2021

Investigation of Ant Colony Optimization Implementation Strategies For Low-Memory Operating Environments

Douglas Hale

Investigation of Ant Colony Optimization Implementation Strategies For Low-Memory Operating Environments

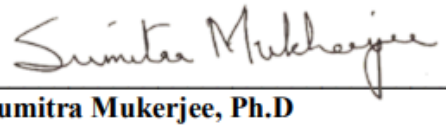
by

Douglas Hale

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in
Computer Science

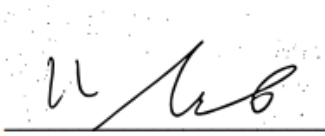
College of Computing and Engineering
Nova Southeastern University
2021

We hereby certify that this dissertation, submitted by Douglas Hale conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.



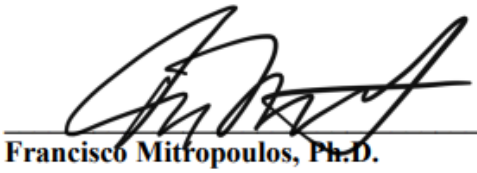
7/6/21

Sumitra Mukerjee, Ph.D **Date**
Chairperson of Dissertation Committee



7/6/21

Michael J. Laszlo, Ph.D. **Date**
Dissertation Committee Member



7/6/21

Francisco Mitropoulos, Ph.D. **Date**
Dissertation Committee Member

Approved:

7/6/21

Meline Kevorkian, Ed.D. **Date**
Dean, College of Computing and Engineering

College of Computing and Engineering
Nova Southeastern University

2021

Abstract

In search guided by *meta-heuristics*, a fundamental tradeoff exists between *exploitation* of good solutions (*intensification*) and *exploration* of the solution space for better solutions (*diversification*). Over-exploitation can limit the search to suboptimal solutions while over-exploration can reduce the efficiency of the overall search. Ant Colony Optimization (ACO) is a well-known meta-heuristic inspired by biological ants for solving NP-hard combinatorial search problems like the Traveling Salesman Problem (TSP). In nature, biological ant colonies navigate to find efficient paths around complex obstacles by depositing and following simple chemicals called pheromones. ACO algorithms model this behavior by implementing sets of artificial ants which iteratively explore a solution space, marking high quality solutions with "pheromone" values and biasing subsequent searches toward those values.

TSP serves as a benchmark for evaluating ACO algorithms. Current implementations of ACO algorithms require the storage of distances and pheromone levels between all pairs of vertices. This dissertation presents a variant that has significantly lower storage requirements, thus rendering it practical for use on larger problem instances. The tradeoff between memory use and solution quality is empirically investigated using benchmark TSP problem instances. MAX-MIN Ant System, one of the best available ACO variants, serves as a baseline method for comparison.

The variant of ACO algorithm proposed in this dissertation provides memory savings in terms of both distances stored and pheromone levels stored. Instead of maintaining distances between all pairs of vertices, distances between each vertex its k nearest neighbors are maintained; k is chosen to be much smaller than the number of vertices in the graph. An efficient approximate algorithm is used to identify nearest neighbors.

Observing that pheromone values between most pairs of vertices lie within a narrow range in an iteration, the ACO variant proposed in this study specifies a default pheromone value and uses a sparse matrix that only maintains values between a pair of vertices if it differs from the default value by more than a specified tolerance limit; the default pheromone level is updated in each iteration. The memory requirement for pheromone storage thus decreases with increasing tolerance limit, potentially at the cost of solution quality.

Experimental results on benchmark TSP instances indicate that with a judicious choice of tolerance level, the proposed approach consistently achieves solution quality that is comparable to those obtained by a MAX-MIN Ant System while using a fraction of the memory. Specifically, experimental ACO solution quality reached ranged between 10% and 25% of the standard solution quality and in the two of the largest TSP instances *exceeded* standard solution quality by 6% and 8%. Experimental memory usage showed a 95% to 99% reduction in required memory over the standard ACO algorithm.

Acknowledgments

I would like to extend special appreciation to my dissertation adviser, Dr. Mukherjee, for his sustained support and contributions throughout the dissertation process. Additionally, my dissertation committee member, Dr. Laszlo, provided instrumental feedback in helping me refine the scope and approach of this project into a successful research product. Indeed, these two professors truly exemplify the outstanding quality of the entire Nova Southeastern University College of Computing and Engineering faculty.

Finally, I would like to thank my wife, Cynthia. Without her tireless love and support for me during this challenging endeavor, I would never have crossed the finish line. She, more than any other, made this achievement possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Background

NP-hard stochastic combinatorial search problems are known to become intractable as input scales, under the widely accepted assumption that $P \neq NP$. Given the wide variety of important applications of NP-hard problems, *heuristics* are used instead to find good suboptimal solutions under certain constraints such as runtime or resource utilization limits. Most heuristic algorithms are parameterized such that the parameter values introduce a spectrum of possible solutions. On one side of the spectrum are better solutions closer to global optimality which tend to consume more computing resources. On the other are less optimal solutions but which tend to consume fewer resources. Thus, a heuristic can produce any variety of solutions along this spectrum. A *meta-heuristic* is an algorithm that manages these parameters to produce the best possible solutions given computation resource constraints. Every meta-heuristic at some level balances the *exploitation* of good solutions and the *exploration* of the solution space for better solutions. The terms *exploitation* and *exploration* are often synonymized with the terms *intensification* and *diversification* respectively. Over-exploitation can limit the search to suboptimal solutions while over-exploration can reduce search efficiency (Blum & Roli, 2003).

This dissertation is based on the *Ant System* (AS) works by (Dorigo, Maniezzo, & Colorni, 1996) and (Colorni, Dorigo, & Maniezzo, 1991) which evolved into a metaheuristic called *Ant Colony Optimization* (ACO). ACO is an approach to the

fundamental problem of stochastic combinatorial optimization inspired by mechanisms used by biological ants to search for food efficiently around complex obstacles using simple chemical trails to coordinate and optimize their path finding. Artificial ants are modeled as indirectly communicating agents to explore a search space. Three key features of this method listed in the original work are its (1) applicability to variations of a problem such as both asymmetric and symmetric traveling salesman (versatility), (2) applicability to different combinatorial problems such as scheduling (robustness), and (3) positive feedback basis for solution improvement (population based).

AS variants have been introduced as improvements over AS such as Elitist Ant System (EAS), Rank-Based Ant System (AS_{rank}), *MAX-MIN* Ant System (*MM*AS), and Ant Colony System (ACS) have been measured by their performance on various standard *Traveling Salesman Problem* (TSP) instances. The simplicity and easy intuition of the TSP problem statement and its practical applicability make TSP instances the most common testbed for evaluating ACO variants. Early AS works used well-known TSP instances including kroA100, d198, and rat783, and oliver-30 from the TSPLIB suite of TSP benchmarks to evaluate the algorithm performance (Dorigo, Maniezzo & Colorni, 1996).

AS describes a proof-of-concept optimization system based on biological ant foraging behavior called *stigmergy*: the "stimulation of workers by the performance they have achieved" (Colorni, Dorigo, & Maniezzo, 1991). While AS was not competitive with state-of-the-art TSP methods, the application of AS to the TSP showed how AS-based algorithms could be used to attack hard search problems. Over the course of the next decade, AS variants were developed which modified the mechanisms of pheromone

deposition, evaporation, and global update. Of the variants, one of the most successful is *MM*AS (Dorigo & Stützle, 2004).

*MM*AS is one of the best performing and most extensively studied AS variants (Dorigo & Stützle, 2004). High level *ACO* pseudo-code common to all AS variants, including *MM*AS, is depicted in *Figure 1.2.*

```
procedure ACOforTSP
        InitializeData
        while (not terminate) do
                ConstructSolutions
                (LocalSearch)
                UpdateStatistics
                UpdatePheromoneTrails
        end-while
end-procedure
procedure InitializeData
        ReadInstance
        ComputeDistances
        ComputeNearestNeighborLists
        ComputeChoiceInformation
        InitializeAnts
        InitializeParameters
        InitializeStatistics
end-procedure
```

Figure 1: ACO pseudo-code (Dorigo & Stützle, 2004).

Problem Statement

A limitation with all ACO algorithms, including *MM*AS, is that several very large matrices of size n^2 (where n is the number of cities in the TSP) are required: (1) a *distance matrix* to store the distances, (2) a *pheromone matrix* to store the ant pheromone values between every possible unique 2-city pair and (3) a *choice_info* optimization matrix for storing precomputed heuristic values $[\eta_{ij}]^{\beta}$ which do not need to be recomputed by each ant. For large values of n, it quickly becomes impractical to compute and store matrices of size n^2. These distance-based matrices are built during the `UpdatePheromoneTrails, ComputeDistances, ComputeNearestNeighborLists,` and `ComputeChoiceInformation` functions

and used at runtime for (1) querying distances, (2) querying/updating pheromone values and (3) querying/updating nearest neighbors in cases where all cities in the nearest-neighbor list have been visited (Dorigo & Stützle, 2004). These functions are underlined in the pseudocode in *Figure 1.2*.

Dissertation Goal

Computing and storing matrices of size n^2 is infeasible for large n. The goal of this dissertation is to investigate strategies for reducing the memory consumption requirements of the standard *MM*AS implementation (Wilke, n.d.).

The experimental strategies for reducing the memory footprint include (1) the use of *candidate sets* defined as the k-nearest neighbors from any given city, (2) removing matrices of size n^2, and (3) using a *sparse matrix* to store only pheromone values which differ from a dynamically updated default value by more than a specified tolerance limit. Proposed strategies will be evaluated using standard TSP instances selected from the TSPLIB library.

Research Questions

This research investigates the following questions:

1. How can the standard ACO implementation be modified to reduce memory requirements without adversely affecting solution quality significantly?
2. What is the tradeoff between memory use and solution quality?

Assumptions, Limitations, and Delimitations

The scope of this dissertation is limited, as specified below:

Limitation 1: Parallel ACO implementations are not considered. Instead, we focus exclusively on a baseline and experimental versions of *sequential MMAS* implementations.

Limitation 2: *MMAS* implementations vary throughout the literature in algorithm details such as branching factor and pheromone upper and lower bounds. Where possible, this work will use simplified *MMAS* implementation details and standard ACO implementation defaults to avoid introducing confounding factors and ease reproducibility.

Limitation 3: Although the best performing *MMAS* implementations use *local search* to optimize the best routes found by ants during iterations, local search will be disabled in this work to isolate the effect of memory limitations purely on the *stigmergic* operations that characterize ACO search.

Chapter Summary

In this chapter we introduced the concept of intractable combinatorial search problems, heuristics to find locally optimal solutions, and meta-heuristic to balance

exploitation with exploration in the context of solution space search. We introduce ACO as a meta-heuristic inspired by biological ant behaviors and the TSP as the standard test to measure meta-heuristic performance. We established that standard ACO exhibits memory complexity $O(n^2)$ which makes scaling ACO to large TSP instances infeasible. To address this problem, we described our approach of reducing ACO's memory storage requirements significantly by storing only a fraction of the inter-city distances and pheromone trails using a sparse matrix.

Chapter 2

Review of the Literature

Overview

This chapter provides an overview of the Ant Colony Optimization (ACO) metaheuristic and family of algorithms used to solve hard combinatorial problems. The review is divided into three sections: Foundations, Important Applications, and Recent Developments. Section 1: *Foundations* introduces the Ant Colony Optimization metaheuristic and catalogs the major algorithm variants chronologically that have emerged since the seminal 1991 Ant System publication. Section 2: *Important Applications* reviews a sampling of successful applications of ACO to a variety of important problems across different science and engineering fields. Section 3: *Recent Developments* reviews a sampling of several new ACO applications published within the last 5 years. Section 4: *Large Problem Instance Scaling* reviews a recent work that uses memory reduction techniques in a parallel ACO implementation. Section 5: *Efficient KNN Algorithm* reviews the replacement algorithm used to construct the *k* nearest-neighbors.

Section 1. Foundations

1.1: Introduction to Ant Colony Optimization

ACO is a meta-heuristic inspired by biological ants in nature for solving hard search problems like the TSP. In nature, ant colonies find good paths by depositing and following simple pheromone chemical signals. ACO algorithms model this behavior by implementing sets of agents (analogous to colonies of ants) which iteratively explore a

solution space, marking high quality solutions with "pheromone" values and biasing search routes toward high concentrations of those values (Dorigo & Stützle, 2004). A defining feature of Ant System and its variants is the probabilistic decision function called the *random proportional rule* which states that an ant *k* located at city *i* chooses the next city *j* with probability

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in \mathcal{N}_i^k}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}}, if\ j \in \mathcal{N}_i^k$$

Figure 2: Ant System Random Proportional Rule (Dorigo & Stützle, 2004).

where:

τ_{ij} is the pheromone trail between cities *i* and *j*

$\eta_{ij} = \frac{1}{d_{ij}}$ is the heuristic (inverse distance) between cities *i* and *j*

α and β are parameters to control the influence of τ_{ij} and η_{ij} on the probability

$\mathcal{N}_i^k$ is the set of unvisited cities by ant *k* at city *i*, also known as the *feasible neighborhood*

Following the introduction of AS, a number of AS variants were introduced and applied to a variety of problems successfully. Ten of the highest profile AS variants are reviewed in the following sections: Ant System, Elitist Ant System, Ant-Q, Ant Colony System, *MAX-MIN* Ant System, Rank-Based Ant System, ANTS, Best-Worst Ant System, Hyper-Cube Framework, and Beam-ACO (Dorigo & Stützle, 2018).

1.2: Ant System (1991)

Ant System (AS) is the original 1991 work by Dorigo which introduced the ant colony paradigm and the signature random proportionality rule for choosing arcs based on a combination of pheromone and heuristic input. At the end of the tour, *pheromone evaporation* occurs which serves to reduce influence of less frequently traveled arcs. The amount of pheromone evaporation is determined by the following equation:

$$\tau_{ij} = \tau_{ij}(1 - \rho)$$

Figure 3: Pheromone Evaporation (Dorigo & Stützle, 2004).

where τ_{ij} is the amount of pheromone along the arc connecting *i* and *j*, ρ is an algorithm parameter (usually 0.5 for AS, but ranging between 0.02 and 0.5 for other AS variants), and *L* is the set of all. After pheromone evaporation is complete, *pheromone deposition* occurs along all arcs according to the equation:

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

Figure 4: Ant System Pheromone Deposition (Dorigo & Stützle, 2004).

where *m* is the number of ants and $\Delta\tau_{ij}^{k}$ is the amount of pheromone ant *k* has deposited on its visited arcs connecting cities *i* and *j*. $\Delta\tau_{ij}^{k}$ is a function of the *total tour distance* C^k achieved by ant *k*:

$$\Delta\tau_{ij}^{k} = \begin{cases} 1/C^k, & if\ arc(i,j)\ in\ tour \\ 0, & else \end{cases}$$

AS was applied to the symmetric TSP to demonstrate how the algorithm attacked combinatorial problems. While AS did not achieve competitive TSP results compared to other existing methods, its novel approach to combinatorial optimization problems fueled

research interest in creating improved AS variants and hybrid methods which take advantage of local search to optimize solutions (Colorni, Dorigo, & Maniezzo, 1991). As discussed previously, AS introduced the idea of positive feedback pheromone deposition according to the *random proportional rule* (Figure 1.1) which combines heuristic and experiential data to produce a probabilistic choice for each ant as solutions are constructed. Subsequently, a variety of modifications to the random proportional rule, algorithm parameter management, and pheromone life cycle would emerge to form a family of Ant Colony Optimization algorithms.

1.3: Elitist Ant System (1992)

Elitist Ant System (EAS), originally introduced in 1992 as part of Dorigo's doctoral thesis and published in 1996, improved upon AS by applying heavier weights to the best solution found. After every iteration, each arc along the best tour receives an additional pheromone quantity: $1/C^{bs}$ where C^{bs} is the length of the best tour. Likewise, the pheromone deposition along each the best tour arcs is an extension of the AS pheromone deposition:

$$\tau_{ij} = \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + \ e/C^{bs}$$

Figure 5: Elitist Ant Pheromone Deposition (Dorigo & Stützle, 2004).

where *e* is a parameter. The authors evaluated EAS against the Oliver30 standard TSPLIB problem and report that, with certain values of *e*, EAS is able to find superior solutions to and in fewer iterations than AS (Dorigo, Maniezzo, & Colorni. 1996) (Dorigo & Stützle, 2004).

1.4: Ant-Q (1995)

Ant-Q was an attempt to integrate reinforcement learning methods into the ACO metaheuristic. The mechanisms of Ant-Q are identical to Ant Colony System (ACS) described in the next section with one exception: in Ant-Q, the initial pheromone trail value τ_0 is computed as a parameterized maximization function of outgoing pheromone trails to unvisited cities. However, it was shown that the expense of the Ant-Q τ_0 function did not achieve significant performance gains over a simple constant value used by ACS. For this reason, Ant-Q was abandoned and superseded by the ACS variant (Gambardella & Dorigo, 1995).

1.5: Ant Colony System (1997)

Ant Colony System (ACS) evolved as a simplification of the Ant-Q system as discussed in the previous section. ACS consists of 4 main modifications/improvements over AS described in (Dorigo & Gambardella, 1997):

1. ACS places more influence on the search experience of ants by replacing the random proportional rule with the *pseudorandom proportional rule*. The pseudorandom proportional rule is driven by a parameter: $0 \geq q_0 \geq 1$. The pseudorandom proportional rule selects a value q from a random uniform distribution in [0,1] and then applies the following decision function:

$$j = \begin{cases} argmax_{l \in \mathcal{N}_i^k} \{\tau_{il}[\eta_{il}]^\beta\}, if\ q \leq q_0 \\ random\ proportional\ rule, \quad else \end{cases}$$

Figure 6: ACS Pseudorandom Proportional Rule

In other words, with probability q_0 the arc with the largest product of the heuristic and pheromone is chosen (strongly exploiting ant exploration) while the rest of the time the choice defaults back to the standard random proportional rule AS choice mechanism.

2. Pheromone deposition and evaporation occurs only on arcs belonging to the best tour found so far. This modification confers a significant algorithm performance gain as the pheromone update operation changes from complexity $O(n^2)$ to $O(n)$ where n is the number cities.
3. The local pheromone update is applied during solution construction, as opposed to most other AS variants which update pheromone values as a *daemon action* after each iteration.
4. As ants traverse arcs, they consume a small amount of pheromone along those arcs. This measure is designed to bias ants away from all following the same path and converging to a local minimum.

(Dorigo & Gambardella, 1997)

Results from (Dorigo & Gambardella, 1997) show that ACS performance performs comparatively to genetic algorithms, evolutionary programming, and simulated annealing. Additionally, ACS was the first AS variant to employ *candidate lists*, which are subsets of all possible available neighboring city *j* from city *i*. Using candidate lists improved both the quality of the final tours and the speed of their discovery. ACS is currently considered one of the top performing AS variants (Dorigo & Stützle, 2004).

1.6: MAX-MIN Ant System (1997)

Along with ACS, *MAX-MIN* Ant System (*MM*AS) is one of the best performing and most actively researched AS variants. *MM*AS makes four improvements over AS (Stützle, Hoos 1997) (Dorigo & Stützle, 2004):

1. Only iteration-best or best-so-far ants are allowed to make pheromone deposits. This feature biases search toward neighborhoods of the most successful routes and away from less promising routes.
2. Pheromone values on trails are limited to values between the lower and upper limits: τ_{min} and τ_{max}. The effect of this limiting interval is to prevent early stagnation from massive pheromone accumulation on one trail. τ_{max} is set whenever a new best-so-far tour is found to be $\frac{1}{\rho C^{bs}}$ where C^{bs} is the best distance found so far. τ_{min} is set to $\frac{\tau_{max}\left(1-\sqrt[n]{0.05}\right)}{(avg-1)}$ where avg is the average number of choices an ant has at each city.
3. Trails are initialized to τ_{max}. This feature favors explorative search by preventing early pheromone depositions from dominating other potentially good routes early in the process.
4. Pheromone trails are re-initialized when stagnation approaches or no improved tour has been found after a number of consecutive attempts.

*MM*AS was tested against a variety of both symmetric and asymmetric standard TSPs. The authors report that *MM*AS outperformed ACS on average and in all cases beat or matched ACS. Additionally, *MM*AS was the first AS variant to have *local search* integrated into solution construction iterations. Using local search means that solutions found by (a limited number of) ants are optimized using a problem specific algorithm. In

the case of *MM*AS, the local search routines are *2-opt* and a specialized version of *3-opt* which rearranges a TSP tour such that all possible 2 (or 3) arc swaps that would improve the tour are considered. After this exhaustive optimization, a TSP tour is said to be "2 (or 3) - optimized" such that no further 2 (or 3) arc swaps exist that could improve the tour (Flood, 1956). Using local search significantly improved the tours generated as compared to *MM*AS without local search (Stutzle & Hoos, 1997).

1.7: Rank-Based Ant System (1997)

Rank-Based Ant System (AS_{rank}) is an extension of EAS where only the set of top $w-1$ ants are allowed to deposit pheromone. Likewise, the pheromone rule for AS_{rank} scales the pheromone deposition by the tour's rank.

$$\tau_{ij} = \tau_{ij} + \sum_{r=1}^{w-1} (w-r)\Delta\tau_{ij}^{r} + w\Delta\tau_{ij}^{bs}$$

Figure 7: Rank Based Ant System Pheromone Deposition

where $\Delta\tau_{ij}^{r} = 1/C^{r}$ and $\Delta\tau_{ij}^{bs} = 1/C^{bs}$. Note that the best solution so far may not be among the iteration's best, but is still awarded the top weighting (Bullnheimer, Hartl, & Strauß, 1997) (Dorigo & Stützle, 2004).

Bullnheimer et al. compared the performance of AS_{rank} with AS, EAS, 2 versions of Simulated Annealing, and Genetic Algorithm across 5 TSP instances. The results of these computational studies show that AS_{rank} competes closely with EAS and for the larger TSP instances beat all other compared algorithms, finding solutions within 1% of the known optimal solution (Bullnheimer, Hartl, & Strauß, 1997).

1.8: Approximate Nondeterministic Tree Search (1999)

Approximate Nondeterministic Tree Search (ANTS) uses concepts from mathematical programming to compute lower bound estimates of tours as they are constructed. Those estimates are used by ants to eliminate choices that are likely to lead to solution costs worse than the best solution found so far. While fruitless selection can be eliminated early by these estimates, computing the estimates is an expensive operation that must be applied to every partial solution.

ANTS deviates from most other AS variants in two fundamental ways:

1. The random proportional rule is replaced with a single parameter function:

$$p_{ij}^{k} = \frac{\zeta\tau_{ij} + (1-\zeta)\eta_{ij}}{\sum_{l \in \mathcal{N}_i^k} \zeta\tau_{il} + (1-\zeta)\eta_{il}}, if\ j \in \mathcal{N}_i^k$$

   Figure 8: ANTS Action Choice Rule

   where $0 \leq \zeta \leq 1$ is a parameter.

2. Traditional AS style pheromone evaporation does not occur with ANTS. Instead, pheromone levels are controlled by the moving average of the last *l* tours: arcs along solutions *worse* than the current moving average *decrease* pheromone values; arcs along solutions *better* than the current moving average *increase* pheromone values. (Maniezzo, 1999)

   While ANTS has not been applied to the TSP, in (Maniezzo, 1999) ANTS was applied to the Quadratic Assignment Problem and showed competitive results compared to other state of the art algorithms.

1.9: Best-Worst AS (2000)

Best-Worst Ant System (BWAS) introduced 3 modifications to the classical AS according to (Cordon, Fernandez de Viana, Herrera, and Moreno, 2000):

1. In addition to (and similar to *MM*AS and ACS) rewarding the best so far solution with extra pheromone, BWAS also penalizes the *worst* iteration solution by removing pheromone along the worst solution's arcs which are not common to the best so far solution arcs.
2. BWAS reinitializes pheromone trails often to bias search toward exploration.
3. Pheromone values undergo a *mutation* mechanism inspired by genetic computing. With random probabilities, the pheromone matrix will experience value mutations such that more recent pheromones are more susceptible to mutation that older values. The idea behind pheromone mutation is to counteract convergence toward suboptimal local minima.

   (Cordon et al., 2000)

Cordon et al. evaluated BWAS against AS and ACS across 8 standard TSP instances. While BWAS achieved positive comparable performance, current BWAS results are not fully conclusive and insufficient to judge BWAS's performance against the top ACO variants: *MM*AS and ACS (Dorigo & Stützle, 2004).

1.10: Hyper-Cube Framework (2004)

The Hyper-Cube Framework limits the pheromone allowed values to the interval [0, 1]. This is accomplished by modifying the pheromone update rule to guarantee that the trail values remain less than one. The pheromone update can be seen as "a shift of the old pheromone vector toward the vector given by the weighted average of the solutions used

in the pheromone update" (Blum & Dorigo, 2004). Like ANTS, Hyper-Cube Framework has not been applied to the TSP. However the Hyper-Cube Framework did produce 2 significant results: (1) the theoretical result that the average solution quality of classical AS is expected to increase over time and (2) experimental demonstration that the automatic scaling of the objective function values results in more robust ACO behavior (Blum & Dorigo, 2004).

1.11: Beam-ACO (2005)

Beam-ACO is an AS variant based on the branch-and-bound derivative algorithm called Beam-search. Classical Beam-search is a type of *tree search* which treats a partial solution as a node on a tree. Decisions as to how to continue extending the partial solution become questions of which areas of the tree are to be explored. In Beam-search, this decision is made using lower bound estimates and results in a set of different and independent exploration paths. The total number of exploration paths is kept within a range to avoid explosive search growth (Blum, 2005).

Blum et al combined the solution construction mechanisms of ACO with the partial search characteristics of Beam-search into the hybrid algorithm called Beam-ACO. Beam-ACO was applied to the Open Shop Scheduling Problem (OSP) which is closely related to the Single Machine Total Weighted Tardiness Scheduling Problem discussed in a later section. Informally, the OSP problem is: given a set *n* jobs that must be processed for given amounts of time at *m* processing stations, assign a time to process each job at each station such that the total processing time is minimized. In (Blum, 2005), Beam-ACO was used to solve a large set of OSP instances and the performance compared to standard OSP-specific ACO and to state-of-the-art Genetic Algorithm variants. Results

show that Beam-ACO met or exceed the most performant OSP algorithms in terms of solution quality (Blum, 2005).

Section 2. Survey of Important ACO Applications

ACO has been successfully applied to a variety of important, fundamental problems and applications. (Dorigo & Stützle, 2018) categorize 25 examples of ACO applications to important problems by problem type: routing, assignment, scheduling, subset, machine learning, and bioinformatics. This section present examples from each of these fields.

2.1: Routing: AntNet

AntNet is an adaptive, distributed network routing protocol used to maintain best route paths between source and destination nodes in a dynamic, packet-switched data communications network. The node graph of a dynamic communications network is fundamentally different from static TSP graphs: heuristics such as cost of travel between each node are not known beforehand, change over time according to the data packets and global update routines (a.k.a *daemon actions*) are infeasible as the information collected by ants is distributed across the network nodes. In fact, the purely decentralized environment of dynamic networks mimic more closely the actual conditions under in which the biological ants that inspired AS operate. AntNet ants act as distributed agents that update routing tables and local (to the node) network conditions by depositing information they individually learn from traversing the network at the individual nodes. *Forward ants* periodically launch from individual source nodes *s* toward destination nodes *d* with probability proportional to the fraction of local network traffic that requires routing from *s* to *d*. In this way, network data conditions naturally prioritize the routes

explored by ants. Once forward ants reach their destination, they are deleted and their memory is transferred to a new *backward ant*.

Backward ants return to the source node via their reverse-route and update pheromone matrices (stored at each node) with information learned from their forward ant counterpart. The AntNet algorithm includes other details and network specific concerns such as route loop detection and forward ant lifetime management which are not discussed here but are available in (Caro & Dorigo, 1998) and (Dorigo & Stützle, 2004). In (Caro & Dorigo, 1998), the authors evaluated AntNet by conducting comparisons on a network simulation system with other algorithms. Results from (Caro & Dorigo, 1998) show that AntNet's performance meets or exceed the performance of leading routing algorithms: OSPF, SPF, Adaptive Bellman-Ford, Q-Routing, and PQ Routing.

2.2: Assignment: Quadratic Assignment Problem (QAP)

Assignment problems are a class of NP-Hard computational problems where the goal is to assign a set of items to a set of resources under some constraints. The QAP is a type of assignment problem that appears widely across many fields. The QAP problem is to assign a set of facilities to a set of locations subject to given (1) distances between locations and facilities and (2) flows (weights) between facilities. (Maniezzo, 1999) used standard QAP problems as the test instances for the introduction of the ANTS system. Specialized versions of AS and *MM*AS have been developed specifically to be applied to QAP. Results show that *MM*AS and ANTS are among the best performing algorithms available for the QAP and that ANTS actually outperforms tabu search algorithms which typically perform better for QAP instances (Dorigo & Stützle, 2004).

2.3: Scheduling: Single Machine Total Weighted Tardiness Scheduling Problem (SMTWTP)

Given *n* jobs which must be processed sequentially. Each job is associated with a processing time, weight, and due date. The SMTWTP problem is defined as finding the sequence of jobs which minimizes the *total weighted tardiness* defined informally as the sum of the products of weights and elapsed past-due times. Versions of ACS developed to solve the SMTWTP have produced optimal solutions to ORLIB standard instances (Dorigo & Stützle, 2004).

2.4: Subset: Set Covering Problem (SCP)

Given a set of *n* elements and a collection S of subsets of the set of all n elements: {1,2,..,*n*} the Set Covering Problem is to find a smallest subcollection in S whose union equals the set of all n elements. (Lessing, Dumitrescu, & Stützle, 2004) applied *MM*AS, ACS, *MM*AS-ACS hybrid, and ANTS to standard instances from the ORLIB. ANTS and ACS were shown to be the most performant of the AS Variants and competitive with state-of-the-art SCP algorithms when tested against 16 TSP instances referenced in the work.

2.5: Machine Learning: Bayesian Networks

A Bayesian Network is a fully connected, directed acyclic graph where arcs between 2 nodes from *i* and to *j* indicate a conditional probability of *j* given *i*. The Bayesian Learning problem is to derive a Bayesian Network from an input data set of variables – i.e. specific instances of arcs. In (Campos, Fernández-Luna, Gámez, & Puerta, 2002), the authors present a specialized ACS variant that builds a network iteratively by adding arcs subject to the acyclic constraint of Bayesian Network. This ACS variant was

compared to 4 other algorithms across 3 standard data set instances. The results were that ACS produced the best solutions of all other algorithms (Campos, Fernández-Luna, Gámez, & Puerta, 2002) (Dorigo & Stützle, 2004).

2.6: Bioinformatics: DNA Sequencing

In (Blum & Vallès, 2006), the authors apply ACO methods to the DNA Sequencing problem. DNA Sequencing is the process of reconstructing an entire DNA chain from processing only fragments of the chain. The state of the art sequencing method is called *sequencing by hybridization*. In this process, a grid of subsequences of length 3 is used as a probe to detect the occurrence of those subsequences from a larger DNA fragment. The existence of false negatives (sequences present in the fragments that are not detected by probes) and false positive errors (probes that falsely identify a subsequence) make sequencing an NP-Hard problem (Blum & Vallès, 2006). In this work, Blum et al. develop a new constructive heuristic and a specialized ACO for sequencing by hybridization. Published results show that their ACO methods achieve state of the art performance for previously published sequence problem instances.

Section 3: Recent Developments (2016-2020)

While advances in ACO algorithm variants continues, most recent research interests into ACO have largely focused on parallelization of ACO processing, applications of ACO to important practical problems, and hybrid approaches that combine ACO with other swarm intelligence algorithms. The following section reviews some of the ACO innovations over the last 5 years (2016-2020).

3.1: Application to Practical Problems

Wireless Sensor Networks (WSN) are composed of a network of sensors (*sensor nodes*) that measure some phenomena of interest and periodically route their data through the network to a central processing unit called a *sink node* to process the collected data. WSNs experience what is known as the *hot spot problem*: sensor nodes neighboring a fixed sink node experience inordinately higher wireless network traffic than sensor nodes further away from the sink node. The result is that the lifetime of the WSN is shortened as the heavily trafficked sensor nodes are depleted of power more quickly than the remote nodes.

A known mitigating strategy is to introduce *mobile sink nodes* which relocate to positions where direct communication with *cluster head* nodes is possible. This direct communication is energy efficient because there is no energy-expensive routing involved. Likewise, finding an optimal path for the mobile sink nodes to reach their destination becomes a critical component of the preserving the lifetime of the WSN. In (Wang, Cao, Sherratt, Park, 2017), the authors develop a specialized ACO algorithm based on distances between cluster head nodes. Using a network simulator, they compared their algorithm with a previously developed ACO based mobile sink node solution and achieved superior results in terms of WSN lifetime and total energy consumption (Wang, Cao, Sherratt, Park, 2017).

3.2: Parallelization of ACO Processing

As a result of the trend to move parallel computation to newly available GPUS, parallel implementations of ACO algorithms that can exploit independent processing tracks have attracted significant attention in recent years. In (Zhou, He, Hou, Qiu, 2018),

the authors present an ACO algorithm designed to use GPU processing architecture. Their contribution consists of 3 main parts:

1. Two new GPU kernel strategies to be used during tour construction
2. A new implementation of the roulette-wheel selection routine optimized for GPU
3. A new algorithm called single-instruction, multiple-data (SIMD) that takes advantage of GPU architecture during tour construction. Zhou et al evaluated their algorithm on TSP instances ranging up to 4461 cities and report (1) a speedup of up to 44 times faster than the CPU version and (2) superior performance to existing parallel ACO GPU implementations

   (Zhou, He, Hou, Qiu, 2018).

3.3: Algorithm Advances

After almost 2 decades since the original AS system was introduced, new AS variants continue to emerge from the research literature. In (Deng, Xu, & Zhao, 2019), Deng et al introduce an AS variant called ICMPACO which designed for both solution convergence speed and quality from good exploration. ICMPACO introduces several new mechanisms to the standard ACO model:

1. The problem is subdivided into smaller problems which are solved independently to compose a solution
2. 2 classes of ants are introduced: *elite* ants which favor best known tours and *common* ants which favor exploration
3. *Pheromone diffusion* is introduced in which pheromone deposition slowly spreads to surrounding local arcs

4. A *co-evolution* mechanism is introduced to share tour information among ant sub-populations

ICMPACO was evaluated against 8 standard TSPLIB TSP instances in comparison with 2 standard ACO implementations. Reported results show that ICMPACO outperformed the competing algorithms in terms of final solution quality, but tended to require longer run-times to reach those superior solutions (Deng, Xu, & Zhao, 2019).

Section 4: Large TSP Instance Scaling Methods

A recent work examines modifying *MM*AS to use the nearest neighbor matrix as the pheromone matrix replacing the $O(n^2)$ storage complexity with the much smaller O(nk) complexity of storing only the nearest neighbor graph (the *candidate set*) where $k \ll n$ (Peake, Amos, Yiapanis, & Lloyd, 2019). This work also tests two *fallback methods* defined as approaches to handle situations where an ant reaches a city during tour construction from which no more moves are available from within the candidate set. Given an ant at node n_i from which no nearest neighbors in the candidate set are available, the next node n_{i+1} is chosen in one of two ways:

1. *Heuristic Fallback*: This method simply selects the nearest node not yet visited from the set of remaining nodes and discards any sense of previous traffic (i.e. pheromone contribution) along the $n_i \rightarrow n_{i+1}$ edge.
2. *Pheromone Map Fallback*: This method stores pheromone values for edges not found in the candidate set but which do belong to a best ant's tour in a HashMap. This map is queried for all remaining nodes not yet visited. Any nodes not found in the map assume the minimum pheromone value τ_{min}. The $n_i \rightarrow n_{i+1}$ edge chosen is that

edge with the highest edge weight computed as function of pheromone value and Euclidean distance.

The work is evaluated using extremely large TSP instances found from the TSP Art standard set (TSP Art Instances) with an experimental version of the Parallel *MAX-MIN* Ant System. The authors conclude that (a) both fallback methods produce about the same solution quality with Heuristic Fallback performing much faster and therefore emerging as the more practical method and (b) fewer than 5% of all "next-city" choices used *either* fallback method.

Section 5: Efficient KNN Algorithm

A highly cited and successful work by (Arya et al., 1998) presents an efficient KNN approximation algorithm which runs in $O(n \log n)$ time. This complexity is superior to the method in the standard ACO implementation that uses the full distance matrix and is therefore at least $O(n^2)$ in memory and processing complexity. The authors introduce a novel concept called a *balanced box-decomposition (BBD) tree* which divides the search space into recursive rectangular regions containing sets of points a introduces a novel method called *priority search* to locate neighboring points quickly. A notable property of this algorithm is that it can be used to locate either approximate nearest neighbors or exact nearest neighbors. In this dissertation, the R package (https://cran.r-project.org/web/packages/RANN/RANN.pdf) which implements this KNN algorithm is used in to precompute KNN data for a TSP instance, referred to as an "index file". The experimental MMAS algorithm developed in this work requires this index file in addition to the TSP file.

Chapter Summary

This chapter provided historical context for ACO by reviewing 11 major ACO variants, 6 application areas to which ACO has significantly contributed, and several examples of recent developments in ACO research.

Chapter 3

Methodology

Overview of Research Methodology

The goal of this dissertation is to investigate methods for eliminating the n^2 sized large matrices in the standard ACO *MM*AS implementation. Specifically, we evaluate (1) the use of a sparse matrix to replace the pheromone matrix and (2) the KNN approximation algorithm (Arya et al., 1998) to replace the exact KNN algorithm in the standard implementation. This section describes the steps used for implementing and evaluating the memory consumed and impact on solution quality of these replacement data structures and algorithms.

Experimental Design

This section discusses the methodology that was used in this dissertation including: (1) a listing of the specific TSPLIB benchmark problems used for evaluation, (2) measures for evaluating solution quality, (3) a description of the experimental *MM*AS algorithm that was developed: $MMAS_{exp}$, and (4) the procedures used for isolating and measuring the effect on solution quality of using $MMAS_{exp}$ over the standard *MM*AS ACO code: $MMAS_{std}$.

TSPLIB benchmark problems

Table 1 below lists the standard symmetric 2-D Euclidean TSPLIB instances that were used for evaluation. In order to execute the standard ACO implementation as a comparison, *n* was kept reasonably small.

TSP
att532
rat783
vm1084
pcb1173
d1291
u1817
u2319
pcb3038
rl5915
pla7397

Table 1: Test TSPLIB instances; the filename's suffix gives the instance size

Solution Quality Metrics

Solution Quality follows the Dorigo text methodology and is defined as the "Percent Deviation From Optimum" (PDFO) value curve measured over a fixed interval of 100 CPU seconds (Dorigo & Stützle, 2004).

Experimental MMAS Algorithm

$MMAS_{exp}$ is a modified version of Java $MMAS_{std}$ implementation by (Wilke, n.d.) with the following changes:

1. *Distance and Choice_Info Matrices Dropped*

   The *distance* and *choice_info* $O(n^2)$ sized matrices (the distance matrix and an optimization matrix that stores the products $[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}$ for every node pair *i,j* respectively) are discarded as a result of their intractable size. Their data is instead recomputed on demand. Abstractions that use memoization to cache results of previous calls were considered but were found to be less efficient than simply recomputing values.

2. *Pheromone Matrix Replaced with Sparse Matrix*

   The n^2-sized *pheromone* matrix is replaced with a *sparse matrix* data structure. The SparseMatrix and SparseVector classes (Sedgewick & Wayne, 2019) together implement a sparse 2-D matrix of doubles.

3. *Pheromone Initialization*

In $MMAS_{std}$, all pheromones are initialized to *trail_max* which requires setting n^2 pheromone values in the dense pheromone matrix. In $MMAS_{exp}$, the SparseMatrix pheromone structure's single default value is set to *trail_max*. In the standard *MMAS* implementation, *trail_max* is initialized to $\frac{\rho}{nn\ best\ tour}$ where $nn\ best\ tour$ is the length of the *greedy nearest-neighbor tour* which is computed in the following way:

for each city c, the next city is:

(a) The closest unvisited city from among the *k* unvisited cities in the nearest-neighbor matrix, if one exists or

(b) if there are no remaining unvisited cities in the nearest-neighbor matrix for city c, the city from all remaining unvisited cities with the minimum distance to city c.

The greedy nearest-neighbor tour exhibits storage complexity $O(n)$ and computational complexity $O(n^2)$. To reduce the computational complexity to $O(n \log n)$, $MMAS_{exp}$ uses a modified greedy nearest-neighbor algorithm which checks *at most* $\log n$ distances from each of n cities in part *b* of the algorithm listed above. This simplification comes at the cost of a longer tour used to initialize the algorithm.

4. Pheromone Access

In $MMAS_{std}$, pheromones values are accessed directly by directly accessing the i, j^{th} element of the dense pheromone matrix. In $MMAS_{exp}$, pheromone values are accessed by

querying the sparse matrix for the i, j^{th} element. If the sparse matrix contains the i, j^{th} element, that element is returned. Otherwise, the default value is returned.

5. *Pheromone Update*

In $MM\text{AS}_{std}$, pheromones are updated by:

1. Depositing pheromone along the best ant's tour
2. Reducing all pheromone values by a factor of $(1 - \rho)$ and
3. Coercing all pheromone values into the interval $[\tau_{min}, \tau_{max}]$.

In $MM\text{AS}_{exp}$, the pheromones are updated by:

1. Depositing pheromone along the best ant's tour
2. Reducing the default value by a factor of $(1 - \rho)$, coercing the value into the interval $[\tau_{min}, \tau_{max}]$
3. Updating all non-default values v_{ij} in the sparse matrix according to the following rule: if the pheromone value to be stored is *sufficiently close* (defined below) to the default value then *delete* the value from the sparse matrix. Otherwise, store the value in the sparse matrix, coercing the value into the interval $[\tau_{min}, \tau_{max}]$.

4. $scale$ is an introduced parameter to $MM\text{AS}_{exp}$ to control the precision of the "*sufficiently close"* evaluation mentioned previously. Larger $scale$ values result in a higher number pheromone values being stored compared to smaller values. Specifically, precision is defined as 10^{-scale}.

a. KNN

The nearest neighbor matrix of the original implementation will be retained. An edge between vertices *u* and *v* exists in an undirected graph if *v* is one of the *k*-nearest neighbors of *u* or if *u* is one of the *k*-nearest neighbors of *v*; thus the *k*-nearest neighbor graph will have at most $n * k$ edges. The k-nearest neighbor graph will be populated by the k nearest neighbors for each city using a KNN implementation based on (Arya, Mount, Netanyahu, Silverman, & Wu, 1998) which runs in $O(n \log n)$ time – an improvement over the $O(n^2)$ implementation in the standard ACO implementation which uses the n^2 sized distance matrix. A nearest neighbor index file is a new required input to $MM\text{AS}_{exp}$.

Experimental Procedure

1. $MM\text{AS}_{exp}$ was coded as a modified version of the standard Java port of the ACO implementation (Wilke, n.d.) as described in the Experimental *MM*AS Algorithm section above.
2. An index input file was generated for each TSPLIB instance evaluated
3. For each TSP instance, both $MM\text{AS}_{std}$ and $MM\text{AS}_{exp}$ were executed and recorded using the *MM*AS algorithm for *k* values 40 and 20 (where *k* is the number of nearest neighbors stored for each of *n* cities in the nearest neighbor matrix), without local search, and with a specific $scale$ value determined empirically for each TSP (described in the next section). All other variables were left to the ACO standard implementation defaults.
4. PDFO of each run for 10 trials of both $MM\text{AS}_{exp}$ and $MM\text{AS}_{std}$ were measured over 100 second trials. These PDFO data sets were analyzed to quantify both the

performance costs incurred and memory savings realized by the sparse matrix and KNN approximation algorithm replacements described above.

Scale Selection and Memory Savings:

In MMAS, only the iteration best ant is allowed to deposit pheromone. This is critically important for the memory savings mechanism in $MMAS_{exp}$ because it means that after each iteration, only *n* values are (potentially) increased while the other $n(n-1)$ values are only reduced. This means that most arc values start with the same initial value and are reduced by the same factor after each iteration. The sparse matrix implementation takes advantage of this fact by updating the *default value* to be what would be the majority arc value in the pheromone matrix without actually having to compute or track a matrix majority value.

In both $MMAS_{std}$ and $MMAS_{exp}$, pheromone values are deposited based on the quality of the entire path: the better the path (i.e. the lower the cost) the more pheromone is deposited. In $MMAS_{exp}$, for any arc along the depositing ant's route, if the new value is within the precision of the default value then no pheromone deposits on that arc. If precision is too high then too many pheromone arcs are retained in the sparse matrix and memory grows too aggressively. On the other hand, if precision is too low then too few pheromone arcs are retained in the sparse matrix and as a result too few solutions are available to evolve better solutions efficiently. Therefore, $scale$ must be selected such that the number of pheromone arcs stored is large enough to facilitate solution improvement without resulting in spiking memory usage.

Chapter 4

Results

Overview of Results

This section details the results of applying the methodology described in Chapter 3 and is divided into 2 parts: (1) *Experimental ACO Algorithm Performance Metrics* which defines the Quality and Memory metrics of $MMAS_{exp}$ and $MMAS_{std}$, the experimental ACO algorithm and standard ACO algorithm respectively and (2) *TSP Instance Results* which shows the results of the Quality and Memory measures on each of the test TSP instances.

Experimental ACO Algorithm Performance Metrics

Let p and p^m denote the mean PDFO over 10 runs of 100 seconds each using the $MMAS_{std}$ and $MMAS_{exp}$ algorithms respectively. Let Q be defined as the ratio of the mean cost using $MMAS_{exp}$ to that using the $MMAS_{std}$ as $Q = (100 + p^m) / (100 + p)$. Similarly, let M be defined as the ratio the *mean memory* usage over 10 runs using $MMAS_{exp}$ to that using the $MMAS_{std}$. Let *memory usage* be defined as the total size of the storage data structures required by the standard implementation, M_{std}, and the experimental implementation, M_{exp} for the TSP instances. M_{std} storage data structures consists of the following:

(1) the distance matrix of size n^2

(2) the *totals matrix* of size n^2, which is an optimization used to cache the *numerator* of the *Random Proportionality Rule* for each iteration

(3) the *pheromone matrix* of size n^2

(4) the *k-nearest-neighbor matrix* of size $n * k$

Hence, for any TSP instance consisting of n cities, $M_{std} = 3n^2 + n * k$.

M_{exp} storage data structures consists of the following:

(1) the maximum size of the dynamic pheromone *sparse matrix* during runtime

(2) the *k-nearest-neighbor matrix* of size $n * k$

TSP Instance Results

The two ratio measures, *Q* and *M*, are shown for each test TSP instance in the 2 tables below for nearest-neighbor lengths K=40 and K=20 respectively.

Quality and Memory Measures for Test TSP Instances for K=40									
TSP	**n**	**Scale**	**Precision**	Q	M	Q_{std}	M_{std}	Q_{exp}	M_{exp}
att532	532	4.66	2.19E-05	1.160	0.043	4.12	8.60E+05	20.73	3.71E+04
rat783	783	4.40	3.98E-05	1.240	0.030	23.85	1.85E+06	53.55	5.57E+04
vm1084	1084	6.00	1.00E-06	1.174	0.020	12.14	3.55E+06	31.64	7.10E+04
pcb1173	1173	5.02	9.55E-06	1.162	0.020	12.73	4.15E+06	30.99	8.16E+04
d1291	1291	5.02	9.55E-06	1.194	0.018	10.44	5.03E+06	31.85	8.84E+04
u1817	1817	5.25	5.62E-06	1.184	0.033	16.68	9.94E+06	38.09	3.30E+05
u2319	2319	5.70	2.00E-06	1.087	0.017	23.51	1.62E+07	34.24	2.71E+05
pcb3038	3038	5.70	2.00E-06	1.123	0.008	29.71	2.77E+07	45.67	2.29E+05
rl5915	5915	7.00	1.00E-07	0.912	0.009	60.06	1.05E+08	46.03	9.37E+05
pla7397	7397	8.00	1.00E-08	0.934	0.003	66.64	1.64E+08	55.59	4.96E+05

Table 2: Measures of Q and M against the test TSPLIB instances for K=40

Quality and Memory Measures for Test TSP Instances for K=20									
TSP	**n**	**Scale**	**Precision**	Q	M	Q_{std}	M_{std}	Q_{exp}	M_{exp}
att532	532	4.66	2.19E-05	1.167	0.043	4.03	8.60E+05	21.35	3.69E+04
rat783	783	4.40	3.98E-05	1.241	0.030	23.81	1.85E+06	53.70	5.61E+04
vm1084	1084	6.00	1.00E-06	1.188	0.020	10.47	3.55E+06	31.20	7.15E+04
pcb1173	1173	5.02	9.55E-06	1.193	0.019	13.11	4.15E+06	34.92	7.78E+04
d1291	1291	5.02	9.55E-06	1.185	0.018	10.62	5.03E+06	31.09	8.88E+04
u1817	1817	5.25	5.62E-06	1.189	0.034	16.77	9.94E+06	38.78	3.41E+05
u2319	2319	5.70	2.00E-06	1.086	0.016	23.15	1.62E+07	33.77	2.64E+05
pcb3038	3038	5.70	2.00E-06	1.132	0.009	28.91	2.77E+07	45.98	2.45E+05
rl5915	5915	7.00	1.00E-07	0.918	0.010	59.25	1.05E+08	46.21	1.01E+06
pla7397	7397	8.00	1.00E-08	0.926	0.003	67.37	1.64E+08	55.03	4.94E+05

Table 3: Measures of Q and M against the test TSPLIB instances for K=20

pla7397 Measure Walk-through for K=40

For clarification, the measures for the pla7397 TSP instance for K=40 are described here. The *n* value of 7,397 indicates that this TSP instance contains 7,397 nodes. The *scale* value was reached manually for each TSP instance. Recall that the *scale* value directly determines *precision* which is defined as 10^{-scale} and that any pheromone value that falls within +/- *precision* of the sparse pheromone matrix default value is removed and considered implicitly to be the default value. The *precision* value of 10^{-8} was reached empirically by manually trying multiple *scale* values between 5.00 and 10.00 in a binary search pattern until the following two conditions occurred simultaneously: (1) a sufficient number of distinct (i.e. non-default) pheromone values were retained across iterations for the standard *MMAS* solution quality improvement mechanics to work – indicated by a steadily improving best tour, and (2) pheromone values were pruned frequently enough to keep the sparse matrix element count low and prevent transient memory consumption *spikes* (detailed in the next section) – indicated by a direct measure of the pheromone sparse matrix element count. $Q_{std} = 66.64$, $M_{std} =$

1.64E+08, $Q_{exp} = 55.59$, $M_{exp} = 4.96$E+05 are the best PDFO and largest memory consumption of standard *MMAS* and the experimental *MMAS* variant respectively. $Q = 0.933675$ is the ratio of the experimental *MMAS* variant solution quality (lower is better) compared to standard *MMAS*. $M = 0.003020$ is the ratio of the experimental *MMAS* memory consumption (lower is better) quality compared to standard *MMAS*.

Transient Memory Consumption

Let *transient memory consumption* be defined as the sparse pheromone matrix memory used during the experimental *MMAS* variant algorithm execution expressed as a percentage of n^2 (i.e. the maximum possible sparse pheromone matrix size). Transient memory consumption is sensitive to choice of *scale*. If scale is too low, no (or very few) values will be saved and as a result no solution improvement outside of random discoveries will occur. Conversely, if scale is too high, then too many distinct pheromone values are stored resulting in a memory consumption *spike* observed during the initial phases of path construction iterations. This transient memory consumption spike is depicted in *Images 4.1* and *4.2* which show pdfo and transient memory consumption for TSP instance att532. Note the spike in *Image 4.1* which shows a transient memory consumption value just over 60% of total pheromone matrix capacity when scale is set too high at 6.00. When scale is reduced to 4.66, this spike is no longer present while pdfo remains in a similar range as depicted in *Image 4.2*.

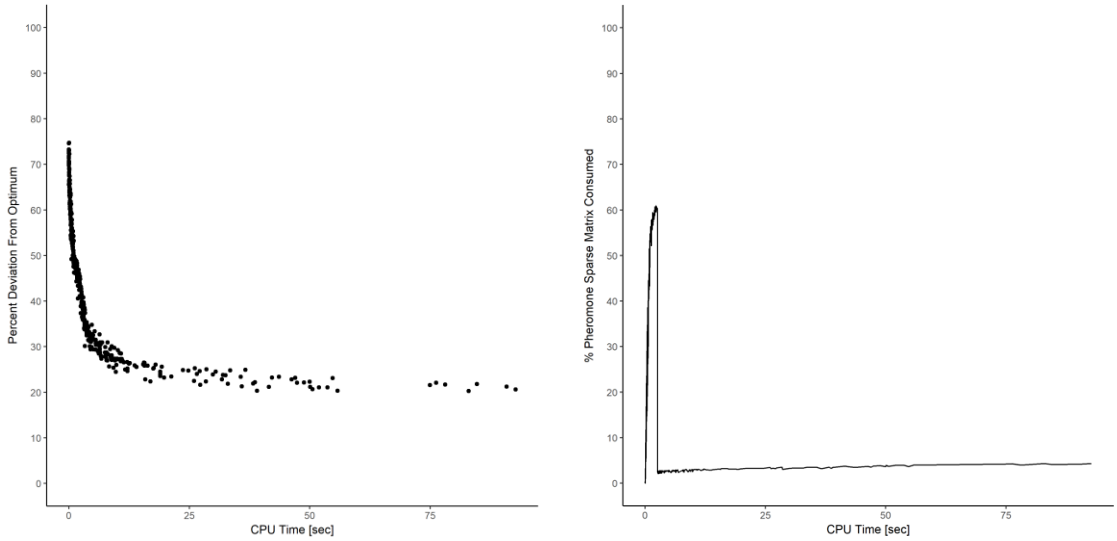


Figure 9: PDFO and Transient Memory Consumption for TSPLIB instance att532 for K=20 and scale = 6.00 over 10 iterations

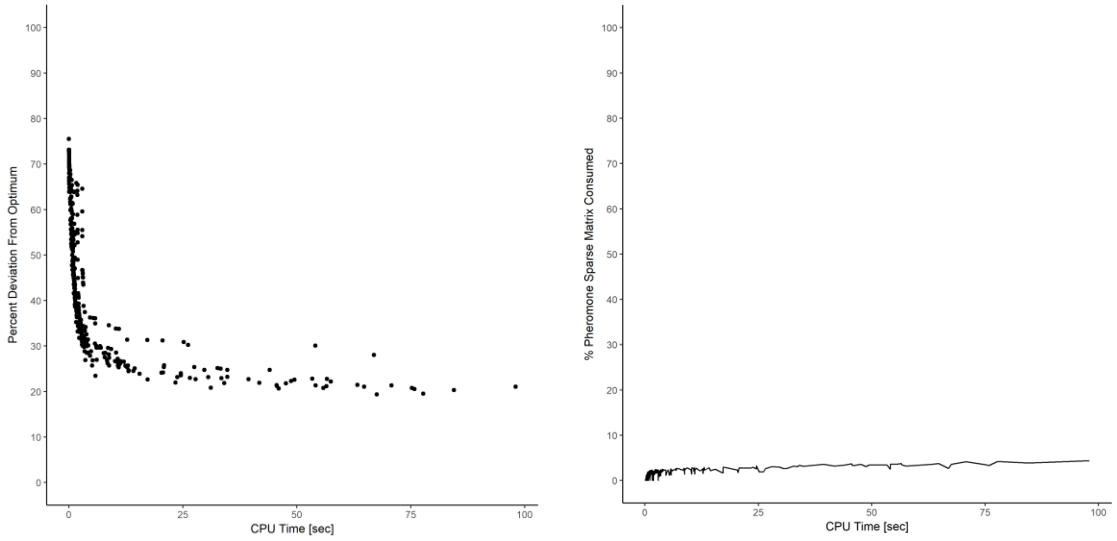


Figure 10: PDFO and Transient Memory Consumption for TSPLIB instance att532 for K=20 and scale = 4.66 over 10 iterations

Chapter 5

Future Work and Conclusions

Three natural extensions of this work - *dynamic scaling, path-dependent scaling,* and *stagnation detection* – are reviewed briefly below. In *dynamic scaling,* methods could be developed for discovering the optimal scale parameter during pheromone deposition and evaporation by locating the memory consumption threshold. During runtime, different scale values could be tested based on some strategy involving sampling differences between pheromone values and the sparse matrix default value. The scale value could then be adjusted to reach a desired goal such as an absolute maximum memory consumption, sparse matrix element count, or a particular PDFO.

Path-dependent scaling - an extension to dynamic scaling - could apply different scale values to different elements of the sparse matrix depending on the "quality" of the element. Element "quality" could be determined by any number of measures such as the element's membership in a recent iteration best path (similar to the elitist ant strategy). Sparse matrix elements (i.e. tour arcs) belonging to better tours and possibly their direct neighbors could use more precise scaling so that pheromone storage capacity skews toward higher quality arcs.

Finally, *stagnation detection* could influence the scale value dynamically. In several Ant System variants, including MMAS, stagnation is detected using a combination of measures such as the *branching factor* – a parameterized measure of the relative distribution of pheromone across all outbound arcs of each city (Dorigo & Stützle, 2004). When stagnation is detected, a reset is triggered where all pheromone values are dropped and the ant construction phase restarts. Instead of discarding all

pheromone values, the scale value could decrease according to some schedule. A reduction in scale would result in a graduated "forgetting" of exploration experience as elements are pruned from the sparse matrix. This could have the advantage of retaining some portion of the ant's experience while increasing the exploration through pruned pheromone values.

The results of this study indicate that substantial memory savings can be achieved by (1) removing the distance and totals matrices, (2) using an efficient k-nearest-neighbors and greedy *KNN* TSP heuristic, and (3) substituting a sparse matrix for the dense pheromone matrix in $MMAS_{std}$. It was shown in multiple TSP test instances that a key factor in optimizing the memory efficiency is choosing a scale parameter value that keeps the number retained elements in the sparse matrix as close to the lower edge of the memory consumption threshold without causing a memory consumption spike.

References

ACOTSP (2004). Available from http://www.aco-metaheuristic.org/aco-code/

Ant Colony Optimization. (n.d.). Retrieved from http://www.aco-metaheuristic.org/

Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., & Wu, A. Y. (1998). An optimal algorithm for approximate nearest neighbor searching fixed dimensions. Journal of the ACM, 45(6), 891–923. doi: 10.1145/293347.293348007

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, *18*(9), 509–517. doi: 10.1145/361002.361007

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization. *ACM Computing Surveys,35*(3), 268-308. doi:10.1145/937503.937505

Blum, C., & Dorigo, M. (2004). The Hyper-Cube Framework for Ant Colony Optimization. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, *34*(2), 1161–1172. doi: 10.1109/tsmcb.2003.821450

Blum, C. (2005). Beam-ACO—hybridizing ant colony optimization with beam search: an application to open shop scheduling. *Computers & Operations Research*, *32*(6), 1565–1591. doi: 10.1016/j.cor.2003.11.018

Blum, C., & Vallès, M. Y. (2006). New Constructive Heuristics for DNA Sequencing by Hybridization. *Lecture Notes in Computer Science Algorithms in Bioinformatics*, 355–365. doi: 10.1007/11851561_33

Bullnheimer, B., Hartl, R. F., & Strauß, C. (1997). *A new rank based version of the ant system: a computational study*. Wien: Vienna University of Economics and Business Administration.

Campos, L. M. D., Fernández-Luna, J. M., Gámez, J. A., & Puerta, J. M. (2002). Ant colony optimization for learning Bayesian networks. *International Journal of Approximate Reasoning*, *31*(3), 291–311. doi: 10.1016/s0888-613x(02)00091-9

Colorni, A., Dorigo, M., & Maniezzo, V. (1991). Distributed Optimization by Ant Colonies. *Proceedings of the First European Conference on Artificial Life*, 134–142.

Concorde Solver. (n.d.). Retrieved from http://www.math.uwaterloo.ca/tsp/concorde.html

Cordon O., Fernandez de Viana, I., Herrera F., and Moreno, L. (2000). A new ACO model integrating evolutionary computation concepts: The best-worst Ant System. *Abstract proceedings of ANTS 2000 – From Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms, IRIDIA, Universite Libre de Bruxelles* 22–29.

Deng, W., Xu, J., & Zhao, H. (2019). An Improved Ant Colony Optimization Algorithm Based on Hybrid Strategies for Scheduling Problem. *IEEE Access*, *7*, 20281–20292. doi: 10.1109/access.2019.2897580

Dong, G., Li, W., Shen, J., Wang, Y., Fu, X., & Guo, W. W. (2018). Solving Traveling Salesman Problems with Ant Colony Optimization Algorithms in Sequential and Parallel Computing Environments: A Normalized Comparison. *International Journal of Machine Learning and Computing, 8*(2), 98-103. doi:10.18178/ijmlc.2018.8.2.670

Dorigo, M., & Gambardella, L. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, *1*(1), 53–66. doi: 10.1109/4235.585892

Dorigo, M., & Stützle, T. (2018). Ant Colony Optimization: Overview and Recent Advances. *Handbook of Metaheuristics International Series in Operations Research & Management Science*, 311–351. doi: 10.1007/978-3-319-91086-4_10

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics),26*(1), 29-41. doi:10.1109/3477.484436

Dorigo, M., Stützle, T. (2004). Ant Colony Optimization. Massachusetts Institute of Technology

Gambardella, L. M., & Dorigo, M. (1995). Ant-Q: A Reinforcement Learning approach to the traveling salesman problem. *Machine Learning Proceedings 1995*, 252–260. doi: 10.1016/b978-1-55860-377-6.50039-6

Karapetyan, D., & Gutin, G. (2011). Lin–Kernighan heuristic adaptations for the generalized traveling salesman problem. *European Journal of Operational Research,208*(3), 221-232. doi:10.1016/j.ejor.2010.08.011

Lessing, L., Dumitrescu, I., & Stützle, T. (2004). A Comparison Between ACO Algorithms for the Set Covering Problem. *Ant Colony Optimization and Swarm Intelligence Lecture Notes in Computer Science*, 1–12. doi: 10.1007/978-3-540-28646-2_1

Lin, S., & Kernighan, B. W. (1973). An Effective Heuristic Algorithm for the Traveling-Salesman Problem. Operations Research, 21(2), 498-516. doi:10.1287/opre.21.2.498

Manfrin, M., Birattari, M., Stützle, T., & Dorigo, M. (2006). Parallel Ant Colony Optimization for the Traveling Salesman Problem. Ant Colony Optimization and Swarm Intelligence Lecture Notes in Computer Science, 224-234. doi:10.1007/11839088_20

Maniezzo, V. (1999). Exact and Approximate Nondeterministic Tree-Search Procedures for the Quadratic Assignment Problem. *INFORMS Journal on Computing*, *11*(4), 358–369. doi: 10.1287/ijoc.11.4.358

Nilsson, C. (2003). Heuristics for the traveling salesman problem. *Technical report, Linkoping University*

Peake, J., Amos, M., Yiapanis, P., & Lloyd, H. (2019). Scaling techniques for parallel ant colony optimization on large problem instances. Proceedings of the Genetic and Evolutionary Computation Conference. doi:10.1145/3321707.3321832

Silva, R. D. A., & Ramalho, G. (2001). Ant system for the set covering problem. *2001 IEEE International Conference on Systems, Man and Cybernetics. e-Systems and e-Man for Cybernetics in Cyberspace (Cat.No.01CH37236)*. doi: 10.1109/icsmc.2001.971999

Sedgewick, R. & Wayne, K. (2019). SparseMatrix and SparseVector source code [Source code]. https://introcs.cs.princeton.edu/java/44st/SparseMatrix.java.html

Steinerberger, S. (2015). New Bounds for the Traveling Salesman Constant. *Advances in Applied Probability*, *47*(01), 27–36. doi: 10.1017/s0001867800007680

Stützle, T., & Hoos, H. (1997). MAX-MIN Ant System and local search for the traveling salesman problem. *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC 97)*. doi: 10.1109/icec.1997.592327

Stützle, T., & Hoos, H. (1998). Improvements on the Ant-System: Introducing the MAX-MIN Ant System. *Artificial Neural Nets and Genetic Algorithms*, 245–249. doi: 10.1007/978-3-7091-6492-1_54

Stützle, T., Dorigo, M. (1999). ACO algorithms for the traveling salesman problem. *Evolutionary Algorithms in Engineering and Computer Science*, 163-183
Stützle, T.

TSPLIB. (n.d.). Retrieved from http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/

TSP Art Instances. (n.d.). Retrieved from http://www.math.uwaterloo.ca/tsp/data/art/index.html

Wang, J., Cao, J., Sherratt, R. S., & Park, J. H. (2017). An improved ant colony optimization-based approach with mobile sink for wireless sensor networks. *The Journal of Supercomputing*, *74*(12), 6633–6645. doi: 10.1007/s11227-017-2115-6

Wilke, Adrian. (n.d.). Retrieved from https://github.com/adibaba/ACOTSPJava/

Zhou, Y., He, F., & Qiu, Y. (2017). Dynamic strategy based parallel ant colony optimization on GPUs for TSPs. *Science China Information Sciences*, *60*(6). doi: 10.1007/s11432-015-0594-2

Zhou, Y., He, F., Hou, N., & Qiu, Y. (2018). Parallel ant colony optimization on multi-core SIMD CPUs. *Future Generation Computer Systems*, *79*, 473–487. doi: 10.1016/j.future.2017.09.073