algorithm which affects the decisions of the ants by altering the pheromone values used in the decision. This modification is called Opposite Pheromone per Node. The second type is an extension to the update phase of the algorithm that performs additional updates to the pheromone content of opposite decisions. The second extension is called Opposite Pheromone Update. The opposition-based approaches were tested on two different problems: the Travelling Salesman Problem (TSP) and the Grid World Problem (GWP).

The remaining of this chapter is organized as follows. Section 7.2 provides background information, including an overview of ant colony optimization and a review of the work that has been conducted to improve the performance of ant algorithms. Section 7.3 presents the two opposition-based approaches. Section 7.4 includes the experimental results of the opposition-based ant algorithms for the TSP and the GWP. Conclusions and future work are discussed in Sect. 7.5.

7.2 Background Information

This section will describe ACO, followed by a discussion on some of the pitfalls of the algorithm that we address using opposition. It also provides an overview of relevant work that has been conducted to improve the performance of the ant algorithm.

7.2.1 Ant Colony Optimization

The ant algorithm was introduced by M. Dorigo in 1992 [5]. It was developed to solve complex discrete combinatorial optimization problems. The first ACO algorithm was the Ant System (AS) [6], which was designed to solve the TSP.

Since the introduction of ACO, researchers have developed multiple versions to improve the performance of the AS. The extensions tend to focus more on the best solutions found to attempt to guide the ants search more effectively. The Ant Colony System (ACS) is a popular revised version of ACO [9] that achieved considerable accuracy improvements [7, 9]. Other extensions include the Best-Worst Ant System [4], the Max-Min Ant System [30], 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 [9].

The ACO algorithm is inspired from the natural behaviour of trail laying and following by ants [1, 9]. When exploring a region, ants are able to find the shortest path between their nest and a food source. This is possible because the ants communicate with each other indirectly via pheromone deposits they leave behind as they travel. The pheromone deposited by one ant influences the selection of the path by the other ants. A high pheromone concentration increases the probability that the path will be selected. The pheromone deposits work as a form of positive feedback, reinforcing good path choices and guiding the ants to better paths.

Ant System

When applied to an optimization problem, the ACO metaheuristic usually involves solution construction on a graph. The solutions are a path along the graph. The AS algorithm is summarized in Alg. 1. In line 3, the ants are distributed randomly among the nodes. Then, they 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 of the edge, represented by values in a $n \times n$ matrix where n is the number of nodes, and the value of the heuristic function of each edge. At each step of construction, ant k selects the next node using a probabilistic action choice rule which dictates the probability with which the ant will choose to go from current node i to next node j (see line 6):

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in \mathrm{N}_i^k} [\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}} \qquad \text{if } j \in N_i^k, \tag{7.1}$$

where τ_{ij} represents the pheromone content on the edge. Node j 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 and are connected to node i. The parameter η_{ij} represents the heuristic information. The heuristic value of an edge is a measure of the cost of extending the current partial solution with that edge (typically the inverse of the weight of the edge). The constants α and β represent the influence of pheromone content and heuristic information respectively. The stochastic component of the algorithm, namely probabilistically selecting a component, leads to exploration of a higher number of solutions.

Algorithm 1. Pseudocode of the Ant System

```
1: Initialize pheromone matrix ($\tau = \tau_o$)
2: while Termination condition is not satisfied do
3:     Place $m$ ants on random nodes
4:     for $k = 1$ to $m$ do
5:         if Solution construction for ant $k$ is NOT complete then
6:             Pick next node $j$ (see (7.1))
7:         end if
8:     end for
9:     Update pheromone matrix (evaporation and trail update) using (7.2) and (7.3)
10: end while
```

When all the ants have completed their paths the pheromone content is updated (see line 9). First, the pheromone content of the arcs, τ_{ij}, is evaporated based on the evaporation rate, ρ, following the relation

$$\tau_{ij}^{new} = (1 - \rho)\tau_{ij}^{current} \quad 0 < \rho < 1. \tag{7.2}$$

After the evaporation step, the solution of each ant is evaluated and pheromone is deposited on the ant's path relative to the quality of its solution. The ants deposit pheromone on the arcs they visited as follows:

$$\tau_{ij}^{new} = \tau_{ij}^{current} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \tag{7.3}$$

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

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{C_k} & \text{if arc is in the path of ant } k, \\ 0 & \text{otherwise,} \end{cases} \tag{7.4}$$

where C_k is the total cost of the solution for ant k. All arcs of one path will receive the same amount of pheromone (i.e. each ant deposits a constant amount of pheromone per edge).

Ant Colony System

Another commonly used version of ACO is the ACS. This version differs from the AS algorithm in three aspects [7, 9]: 1) different selection rule for path 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.

The general steps of the ACS algorithm are summarized in Alg. 2. When ants construct their paths in 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

$$j = \begin{cases} \underset{l \in \mathrm{N}_i^k}{\operatorname{argmax}} \left\{ \tau_{il} [\eta_{il}]^{\beta} \right\} & \text{if } q < q_o, \\ J & \text{otherwise.} \end{cases} \tag{7.5}$$

The parameter q is a uniform random number between 0 and 1 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 the node selected by the probabilistic action rule used in the AS (see (7.1)). This pseudorandom rule is a greedy selection approach that will tend to favour edges with a higher heuristic value and a high pheromone content.

The ACS has a pheromone update approach that exploits the best solutions found. The solutions found by the ants during the iteration are compared to the best solution found so far (best-so-far), and if one of the solutions is better, then the best-so-far solution is revised (see line 10). In line 11 the evaporation and deposit of pheromone is only applied to the arcs contained in the current best solution. The update is implemented by

$$\tau_{ij}^{new} = (1-\rho)\tau_{ij}^{current} + \rho(\Delta\tau_{ij}^{bs}) \quad \forall (i,j) \in T^{bs}, \tag{7.6}$$

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 [9]. The additional pheromone is calculated using the cost of the best-so-far path.

The ACS includes a local pheromone update to reduce emphasis on exploitation of existing solutions (see line 7). Immediately after an ant adds an arc to its current path the amount of pheromone on the arc is decreased as follows:

$$\tau_{ij}^{new} = (1-\xi)\tau_{ij}^{current} + \xi\tau_o \quad 0 < \xi < 1, \tag{7.7}$$

where τ_o is the initial amount of pheromone. The parameter ξ is the local evaporation rate, which controls the amount of pheromone that is removed. In the case of the TSP, research indicates that for the ACS, this value should be set to $\frac{1}{nC_{nn}}$, where n represents the number of cities and C_{nn} is cost of the nearest neighbour solution [9]. 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.

7.2.2 Challenges and Drawbacks

Despite being a powerful algorithm, ACO can benefit from performance improvements. Like other optimization techniques, ACO can remain trapped in a local optimum, miss a portion of the solution space or, in some cases, it can be slow to converge. ACO has many applications and deals with complex optimization problems, such as the travelling salesman problem [6, 7, 10], the quadratic assignment problem [18], vehicle routing [2, 3, 11] and many more [9]. Thus, any increase in speed of convergence and accuracy is beneficial.

Algorithm 2. Ant Colony System

```
1:  Initialize pheromone matrix ($\tau = \tau_o$)
2:  while Termination condition is not satisfied do
3:      Place $m$ ants on random nodes
4:      for $k = 1$ to $m$ do
5:          if Solution construction for ant $k$ is NOT complete then
6:              Pick next node $j$ (see (7.5))
7:              Apply local pheromone update using (7.7)
8:          end if
9:      end for
10:     If necessary, revise the overall best (best-so-far) solution
11:     Update pheromone matrix (trail update) according to (7.6)
12: end while
```

Given the fundamental structure of ACO, which involves reinforcement of good solutions, 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 the algorithm move away from a local optimum will likely lead to an increase in accuracy, are also welcome. Moreover, by moving away from a local optimum, one will increase exploration, which may also lead to improved solution quality.

Another interesting aspect of ACO is that it generates new solutions in every iteration using pheromone information. It is the progress and quality of the pheromone information that will affect the quality of the solutions. ACO can be slow to converge because sometimes it will take quite a number of iterations before the pheromone content of edges start having a strong impact on the ants decisions. Thus, one may improve the ant algorithm by making additional pheromone updates, which will help achieve accurate pheromone information faster.

7.2.3 Related Works

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 extension of the original ant algorithm that was developed to improve upon the performance of the AS [9]. The ACS has a greedy selection rule, but provides regular pheromone reduction as a measure to decrease desirability of edges once they are visited [7]. This attempts to prevent 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) [9, 30]. The MMAS strongly exploits the best tours found, but 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) [4], 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 bi-criterion optimization problems, Iredi et al. proposed a version of the ACS where two different pheromone trail matrices and two heuristic functions are considered simultaneously [13]. Randall and Montgomery proposed the Accumulated Experience Ant Colony as a method to determine the effect of each component on the overall solution quality [24]. 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 [26]. Montgomery and Randall developed three methods based on the concept of anti-pheromone as an attempt to capture complex pheromone behaviour [19]. In the first method, the pheromone content 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. On average, their methods produced better solutions on the smaller TSP problems (less than 200 cities).

Given these existing extensions, their results, and the potential for performance improvement, there was motivation to investigate the application of oppositional concepts to ant colony optimization. Moreover, opposition-based computing has already been successfully applied to reinforcement learning [27, 28, 31], evolutionary algorithms [20, 21, 22, 23] and neural networks [33, 34]. Opposition can potentially lead to a new way of developing more complex pheromone and path selection behaviour for ACO. The use of opposition is very interesting because it provides a structured way to investigate how to modify the ant algorithm.

7.3 Opposition and Ant Colony Optimization

The main idea of opposition-based optimization (OBO) is that by considering "opposites" one can increase the coverage of the solution space leading to increased accuracy and/or faster convergence [17, 20, 21, 22, 23, 31] (see also Chapter 2). OBO provides a general strategy that can be tailored to the technique of interest. Increasing the speed of convergence of the ant algorithm can be seen as a type-I opposition problem (Definitions 1 and 2 on page 14 in Chapter 2), because we can attempt to reach a better solution faster by using opposite guesses. It could also be addressed as a type-II opposition problem (Definitions 5 and 6 on page 17 in Chapter 2), because we can attempt to look for solutions that have opposite fitness, which would guarantee that we are moving to a better location. However, it is not obvious how to determine the solution that has the opposite fitness. Thus, we can use the type-I opposition as an approximation of type-II. An increase in accuracy can be achieved by using opposition to move away from a local optimum. Remaining trapped in a local optimum can be characterized as a type-I opposition problem since picking a guess that is opposite to the current guess will likely lead to a new area of the solution space. The following sections will dicuss the different ways that opposition can be applied to ACO and it will introduce two main opposition-based extensions to the ant algorithm: Opposition per Node and Opposite Pheromone Update.

7.3.1 Motivating Principles

In the case of ACO, the application of opposition is not as straightforward as mapping between an estimate and its opposite. Due to the graph structure of the problems, the opposite of a solution or of the partial components of the solutions are not clearly defined. Given the combinatorial aspect of solutions on a graph, even if only one element of a solution is changed, it leads to 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 because it might not lead to a connected path. It is not clear how one can generate an opposite solution, mainly because of the combinatorial aspect of the applications associated with ACO. To fit in the OBO scheme, the term *opposite* must be related to ant algorithms.

The concept of opposition serves as a starting point for the proposed extensions. The main idea is to think of opposition as a way of increasing the coverage of the solution space, which may lead to greater accuracy and faster convergence.

Ant algorithms do not work by modifying the existing solutions at each iteration; instead, 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 algorithm 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 of opposition has to involve the pheromone matrix.

However, it is not easy to define an opposite pheromone matrix because a pheromone matrix is not a point in the solution space. The pheromone matrix is indirectly related to the final solutions. Given 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 paths and, hence, there is no one-to-one relationship between a particular matrix and a single path. Thus, instead of focusing on an opposite pheromone matrix, another idea is to find a way to use opposition to move in the pheromone matrix solution space.

Consequently, it was determined that opposition could be applied to directly or indirectly affect the pheromone matrix. There are many different ways in which this can be achieved. For example, instead of initializing the pheromone content of all the edges to the same value, one could use opposition to determine a better initial value. However, while there can be many different places where opposition can be applied, we decided to concentrate on two main parts of the ACO algorithm: 1) the construction phase and/or 2) the update phase. These two phases were selected because, unlike initialization, they were present in every iteration of the algorithm. Thus, modifications and extensions to these phases will likely have a greater impact on the performance of the algorithm.

The construction phase can be modified by affecting the ant's decision. This can be done by altering the parameters used by the decision, namely the pheromone content. The modification to the update phase involves changing 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 [4], which uses the worst-ant to remove pheromone. However, there are other ways to affect the update phase. One way is to use opposite components of the current solutions without necessarily creating an opposite solution, which can be seen as identifying the opposite actions of the ants.

With these oppositional modifications, the algorithm is able to move to a new region of pheromone space. By changing the decision rule 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, which is useful to help the algorithm escape a local optimum. Moreover, this leads to exploration of a higher number of solutions because components with lower pheromone content can be selected. In contrast, in the case of opposition-based pheromone updates, the algorithm is actually moving to a new pheromone matrix, which 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. The ideas were used to design specific opposition-based algorithms which were tested with travelling salesman and the grid world problems. The next section will describe two successful extensions to the ant algorithm: Opposite Pheromone per Node and Opposite Pheromone Update. Detailed descriptions of these extensions, and of other less successful implementations can be found in [16].

7.3.2 Opposite Pheromone Per Node

The Opposite Pheromone per Node (OPN) is a direct modification of the pheromone value used by the ants to make their selection. It was designed to help the ants try different paths, and addresses the problem where the ant algorithm remains trapped in a local optimum. OPN attempts to move the ants out of their current paths by altering the pheromone they use in their decisions.

Algorithm 3 describes the OPN extension to ACS. The local pheromone update in line 12 and the best-trail update (line 16 are identical to the ones of the normal ACS algorithm. During the construction phase from line 4 to 14, the ants will move from node to node creating a solution until they have visited all nodes, achieved a specific number of steps or reached a goal.

Every time an ant k has to select the next node, the pheromone content used for its decision will depend on the value of a random number λ and the *opposite-rate*, $\breve{\lambda}$. The *opposite-rate*, $\breve{\lambda}_o$, determines the rate at which opposite pheromone will be used in the construction step of the algorithm. If $\lambda < \breve{\lambda}$ (line 6), then the algorithm calculates the opposite pheromone content for the edges in line 7 and the ant will use the opposite pheromone content, $\breve{\tau}$, to pick its next city (line 8). Otherwise, the ant will simply select the next city in line 10 using the original pheromone content.

The opposite pheromone content, $\breve{\tau}_{ij}$, for the edge connecting the current node i to an available node j is calculated as follows:

$$\breve{\tau}_{ij} = \tau_{min} + \tau_{max} - \tau_{ij}. \tag{7.8}$$

Algorithm 3. Opposite Pheromone per Node Algorithm

```
1: Initialize pheromone matrix ($\tau = \tau_o$)
2: while Termination condition is NOT satisfied do
3:    Place $m$ ants on random nodes
4:    for $k = 1$ to $m$ do
5:       if Solution construction for ant $k$ is NOT complete then
6:          if $\lambda < \breve{\lambda}$ then
7:             Calculate opposite pheromone values, $\breve{\tau} = \tau_{min} + \tau_{max} - \tau$
8:             Pick next node $j$ using pseudorandom rule (see (7.5)) with $\breve{\tau}$
9:          else
10:            Pick next node $j$ using pseudorandom rule (see (7.5)) with $\tau$
11:         end if
12:         Apply local pheromone update using (7.7)
13:      end if
14:   end for
15:   If necessary, revise the overall best (best-so-far) solution
16:   Update pheromone matrix (trail update) according to (7.6)
17: end while
```

The parameters τ_{min} and τ_{max} represent the minimum and maximum available pheromone contents, respectively. In the case of the ACS algorithm, τ_{min} can be the initial pheromone deposit and τ_{max} can be the inverse of the length of the best-so-far path, L_{bs}. These values can be used to determine the opposite pheromone content because, given the pheromone update equations of ACS (see (7.6) and (7.7)), the pheromone content is bounded by the initial pheromone deposit and the global optimal value [9]. With the AS, the maximum and minimum pheromone contents are not bounded. Thus, the opposite can be calculated using the maximum and minimum pheromone contents of the available edges or of the entire pheromone matrix.

7.3.3 Opposite Pheromone Update

The Opposite Pheromone Update (OPU) extends the pheromone update phase of the ant algorithm. This extension focuses more on the convergence issues existing in the ant algorithm. By adding or removing pheromone from opposite edges, the algorithm will modify the pheromone content faster than the ants normally would, which will speed up their learning. OPU performs additional updates using opposition information. When an ant completes its path, pheromone is added to every decision along the path, or in the case of ACS the edges of the best-so-far path. With the OPU extension, pheromone can also be added or removed from opposite edges.

OPU has a standard framework, where opposite edges receive additional updates. However, the specific way to define an opposite edge and how to update the pheromone varies depeding on 1) the version of the ant algorithm that is used, and 2) the problem that is being solved. In the case of the ACS, where pheromone is only added to the best-so-far path, the pheromone levels will not be very high on the other edges. Thus, OPU might work better if pheromone is added to the opposite edges. In constrast, in

Algorithm 4. Opposite Pheromone Update Algorithm

```
1: Initialize pheromone matrix ($\tau = \tau_o$)
2: while Termination condition is NOT satisfied do
3:    Place $m$ ants on random nodes
4:    for $k = 1$ to $m$ do
5:       if Solution construction for ant $k$ is NOT complete then
6:          Pick next node $j$ using pseudorandom rule (see (7.5)) with $\tau$
7:       end if
8:    end for
9:    Update pheromone matrix (trail update) according to (7.3)
10:   if $\breve{\lambda} < \breve{\lambda}_o$ then
11:      Calculate opposition-rating $\breve{o}$ for all edges using (7.9)
12:      Apply opposite pheromone update (see 7.10)
13:   end if
14: end while
```

the case of the AS, it is probably best to remove pheromone because the AS algorithm deposits pheromone on all generated paths.

Algorithm 4 describes the OPU extension to the AS. The initialization, the termination conditions, and the pheromone trails update (line 9) are identical to the ones of the normal AS algorithm. Also, like in the AS framework, the ants will construct their solutions (line 4 to 8) until they have a complete solution (i.e. visited all nodes, visited a specific number of nodes or found a goal). The selection of the next nodes in line 6 also follows the AS framework.

After the pheromone trails update (line 9), the OPU algorithm will potentially perform an opposite update. The rate at which the opposite update occurs depends on the value of a random number λ and the *opposite-rate*, $\breve{\lambda}$. If $\lambda < \breve{\lambda}$ (line 10), the *opposition-rating* is calculated in line 11 and the OPU algorithm performs the opposite update in line 12. If $\breve{\lambda} = 1$, the opposite update is done in every iteration.

The *opposition-rating* in line 11 of the OPU algorithm is a way to evaluate the degree of opposition of other edges in the graph relative to the current solution. In the case of AS, there are multiple current solutions: the solution found by each ant. In the case of ACS, the current solution is the best-so-far solution. For a given current solution, at every node of that solution, one outgoing edge will receive the trail pheromone update. Then, the *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 edges.

There are different ways to evaluate the degree of opposition of an edge. For example, $\breve{o}$ can be calculated using the heuristic function values:

$$\breve{o}_{ij} = \frac{\left|\eta_{ij} - \eta_i^{bs}\right|}{\eta^{\max} - \eta^{\min}}, \tag{7.9}$$

where η_{ij} represents the heuristic function value for the edge going from node i to node j, and η_i^{bs} is the value for the edge outgoing from node 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. This calculation method was used in OPU experiments for the TSP which can be found in [16].

In problems where each edge has a clearly defined opposite, the *opposition-rating* is straightforward: $\breve{o} = 1$ for the opposite edge and $\breve{o} = 0$ for the other outgoing edges. For example, in a path finding problem where the ants must pick a direction to take at every node, the selected direction (or edge) and opposite-direction pairs are clear. If, at node i, the ant chooses to move "up", then for that particular node, the "down" direction is the opposite edge. Thus, the "down" direction will have an *opposition-rating* of 1, and the "left" and "right" directions will have a rating of 0.

Once the *opposition-rating* is determined in line 11, the pheromone content of the "opposite-rated" edges is updated in line 12. The additional pheromone can be added or removed depending on the type of ant algorithm. The opposite update is based on the equation used in the regular pheromone trail update. In the case of AS, the opposite update equation involves pheromone removal:

$$\tau_{ij}^{new} = \tau_{ij}^{current} - \sum_{k=1}^{m} \breve{o}_{ij} \Delta\tau^{k}, \tag{7.10}$$

where $\breve{o}_{ij}$ is the *opposition-rating* for the edge, $\tau_{ij}^{current}$ is the current pheromone level on the edge going from node i to node j, $\Delta\tau^{k}$ is the pheromone added to the path found by ant k (see 7.4). This equation can be modified depending on the ant algorithm used and the application. In some cases, the opposite pheromone value can be divided by a weight to reduce its impact and, instead of a removal, the opposite pheromone can be added. In OPU experiments for the TSP [16], the opposite pheromone was added and it was divided by a weight.

Finally, in the OPU implementation for the AS, where the opposite update involves a removal of pheromone, the opposite update can replace evaporation. Evaporation is used as a way to "forget" bad decisions and thus, removing pheromone from opposite edges is achieving the same goal as evaporation. Keeping both the evaporation and the opposite pheromone update is not necessary, especially if the amount of pheromone being removed is as high as $\Delta\tau^{k}$ (the pheromone added to edges of the path of ant k). In summary, OPU provides an additional opposite update that can be interpreted as an intelligent evaporation.

7.4 Experimental Evidence

This section will outline some of the experimental results of applying opposition to ACO. The OPN algorithm was tested with the Travelling Salesman Problem (TSP) and the OPU algorithm was tested using the Grid World Problem (GWP). The Wilcoxon rank sum (or Mann-Whitney) test was used to compare the results [12]. If the result of the test comparing the 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. Other experimental investigations can be found in [16].

7.4.1 OPN Experiment

The OPN algorithm was compared to the ACS algorithm on 9 symmetric TSP instances, namely att48, eil51, eil76, kroA100, pr124, ch150, d198, lin318 and pcb442 [25]. Table 7.1 provides more details about each instance.

Table 7.1. Overview of the TSP Instances

Instance	#Cities	Optimal Tour
att48	48	10628
eil51	51	426
eil76	76	538
kroA100	100	21282
pr124	124	59030
ch150	150	6528
d198	198	15780
lin318	318	42029
pcb442	442	50778

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 only once before returning to the starting point. Mathematically, the TSP involves finding the minimum cost path in a weighted graph, which is an NP-hard problem [15]. A particular TSP instance has a specific number of cities (nodes) and arc weights (typically the distance between the cities).

Experimental setup

The parameters of the ant algorithms were all set to the same values, namely $\beta = 2$, $\rho = 0.1$, $\xi = 0.1$, $m = 10$, and $q_o = 0.9$. These values were selected based on other research done using ACS and TSP [7, 19]. The algorithms completed 100 trials and each trial was terminated after 5000 iterations or if the optimal solution was found. For the OPN algorithm, the *opposite-rate*, $\breve{\lambda}_o$, was set to a fixed rate of 0.0005, 0.001, 0.05, and 0.1.

Experimental results

The accuracy of each algorithm was evaluated in terms of the median final path length, the median accuracy difference with the ACS, the mean and standard deviation of the final path length and the number of times the optimal solution was found. The Wilcoxon rank sum (or Mann-Whitney) test was used to compare the medians of the results [12].

The median accuracy difference between ACS and the OBO algorithms was quantified as follows:

$$\bar{A}_{diff}(\%) = \left(\frac{\bar{A}^{OPN}}{\bar{A}^{ACS}} - 1\right) \times 100\%. \tag{7.11}$$

where $\bar{A}^{OPN}$ and $\bar{A}^{ACS}$ are the median accuracies of the best path found for the OPN and ACS algorithms. The accuracy of final path found by each algorithm is determined by

$$A = 2 - (\frac{L^{bs}}{L^{opt}}) \times 100\%. \tag{7.12}$$

where L^{bs} is the length of the best path found by the algorithm and L^{opt} is the length of the optimal solution. The accuracy results are reported in Table 7.2.

The results show that the *opposite-rate*, $\breve{\lambda}_o$, is an important factor in the success of the OPN strategy. In the smaller instances, $\breve{\lambda}_o = 0.1$ provided the best results, but as the number of cities increased the *opposite-rate* had to decrease to achieve good results. A high *opposite-rate* becomes detrimental to a problem with higher number of cities because the use of opposite pheromone becomes too frequent during the path construction. OPN is meant to be a strategy to affect a small number of the decisions in the hopes of helping the ants move from a local optimum. The frequent use of opposite pheromone does not let the ants benefit from their learning.

OPN performed very well in instances with 150 cities and less. Given the appropriate $\breve{\lambda}_o$, OPN achieved statistically significant results in all the smaller instances. Moreover, for the instances where optimal solutions were found, the OPN was able to achieve a higher number of optimal solutions. The statistically significant improvements in accuracy ranged from 0.187% to 0.757%. Even if these improvements are below 1%, they are important because the accuracy achieved by the different algorithms is already very high. On problems d198 and lin318, OPN achieved a lower median path length than ACS, but the difference was not statistically significant. In general, the standard deviation from the mean final path length was lower for the OPN algorithm. OPN produced worse results than the ACS for the pcb442 problem. It can be seen that OPN helps improve the accuracy of the ACS, which suggests it is addressing the local optimum trap issue faced by the ant algorithm.

To evaluate the convergence rate of the algorithms, a desired accuracy of 85% was set for the two larger instances (lin318, pcb442) and 95% level of accuracy was set for the other instances. The number of iterations needed to reach the accuracy was used as the convergence measure. The total computational time in seconds is also 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 OBO algorithm:

$$S = \left(1 - \frac{\bar{n}_I^{OBL}}{\bar{n}_I^{ACS}}\right) \times 100\% \tag{7.13}$$

Table 7.3 summarizes the convergence results for OPN algorithms. Like the accuracy results, the convergence results support the idea that the performance of OPN depends on the *opposite-rate*. The OPN algorithm was able to achieve an increase in convergence rate in the instances with less than 200 cities. The increases ranged from 4.6% to 22.7%. In three of the problem instances, namely kroA100, pr125 and ch150 with speed-up factors of 22.1%, 22.7% and 19.3% respectively, the difference was statistically significant. The standard deviations of the means for the OPN algorithm were generally lower than for the ACS. Typically, the OPN that achieved the best convergence results had the lower standard deviations. OPN did not perform as well for the

Table 7.2. Median ($\bar{A}$), Mean (μ_A), and Standard Deviation (σ_A) of the Accuracy, Accuracy Difference ($\bar{A}_{diff}(\%)$) and Number of Optimal Solutions Found Comparing the AS and OPN for the TSP

Instance	Algorithm	Median		$\bar{A}_{diff}(\%)$	$\mu_A \pm \sigma_A$	#Opt
att48	ACS	10653		–	10682 ± 46	6
	OPN 0.005	10653		0	10687 ± 54	7
	OPN 0.01	10653		0	10674 ± 45	9
	OPN 0.05	10653		0	10670 ± 42	11
	OPN 0.1	10653		0	**10663** ± **36**	**15**
eil51	ACS	428		–	429.3 ± 2.9	7
	OPN 0.005	429.5		-0.354	430.3 ± 3.4	11
	OPN 0.01	428		0	429.6 ± 3.3	9
	OPN 0.05	428		0	428.5 ± 3.5	10
	OPN 0.1	**427**	†	**0.236**	**428.2** ± **2.3**	**16**
eil76	ACS	548		–	547.7 ± 5.2	1
	OPN 0.005	547		0.189	547.4 ± 5.0	2
	OPN 0.01	548		0	547.2 ± 5.2	5
	OPN 0.05	545	†	0.568	545.4 ± **4.9**	8
	OPN 0.1	**544**	†	**0.757**	**545.0** ± 5.1	**11**
kroA100	ACS	21423		–	21530 ± 238	3
	OPN 0.005	21460		-0.177	21542 ± 242	2
	OPN 0.01	21389	†	0.161	21471 ± 235	**14**
	OPN 0.05	**21383**	†	**0.187**	**21435** ± **179**	12
	OPN 0.1	21393	†	0.142	21474 ± 219	6
pr124	ACS	59385		–	59475 ± 439	4
	OPN 0.005	59185		0.342	59401 ± **425**	5
	OPN 0.01	59242		0.243	59512 ± 494	6
	OPN 0.05	**59087**	†	**0.508**	**59352** ± **425**	**12**
	OPN 0.1	59159		0.385	59431 ± 431	9
ch150	ACS	6641		–	6654 ± 67.8	0
	OPN 0.005	6643		-0.031	6656 ± 74	0
	OPN 0.01	6621	†	0.32	6636 ± 59.7	0
	OPN 0.05	**6601**	†	**0.631**	**6612** ± **49.7**	0
	OPN 0.1	6623		0.281	6638 ± 64.1	0
d198	ACS	16093		–	16113 ± 116.9	0
	OPN 0.005	**16076**		**0.11**	**16109** ± 155	0
	OPN 0.01	16105		-0.08	**16109** ± **103**	0
	OPN 0.05	16275	†	-1.18	16275 ± 142	0
	OPN 0.1	16690	†	-3.86	16695 ± 180	0
lin318	ACS	44426		–	44372 ± **495.3**	0
	OPN 0.005	**44230**		**0.496**	44353 ± 519	0
	OPN 0.01	44305		0.305	**44318** ± 519	0
	OPN 0.05	46296	†	-4.72	46402 ± 891	0
	OPN 0.1	49274	†	-12.2	49233 ± 1012	0
pcb442	ACS	**55684**		–	**55610** ± 982	0
	OPN 0.005	55955	†	-0.59	56100 ± 1233	0
	OPN 0.01	57519	†	-0.305	57656 ± 1467	0
	OPN 0.05	63253	†	-4.72	63301 ± 1249	0
	OPN 0.1	64833	†	-12.2	64848 ± **965**	0

Bold values indicate the best results.

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

Table 7.3. Median ($\bar{I}$), Mean (μ_I), and Standard Deviation (σ_I) of the Number of Iterations, Speed-up Factor (S), and Median Time ($t(s)$) to Reach Desired Accuracy Comparing the AS and OPN for the TSP

Instance	Algorithm	$\bar{I}$		$S(\%)$	$\mu_I \pm \sigma_I$	$t(s)$
att48	ACS	40.5		–	51.6 ± 45.4	0.031
A=95%	OPN 0.005	44.5		-9.88	56.4 ± 40	0.031
	OPN 0.01	40.0		1.23	49.9 ± 33.0	0.031
	OPN 0.05	32.5		19.8	42.0 ± 34.5	0.031
	OPN 0.1	**32.0**		**21**	**39.5** ± **29.5**	0.031
eil51	ACS	76.0		–	109.2 ± 103	0.054
A=95%	OPN 0.005	72.0		5.26	123.1 ± 150.5	**0.047**
	OPN 0.01	**66.0**		**13.2**	91.3 ± 81.6	**0.047**
	OPN 0.05	68.5		9.87	**80** ± **52.9**	0.062
	OPN 0.1	74.5		1.97	99.1 ± 95	0.078
eil76	ACS	173		–	310 ± 390	**0.203**
A=95%	OPN 0.005	204		-17.63	305 ± 369	0.235
	OPN 0.01	169		2.60	270 ± 492	0.204
	OPN 0.05	**165**		**4.62**	**219** ± **199**	0.265
	OPN 0.1	186		-7.51	266 ± 222	0.360
kroA100	ACS	238		–	453 ± 707	0.422
A=95%	OPN 0.005	231		2.94	363 ± 583	0.438
	OPN 0.01	**186**	†	**22.1**	324 ± 614	**0.360**
	OPN 0.05	200		16	**304** ± **371**	0.492
	OPN 0.1	284		-19.3	394 ± 380	0.891
pr124	ACS	64.0		–	74.4 ± 46.1	**0.172**
A=95%	OPN 0.005	66.0		-3.13	80.6 ± 62.3	0.187
	OPN 0.01	62.5		2.34	86.2 ± 81.1	0.180
	OPN 0.05	**49.5**	†	**22.7**	**61.7** ± **46**	0.188
	OPN 0.1	55.5		13.3	76.5 ± 65.9	0.266
ch150	ACS	468		–	791 ± 915	1.66
A=95%	OPN 0.005	471		-0.749	709 ± 872	1.77
	OPN 0.01	**378**	†	**19.3**	**490** ± 484	**1.48**
	OPN 0.05	379		19.0	542 ± **443**	1.95
	OPN 0.1	705	†	-50.7	960 ± 840	4.69
d198	ACS	979		–	1129 ± 725	5.42
A=95%	OPN 0.005	**916**		**6.44**	1095 ± 796	5.36
	OPN 0.01	1050		-7.25	1137 ± 599	6.45
	OPN 0.05	1925	†	-96.6	2145 ± 1011	15.8
	OPN 0.1	5000	†	-410.7	4650 ± 799	54.3
lin318	ACS	**672**		–	**706** ± 331	**9.34**
A=85%	OPN 0.005	700		-4.09	721 ± 329	10.3
	OPN 0.01	752		-11.9	802 ± **303**	11.6
	OPN 0.05	2107	†	-213.5	2297 ± 1044	44.2
	OPN 0.1	5000	†	-644	4770 ± 720	140.1
pcb442	ACS	**2399**		–	**2406** ± **762**	**67.9**
A=85%	OPN 0.005	2946	†	-22.8	2883 ± 971	88.5
	OPN 0.01	4081	†	-70.1	3977 ± 945	128.6
	OPN 0.05	–	†	–	–	–
	OPN 0.1	–	†	–	–	–

Bold values indicate the best result.
† Difference with the ACS median is significant ($p < 0.05$).

two larger instances. It is also important to note that the computational time (in seconds) for OPN are comparable to AS, and even below ACS in most of the smaller instances. This indicates that the speed-ups achieved do not have a high computational cost. Overall, the results indicate that fixed rate OPN has a faster convergence rate than the normal ACS.

7.4.2 OPU Experiment

The OPU algorithm was compared to the Ant System (AS) using the Grid World Problem (GWP) on three different grid sizes, namely 20×20, 50×50 and 100×100. The GWP involves a $n \times n$ grid where one square is randomly selected as the goal. 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. The GWP was selected because it has been previously used as a benchmark problem for studies involving opposition-based Reinforcement Learning (RL) [27, 28, 31].

We adapted the AS algorithm to solve the GWP. In our implementation, the location of the goal is unknown to the ants until they reach it. The ants start in a random square in the grid and travel until they reach the goal, which means that in one iteration they may not travel on every square of the grid. Each square is associated with four pheromone contents, one for each available direction. Also, since ants do not visit every square in every iteration, the evaporation was only applied to the squares visited by the ants. Complete details of the actual implementation can be found in [16].

Experimental Setup

The algorithms were terminated after 10000 iterations. Each algorithm completed 100 trials on each grid set. The parameters of the ant algorithms were set to the following values: $\alpha = 1$, $\rho = 0.001$, $\tau_o = 1$, and $m = 10$. These parameters were selected based on some general experimentation. The initial pheromone value (τ_o) was set to the high value of 1 to encourage more exploration in the early stages of the algorithm, so that the ants do not focus too fast on a single direction.

In the case of the OPU extension the removal of pheromone was done on every iteration ($\breve{\lambda}_o = 1$). Also, since the GWP has clearly defined opposites, the edges that are true opposites have an *opposition-rating* of 1 and the other have a rating of 0. For example, if, at square (node) i, the ant chose to move up, then for that particular square (node) the up direction is part of the current solution, the down direction will have an *opposition-rating* of 1, and the left and right directions will have a rating of 0.

Experimental Results

The perfomance of each algorithm was evaluated based on the accuracy of the final policy and the convergence rate of the algorithm. The Wilcoxon rank sum (or Mann-Whitney) test was used to compare the medians of the results [12]. The accuracy or quality of the policies is determined by comparing them to an optimal policy. This

Table 7.4. Median ($\bar{A}$), Mean (μ_A), and Standard Deviation (σ_A) of the Accuracy Comparing the AS and OPU for the GWP

Instance	Algorithm	$\bar{A}$	$\mu_A \pm \sigma_A$
20×20	AS	98.00	98.05 ± 0.818
	OPU	98.25 †	98.26 ± 0.801
50×50	AS	97.10	97.23 ± 0.612
	OPU	97.84 †	97.87 ± 0.487
100×100	AS	93.21	93.4 ± 0.7235
	OPU	96.67 †	96.74 ± 0.465

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

Table 7.5. Median ($\bar{I}$), Mean (μ_I), and Standard Deviation (σ_I) of the Number of Iterations, Speed-up Factor (S), and Median Time ($t(s)$) to Reach a 90% Accuracy Comparing the AS and OPU for the GWP

Instance	Algorithm	$\bar{I}$	$S(\%)$	$\mu_I \pm \sigma_I$	$t(s)$
20×20	AS	321	–	343.6 ± 113.9	0.312
	OPU	108.5 †	66.2	114.8 ± 25.4	0.109
50×50	AS	1885.5	–	1885.1 ± 212.1	10.45
	OPU	755.5 †	59.9	752.4 ± 62.7	4.28
100×100	AS	6155	–	6018.2 ± 563.4	135.3
	OPU	3048.5 †	50.5	2995.4 ± 197.3	72.1

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

accuracy calculation, which was used in other work with GWP experiments [31], is defined as follows:

$$A_{\pi^*} = \frac{\|(\pi^* \cap \pi_1) \cup (\pi^* \cap \pi_2)\|}{n \times n}, \tag{7.14}$$

where π^* is the policy being evaluated and π_1 and π_2 represent the two optimal possibilities for each square given a goal. Table 7.4 reports the overall accuracy results including the median, mean and standard deviation of the accuracy.

The OPU extension performed very well. OPU improved the accuracy for all grid sizes. The difference of the medians is statistically 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 median accuracy was improved by 3.7%, which is good considering that the base accuracy is already above 90%. Moreover, comparing to the AS, the OPU mean accuracies are all higher and their standard deviations are all lower.

In order to evaluate the convergence rate of the algorithms, a desired accuracy of 90% was set. The median, mean and standard deviation of the number of iterations to reach the desired accuracy were used as comparative measures. The Wilcoxon test was used to statistically compare the median number of iterations. The speed-up factor (see (7.13)) and the computational time to reach the accuracy are also reported. Table 7.5 summarizes the convergence results for the AS and OPU algorithms.

The OPU algorithm was significantly faster than the AS. It achieved a speed-up factor of 66%, 60% and 50% for the 20×20, 50×50, and 100×100 grids, respectively. The mean number of iterations and their standard deviation were also lower. The lower standard deviations indicate that the OPU will reliably reach the 90% accuracy with fewer number of iterations that the AS. It is also important to note that the computational time (in seconds) for the OPU are also below the AS, which shows that the speed-up achieved does not have a high computational cost.

7.4.3 Discussion

While the application of opposition to ACO can be challenging, in general, results indicate that the use of opposition can be beneficial. Specifically, the OPN approach, using opposite pheromone for some decisions was beneficial for improving the accuracy for the TSP. The OPU extension, which involved performing additional updates during the best trail update phase, led to excellent results for the GWP. 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.

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. 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 is helpful for the TSP.

Moreover, in the GWP, the path travelled by the ants from their starting 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 and so problem-type dependent. 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.

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.

7.5 Conclusions and Future Work

The work of investigating the application of opposition to ACO is just beginning. The use of 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 OBO framework for ACO.

While the OPN extension proved successful for the smaller TSP instances, more work is required to determine all the benefits of this extension. The results show that *opposite-rate*, $\breve{\lambda}_o$, is a key element in the success of the OPN algorithm. Thus, selecting the rate wisely can lead to a better accuracy at a faster rate. Additional investigations in new ways to vary the pheromone rate are necessary.

Further work is 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.

While it is true that the GWP is not a typical ACO problem, it helped reinforce some of the good results achieved with the TSP. Some of the differences might be attributed to the implementations, the use of different ACO versions and different opposition algorithms. However, the problem is what defines the algorithm that is used. Thus, future work should include more applications of ACO.

Another potential issue is 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 is 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 well with the GWP. 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 [27, 28, 31], opposite estimates [20, 21, 22], and opposite transfers functions [33, 34]. Perhaps, the next step is to establish the “opposite graph”.

References

1. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, New York (1999)
2. Bullnheimer, B., Hartl, R.F., Strauss, C.: Applying the Ant System to the Vehicle Routing Problem. In: Osman, I.H., Voβ, S., Martello, S., Roucairol, C. (eds.) Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, pp. 109–120. Kluwer Academic Publishers, Dordrecht (1998)
3. Bullnheimer, B., Hartl, R.F., Strauss, C.: An Improved Ant System Algorithm for the Vehicle Routing Problem. Ann. Oper. Res. 89, 319–328 (1999)
4. Cordón, O., de Viana, I.F., Herrera, F., Moreno, L.: A New ACO Model Integrating Evolutionary Computation Concepts: The Best-Worst Ant System. In: Proc. of the 2nd Int. Workshop on Ant Algorithms (ANTS 2000), Brussels, Belgium, pp. 22–29 (2000)
5. Dorigo, M.: Optimization, Learning and Natural Algorithms (in Italian). PhD dissertation, Dipartimento di Elettronica, Politecnico di Milano, Italy (1992)
6. Dorigo, M., Maniezzo, V., Colorni, A.: The Ant System: Optimization by a Colony of Cooperating Agents. SMC 26, 29–41 (1996)
7. Dorigo, M., Gambardella, L.M.: Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transactions On Evolutionary Computation 1(1), 53–66 (1997)
8. Dorigo, M., Stützle, T.: The Ant Colony Optimization Metaheuristic: Algorithm, Applications, and Advances. In: Glover, F., Kochenberger, G.A. (eds.) Handbook of Metaheuristics, pp. 55–82. Kluwer Academic Publishers, Boston (2003)
9. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge (2004)
10. Gambardella, L.M., Dorigo, M.: Solving Symmetric and Asymmetric TSPs by Ant Colonies. In: Proc. IEEE Int. Conf. on Evolutionary Computation (ICEC 1996), Nagoya, Japan, pp. 622–627 (1996)
11. Gambardella, L.M., Taillard, E., Agazzi, G.: MACS-VRPTW: A multiple Ant Colony System for Vehicle Routing Problems with Time Windows. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization, pp. 63–76. McGraw-Hill, New York (1999)
12. Hollander, M., Wolfe, D.A.: Nonparametric Statistical Methods. Wiley, Chichester (1973)
13. Iredi, S., Merkle, D., Midderndorf, M.: Bi-Criterion Optimization with Multi Colony Ant Algorithms. In: Zitzler, E., Deb, K., Thiele, L., Coello Coello, C.A., Corne, D.W. (eds.) EMO 2001. LNCS, vol. 1993, pp. 359–372. Springer, Heidelberg (2001)
14. Kennedy, J., Eberhart, R.C., Shi, Y.: Swarm Intelligence. Morgan Kaufmann, San Mateo (2001)
15. Lawler, E.L., Lenstra, J.K., Rinnooy-Kan, A.H.G., Shmoys, D.B.: The Travelling Salesman Problem. Wiley, New York (1985)
16. Malisia, A.R.: Investigating the Application of Opposition-Based Ideas to Ant Algorithm. MASc. thesis, University of Waterloo, ON, Canada (2007), `http://hdl.handle.net/10012/3233`
17. Malisia, A.R., Tizhoosh, H.R.: Applying Opposition-Based Ideas to the Ant Colony System. In: Proc. of IEEE Swarm Intelligence Symposium, Honolulu, HI, April 1-5, pp. 182–189 (2007)
18. Maniezzo, V., Colorni, A.: The Ant System Applied to the Quadratic Assignment Problem. IEEE Trans. Knowl. Data Eng. 11(5), 769–778 (1999)
19. Montgomery, J., Randall, M.: Anti-Pheromone as a Tool for Better Exploration of Search Spaces. In: Proc. 3rd Int. Workshop on Ant Algorithms (ANTS 2002), Brussels, Belgium, pp. 100–110 (September 2002)

20. Rahnamayan, S., Tizhoosh, H.R., Salama, M.M.: Opposition-Based Differential Evolution Algorithms. In: Proc. IEEE Congress on Evolutionary Computation, Vancouver, July 16-21, pp. 7363–7370 (2006)
21. Rahnamayan, S., Tizhoosh, H.R., Salama, M.M.: A Novel Population Initialization Method for Accelerating Evolutionary Algorithms. Computers and Mathematics with Applications 53(10), 1605–1614 (2007)
22. Rahnamayan, S., Tizhoosh, H.R., Salama, M.M.: Opposition-Based Differential Evolution (ODE) With Variable Jumping Rate. In: Proc. of IEEE Symposium on Foundations of Computational Intelligence (FOCI 2007), Hawaii, April 1-5, pp. 81–88 (2007)
23. Rahnamayan, S., Tizhoosh, H.R., Salama, M.M.A.: Opposition-Based Differential Evolution. IEEE Transactions of Evolutionary Computation (in press, 2008)
24. Randall, M., Montgomery, J.: The Accumulated Experience Ant Colony for the Travelling Salesman Problem. In: Proc. of Inaugural Workshop on Artificial Life, Adelaide, Australia, pp. 79–87 (2001)
25. Reinelt, G.: TSPLIB - A traveling salesman problem library. ORSA J. Comput. 3, 376–384 (1991)
26. Schoonderwoerd, R., Holland, O.E., Bruten, J.L., Rothkrantz, L.J.M.: Ant-Based Load Balancing in Telecommunications Networks. Adaptive Behavior 2, 169–207 (1996)
27. Shokri, M., Tizhoosh, H.R., Kamel, M.S.: Opposition-Based Q(λ) Algorithm. In: Proc. IEEE International Joint Conf. on Neural Networks (IJCNN), Vancouver, July 16-21, pp. 646–653 (2006)
28. Shokri, M., Tizhoosh, H.R., Kamel, M.S.: Opposition-Based Q(λ) with Non-Markovian Update. In: Proc. IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007), Hawaii, April 1-5, pp. 288–295 (2007)
29. Song, Y., Irving, M.R.: Optimisation techniques for electrical power systems. II. Heuristic optimisation methods. Power Engineering Journal 15(1), 151–160 (2001)
30. Stützle, T., Hoos, H.H.: MAX-MIN Ant System. Future Generation Computer Systems 16(8), 889–914 (2000)
31. Tizhoosh, H.R.: 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, vol. I, pp. 695–701 (2005)
32. Tizhoosh, H.R.: Opposition-Based Reinforcement Learning. Journal of Advanced Computational Intelligence and Intelligence Informatics 10(4), 578–585 (2006)
33. Ventresca, M., Tizhoosh, H.R.: Improving the Convergence of Backpropagation by Opposite Transfer Functions. In: Proc. IEEE International Joint Conf. on Neural Networks (IJCNN), Vancouver, July 16-21, pp. 9527–9534 (2006)
34. Ventresca, M., Tizhoosh, H.R.: Opposite Transfer Functions and Backpropagation Through Time. In: Proc. IEEE Symposium on Foundations of Computational Intelligence (FOCI 2007), Hawaii, April 1-5, pp. 570–577 (2007)