Analysis of Ant Colony Optimization for Dynamic Shortest Path Problems

Andrei Lissovoi

Kongens Lyngby 2012
IMM-M.Sc.-2012-104

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Preface

This thesis was prepared at the department of Informatics and Mathematical Modelling at the Technical University of Denmark in fulfillment of the requirements for acquiring an M.Sc. in Computer Science and Engineering. It was supervised by associate professor Carsten Witt, and assigned a workload of 30 ECTS credits.

The thesis examines how nature-inspired algorithms based on the Ant Colony Optimisation metaheuristic are able to solve dynamic shortest path problems.

Source code for the software developed for this thesis has been submitted electronically, and can also be extracted from the PDF version by viewers that support file annotations. Extract source code (ZIP archive)

Lyngby, September 2, 2012,

Andrei Lissovoi

Summary

Combinatorial optimisation problems have traditionally been solved by specialized algorithms. Such problems also occur abundantly in nature, where they are solved without requiring excessive amounts of computing power or explicit algorithm design. Nature-inspired algorithms are based on behavior observed in nature, and are able to solve a wide variety of combinatorial optimisation problems without requiring much adaptation to a particular problem. Ant Colony Optimisation (ACO) is a metaheuristic encompassing a family of nature-inspired algorithms that are based on the foraging behavior of ants – using simulated pheromone trails to influence construction of random solutions to a problem, and updating the pheromone values to favor constructing good solutions over time.

This thesis considers how variants of the Max-Min Ant System algorithm, based on the ACO metaheuristic, are able to solve dynamic shortest path problems. Known results bounding its expected run time on static shortest path problems are presented and applied to rediscovering the shortest paths after a one-time change is made to the graph, showing $O(n^3)$ and $\Omega(n^3)$ upper and lower bounds (where n is the number of vertices in the graph) on the expected number of iterations to rediscover the shortest paths after specific one-time changes to the weight function.

It is then shown that the λ-MMAS algorithm, which constructs λ solutions in a single iteration in expectation requires fewer iterations to find the shortest paths, and, if parallel computation resources are available, also requires less running time. This approach is shown to reduce the expected number of iterations to $O(n)$ while only requiring a polynomial number of ants (with respect to n) to be started at each vertex. Using additional ants is also shown to allow iteration-best pheromone reinforcement, where the colony reinforces the best paths constructed in the current iteration only.

Patterns of periodic changes to the graph are then considered, and it is shown that when the changes to the shortest paths specified by the weight functions are relatively minor (adding or removing few arcs), λ-MMAS is able to keep track of the shortest paths reliably when $\log^2 n$ ants are started at each vertex as long as the pheromone evaporation rate is not too high. When the changes to the shortest paths are more significant, it is shown that $\log^2 n$ ants are no longer sufficient.

The considered algorithms are implemented in C. This implementation is used to perform experiments to supplement the analytical results with more detailed empirical data for small problem sizes.

Contents

Preface iii

Summary iv

Contents v

1 Introduction 1
- 1.1 Max-Min Ant System . . . 3
  - 1.1.1 Choosing pheromone bounds . . . 6
  - 1.1.2 Freezing time . . . 8

2 Updating after a one-time change to the graph 10
- 2.1 An upper bound on expected optimisation time . . . 11
- 2.2 A lower bound on expected optimisation time . . . 13

3 Using multiple ants 16
- 3.1 Multiple ants in best-so-far reinforcement . . . 17
- 3.2 Iteration-best reinforcement . . . 19
  - 3.2.1 Constant evaporation rate . . . 22
  - 3.2.2 Low evaporation rates . . . 24

4 Periodic changes 26
- 4.1 Tracking a fast oscillation . . . 27
- 4.2 Low evaporation rates . . . 29
- 4.3 Impact of oscillating component size . . . 31

5 Implementation and Experiments 34
- 5.1 Implementation overview . . . 34
- 5.2 Experiments . . . 36
  - 5.2.1 Recovering from a one-time change . . . 36
  - 5.2.2 Tracking a fast oscillation . . . 37
  - 5.2.3 Effects of oscillating component size . . . 37

6 Discussion 42
- 6.1 Resume . . . 42
- 6.2 Future work . . . 43

References 45

A Implementation details 47
- A.1 Using the implementation . . . 47
- A.2 Graph format example . . . 47
- A.3 Functions available to the Lua environment . . . 48

1 Introduction

Optimisation problems are omnipresent in both nature and human civilization. In the latter, where they might touch upon technological, economic or social aspects of our lives, they are typically solved using specialized algorithms, traditionally designed by analyzing the underlying mathematical properties of specific problems. Such algorithms are then formally evaluated to ensure that they produce correct solutions to the the problem they're designed to solve within desired time and memory constraints.

In nature, optimisation problems are just as prevalent – living organisms adapt to new environments through genetic mutation and natural selection, fish swim in schools to conserve energy, and ants construct efficient routes to food sources using pheromone trails. In all of these examples no algorithms have been formally designed and verified, and yet a reasonable solution to an optimisation problem allows life to thrive. These examples have been used as inspiration in algorithm design: Evolutionary Algorithms adapt random mutation and selection, Particle Swarm Optimisation algorithms are based on flocking behaviors, and Ant Colony Optimisation algorithms mimic the implicit memory of pheromone trails to guide random walks through a graph towards the optimal solution. Some of these examples of nature-inspired algorithms, as well as many others, are described in further detail in [10] and [12].

Nature-inspired algorithms are often popular due to relative ease of implementation, wide applicability and reasonable quality of solutions produced for many optimisation problems. Algorithms inspired by ant behavior have been proposed as early as 1992 in [2], and were formalized as the Ant Colony Optimisation metaheuristic in [3]. The metaheuristic has since been shown to be applicable to a wide variety of practical combinatorial optimisation problems, see e.g. [4]. More formal theoretical evaluations of ACO-based algorithms have been developed recently, an overview of which can be found in [8] and [13], and formal analysis of nature-inspired algorithms is still a relatively new subfield of computer science. This thesis examines the performance of several algorithms based on the ACO metaheuristic on dynamic shortest path problems.

The problem of finding an optimal way to reach a particular goal occurs in many natural contexts, and is perhaps most familiar when framed as a navigation problem: finding the shortest, quickest, simplest, safest, or cheapest way to travel between two physical locations using some mode of transport. Shortest path problems also occur in non-navigational contexts, like solving a Rubik's cube, devising the fastest way to prepare a three-course dinner, or forwarding messages in communication networks to minimize delivery time, information loss, or chance of eavesdropping.

In general, a shortest path problem involves a finding a path between two vertices s and t in a weighed directed graph $G = (V, \mathcal{A})$ with the minimum total weight according to a weight function $w : \mathcal{A} \to \mathbb{R}$. More complex variations of the problem require finding the shortest path to all vertices from a given source vertex s (single-source shortest path problems), or from all vertices to a given destination vertex t (single-destination shortest path problems), or even between all pairs of vertices in the graph (all-pairs shortest path problems). Notably, the single-source and single-destination variations are equivalent: a single-source shortest path problem can be solved by reversing the direction of all arcs in G and using a single-destination shortest path algorithm on the resulting graph, and vice versa.

While the weight function w may assign any real weight to any arc in the graph, G should not have cycles where the sum of arc weights is negative. This constraint is a consequence of the defining a shortest path as the path with the minimum total weight. If cycles of negative weight exist in the graph, and v is a vertex in such a cycle, then any path from v to any other vertex w can be "shortened" by prefixing it with the negative-weight cycle from v to v. Thus, if negative-weight cycles are permitted, the shortest paths between some pairs of vertices may have both infinite length (number of arcs in the path), and infinitely negative total weight.

There are well-known deterministic algorithms capable of solving shortest path problems in a directed graph with n vertices and m arcs. The single-source variant (and hence also the single-destination) with negative arc weights can be solved using the Bellman–Ford algorithm in $O(nm)$ time (i.e. $O(n^3)$ for complete graphs), while the all-pairs variant can be solved using the Floyd–Warshall algorithm in $O(n^3)$ time.

Dynamic variants of shortest path problems allow the weight function to be changed while the shortest paths are being computed, or after they have been computed. This might occur in shortest path problems dealing with navigation when the weight function reflects travel time (which may be affected by traffic), or travel cost (where prices may fluctuate based on demand). There exist algorithms that are able to re-use some of the already computed shortest paths to speed up the processing of the updated graph, as discussed in [7].

The performance of ACO-based algorithms on single-destination and all-pairs variants of shortest path problems has been analyzed in [18]. Results include $O\left(\Delta\ell \max(\ell, \ln n) + \ell/\rho\right)$ and $O\left(n \log n + \frac{\log(\ell)\log(\Delta\ell)}{\rho}\right)$ bounds on the expected number of iterations to solve the single-destination and the all-pairs variants respectively, where Δ is the maximum vertex out-degree and ℓ is the maximum length of any shortest path. Notably, the latter bound achieves an

improvement over simply running n single-destination instances in parallel by allowing those instances to share information.

Ant colony optimisation algorithms are able to continue the optimisation process even after the graph changes, potentially benefitting from some of the progress made previously – and for minor changes in the graph, they may achieve better performance than algorithms that have to start from scratch whenever the graph is modified.

The remainder of this section introduces the $\text{MMAS}_{\text{SDSP}}$ ACO algorithm, as well as considerations about the choice of parameters that influence further analysis. Section 2 proves upper and lower bounds on the number of iterations required for $\text{MMAS}_{\text{SDSP}}$ to recompute the shortest paths after a one-time change to the graph, demonstrating that the number of arcs changed, and the magnitude of the arc weight change do not necessarily predict the amount of additional iterations required. Section 3 examines the effects of simulating increased number of ants per iteration of the algorithm. Section 4 explores how well ACO-based algorithms handle periodic changes to the weight function. Section 5 discusses the implementation of an ACO-based single-destination shortest path solver and presents some experimental results, shedding further light on the effects of parameter choice in various situations. Finally, Section 6 presents a summary of the analytical and experimental results, as well as a discussion of topics that could be further expanded upon.

1.1 Max-Min Ant System

Ant Colony Optimisation algorithms are based on the observed behavior of ants foraging for food. Upon locating a food source, an ant returns to the colony while leaving a pheromone trail leading towards the food source. Other ants can sense these pheromone trails, and, upon returning from the food source, can reinforce the trail further, potentially improving the found path through random variations. Pheromone trails that are not reinforced will eventually evaporate, ensuring that if a source of food is exhausted, ants will no longer be attracted to it.

The $\text{MMAS}_{\text{SDSP}}$ algorithm considered in [18], shown as Algorithm 1 below emulates this behavior by associating a pheromone τ_a value with each arc a in the graph, and simulating a number of virtual ants. For dynamic shortest path problems, the fitness value $f^{(i)}(x)$ of any path x ending at t is simply the sum

of the arc weights in the path per the iteration-determined weight function $w^{(i)}$:

$$f^{(i)}(x) = \begin{cases} \sum_{a \in x} w^{(i)}(a) & \text{if } x \text{ ends at } t \\ \infty & \text{otherwise} \end{cases}$$

Algorithm 1 The $\text{MMAS}_{\text{SDSP}}$ algorithm on a directed graph $G = (V, \mathcal{A})$, with pheromone bounds $\tau_{\min}$ and $\tau_{\max}$, and evaporation rate ρ. The functions $\text{tail}(a)$ and $\text{head}(a)$ denote the start and end vertices of an arc a, while $\deg^+(v)$ denotes the out-degree of a vertex.

Initialize $\tau_a \leftarrow 1/\deg^+(\text{tail}(a))$ for all $a \in \mathcal{A}$
for $i \leftarrow 1, 2, \ldots$ **do**
  for each $v \in V$ **do**
    Let x_v be a new path starting at v
    $p \leftarrow v$, $S \leftarrow \{a \in \mathcal{A} \mid \text{tail}(a) = p \wedge \text{head}(a) \notin x_v\}$
    while S is not empty and $p \neq t$ **do**
      Select an arc a from S, with probability:
        $p_a = \tau_a / \sum_{s \in S} \tau_s$
      Append a to x_v
      $p \leftarrow \text{head}(a)$, $S \leftarrow \{a' \in \mathcal{A} \mid \text{tail}(a') = p \wedge \text{head}(a') \notin x_v\}$
    if $i = 1$ or $f^{(i)}(x_v) < f^{(i)}(x_v^*)$ **then**
      $x_v^* \leftarrow x_v$ ▷ Update the best-so-far path
  for each $a \in \mathcal{A}$ **do**
    $\tau_a \leftarrow \begin{cases} \min(\tau_{\max}, (1-\rho)\tau_a + \rho) & \text{if } a \in x^*_{\text{tail}(a)} \\ \max(\tau_{\min}, (1-\rho)\tau_a) & \text{otherwise} \end{cases}$

The pheromones values are initialized such for each vertex in the graph, the sum of the pheromone values on outgoing arcs is 1, and all outgoing arcs have equal pheromone values.

In an iteration of the algorithm, a virtual ant is started at each vertex v of the graph, and constructs a simple path through the graph randomly, until it either reaches the destination vertex t or is unable to continue further. For each vertex v, the colony keeps track of the best-so-far path x_v^*: the shortest path from v to t it has constructed in the past.

Once all ants in a given iteration have been simulated, and potentially updated x_v^* paths, the pheromone values are updated in order to make future ants visiting each vertex v more likely to choose to follow the arc belonging to x_v^*. In dynamic shortest paths problems, the fitness value $f^{(i)}(x_v^*)$ needs to be reevaluated whenever the weight function of the graph is altered, or the algorithm is no longer correct. This is easily illustrated by altering the weight function so as to change the first arc on the shortest path from a given vertex, and making

the new shortest path have a larger weight than the old shortest path – if the fitness value is not reevaluated, the new shortest path will never replace the old x_v^*. Reevaluating the fitness values of the best-so-far paths is also common in other contexts, for instance when dealing with stochastic fitness functions, as examined further in [14] and [15].

The evaporation of natural pheromone trails is modeled in the algorithm by the parameter $0 < \rho \leq 1$, which controls the speed with which the pheromone values τ are updated. Setting $\rho = 1$ would enable the ant colony to instantly change the pheromone values to their bounds upon discovering a new shortest path. Lower values allow information about previous shortest paths to persist in pheromone values for some number of iterations, which may be useful if the weight function changes in a periodic manner, as discussed further in Section 4.

To analyze the performance of this ACO algorithm, we consider the number of expected iterations of the outer loop required for all of the best-so-far paths x_v^* to be set to an actual shortest path from their starting vertices. This is somewhat different from the traditional running-time analysis of deterministic algorithms; for ACO, the inner loop is considered to take $O(1)$ time, as the ants are independent and can be simulated in parallel, and, traditionally, the cost of evaluating the fitness function is considered to dominate that of the other operations. These considerations make the bounds on expected running time not directly comparable to the run-time bounds of deterministic algorithms. In the worst case, each iteration of the algorithm involves $2n$ fitness function evaluations, or $O(n^3)$ basic operations in total, as all but one of the n simulated ants may need to examine the pheromone value on each one of $m = O(n^2)$ arcs in the graph in order to construct a path to the destination vertex.

$\text{MMAS}_{\text{SDSP}}$ solves the single-destination shortest path variant of the problem – the virtual ants keep track of the best-so-far path from each vertex, and will eventually find the shortest path from each starting vertex. However, the $|V|$ ants are actually required even to solve the simpler single shortest path variant, as illustrated in [18] using the graph shown in Figure 1.

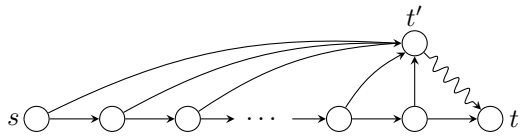


Figure 1: When the (t', t) arc has weight n, and all other arcs have weight 1, the shortest path from s to t in this graph avoids t'.

Proof If a single ant were to start in the vertex s, it would follow an arc to t' with probability $1/2$ from each vertex it visits. The real shortest path from s to t involves visiting $n-2$ vertices with an outgoing arc to t', and never visiting t' itself. With probability $1-2^{-n/2}$, the ant will deviate from the correct shortest path after no more than $n/2$ arcs. In future iterations, only paths with less weight than this original path will be reinforced – so unless all of the remaining $n/2-2$ arcs are selected simultaneously, a solution still passing through t' will be reinforced. Thus, the ant must pick at least those $n/2-2$ arcs in a single iteration in order to discover the shortest path, which occurs with probability at most $2^{-n/2+2} = 2^{-\Omega(n)}$. The probability that an outcome that occurs with probability p occurs for the first time after k trials is described by the geometric distribution, the expectation of which is $1/p$ – thus, the expected number of iterations for the correct shortest path to be discovered is at least $2^{\Omega(n)}$. This shows that with only a single ant, finding the shortest path in the graph shown in Figure 1 will require $2^{\Omega(n)}$ iterations in expectation. In addition, a union bound can used to show that $2^{n/4}$ iterations are insufficient with overwhelming probability – the shortest path will be found with probability at most $2^{n/4} \cdot 2^{-n/2+2} = 2^{-n/4+2} = 2^{-\Omega(n)}$. □

Starting an ant at each vertex addresses this problem by ensuring that a shortest path can eventually be discovered by ants constructing a path with no more than one arc with a low pheromone value at a time.

1.1.1 Choosing pheromone bounds

The pheromone bounds $\tau_{\min}$ and $\tau_{\max}$ serve to bound the total pheromone value, τ_{sum}, on outgoing arcs from any vertex in the graph. This ensures that there is always a positive probability for an ant to select any specific outgoing arc, or construct any simple path through the graph, guaranteeing that $\text{MMAS}_{\text{SDSP}}$ will eventually discover any shortest path. In particular, $\tau_{\text{sum}} \leq 1 + \Delta\tau_{\min}$ is shown in [18] using induction: $\tau_{\text{sum}} = 1$ initially, and the most τ_{sum} can increase is as a consequence of ρ being added to some arc and none of outgoing arcs being affected by pheromone evaporation due to being bounded by $\tau_{\min}$:

$$(1-\rho)(1+\Delta\tau_{\min}) + \rho + \rho\Delta\tau_{\min} = 1 + \Delta\tau_{\min}$$

thus, τ_{sum} will never exceed $1 + \Delta\tau_{\min}$.

I will now show that this bound on τ_{sum} can be refined by examining the pheromone update rules more closely. For instance, the pheromone value on a single arc can never exceed 1, as the evaporation exactly cancels out the

reinforcement at that value. Additionally, the pheromone sum will not increase from its initial value of 1 until some of the pheromones start being affected by the $\tau_{\min}$ lower bound, at which point reinforcing the arcs not affected by the lower bound would increase the sum further. This leads to a strategy that maximizes τ_{sum} by continuously reinforcing a single arc, letting the pheromones on the other $\Delta - 1$ arcs evaporate down to $\tau_{\min}$, which yields a tighter bound bound:

$$\tau_{\text{sum}} \leq 1 + (\Delta - 1)\tau_{\min}$$

This bound holds regardless of the which pheromone reinforcement strategy is chosen.

Proof Suppose for a contradiction that a different strategy exists that does increase τ_{sum} further. Observe that reinforcing the arc with the highest pheromone value will never *reduce* τ_{sum} if the pheromone value on that arc does not exceed 1 (which is necessarily the case). By repeatedly reinforcing the highest pheromone value arc after performing that strategy, the pheromone values on all other arcs will eventually converge to $\tau_{\min}$ – and therefore τ_{sum} will be no greater than the above bound, but also *no less* than it would've been after performing that strategy. This is a contradiction – τ_{sum} was initially claimed to be greater than the bound, but may not be greater after a number of iterations that do not reduce it. Therefore, the above bound on τ_{sum} holds regardless of how the reinforced arcs are selected. □

The pheromone values are chosen so as to control the probabilities of selecting specific arcs with different pheromone values. Let $p_{\max}$ be the probability of an ant selecting an outgoing arc with pheromone value $\tau_{\max}$ (a reinforced arc), and let $p_{\min}$ be the probability of an ant selecting an outgoing arc with pheromone value $\tau_{\min}$. This also makes $p_{\min}$ a lower bound on the probability of selecting an arbitrary non-reinforced arc, which has the pheromone value $\tau_{\min} \leq \tau < \tau_{\max}$.

In particular, $p_{\max}$ should be large enough to allow an ant to construct any shortest path in the graph, with at least constant probability when all of its arcs are reinforced (i.e. have pheromone value $\tau_{\max}$). Let the *length* of a path refer to the number of arcs in the path (as opposed to the sum of their weights), and $\ell < n$ be the length of the longest shortest path to t in the graph. The requirement is then that $(p_{\max})^{\ell}$ is at least a positive constant, which can be achieved by exploiting $(1 - 1/n)^{n-1} \geq e^{-1}$, or $(1 - 1/\ell)^{\ell} \geq 1/4$ (for $\ell \geq 2$). Conventionally, bounds are chosen such that $\tau_{\min} + \tau_{\max} = 1$, and

using $p_{\max} \geq \tau_{\max}/\tau_{\text{sum}}$ yields:

$$\begin{aligned} \tau_{\max}/\tau_{\text{sum}} &\geq 1 - 1/\ell \\ 1 - \tau_{\min} &\geq (1 - 1/\ell)(1 + (\Delta - 1)\tau_{\min}) \\ 1/\ell &\geq \tau_{\min}(\Delta - (\Delta - 1)/\ell) \\ \tau_{\min} &\leq 1/(\ell\Delta) < 1/(\ell\Delta - (\Delta - 1)) \end{aligned}$$

Picking $\tau_{\min} = 1/(\ell\Delta)$ ensures $\tau_{\max}/\tau_{\text{sum}} \geq 1 - 1/\ell$, and ants are therefore able to follow every pheromone-reinforced shortest path with constant probability. In situations when ℓ or Δ are not known in advance, the upper bounds $\ell < n$ and $\Delta < n$ (in graphs without multiple edges) can be used instead, so $\tau_{\min} = n^{-2}$ would be sufficient to ensure the desired property for any such graph. The effects of this choice of pheromone bounds are summarized in the following Lemma, which is similar in purpose to Corollaries 2 and 3 in [18].

Lemma 1 *The pheromone bounds $\tau_{\min} \leq 1/(\ell\Delta)$ and $\tau_{\max} = 1 - \tau_{\min}$ ensure that the probability $p_{\max}$ of selecting a reinforced arc, with pheromone value $\tau_{\max}$, is at least $(1 - 1/\ell)$, and the probability $p_{\min}$ of selecting an arbitrary non-reinforced arc, is between $\tau_{\min}/2$ and $\tau_{\min}$.*

Proof The first part of the Lemma, handling $p_{\max}$, stems directly from the bound imposed on $\tau_{\min}$ by the considerations above. The case for $p_{\min}$ is simpler, as subsequent proofs do not rely on an ant following a path composed of non-reinforced arcs, so a simple bound based on $p_{\min} \geq \tau_{\min}/\tau_{\text{sum}}$ is enough:

$$\begin{aligned} p_{\min} &= \tau_{\min}/\tau_{\text{sum}} \geq \tau_{\min}/2 \\ p_{\min} &\leq \tau_{\min} \end{aligned}$$

as for $\tau_{\min} \leq 1/(\ell\Delta)$, $\tau_{\text{sum}} \leq 1 + (\Delta - 1)\tau_{\min} < 2$, and $\tau_{\text{sum}} \geq 1$ for any vertex with multiple outgoing arcs. □

1.1.2 Freezing time

When x_v^* is updated to follow a different arc from of v, pheromone values on arcs leaving v will change over time, governed by the pheromone evaporation rate ρ. If enough iterations pass without a new x_v^* being discovered, the pheromone values will reach the pheromone bounds, becoming *frozen*: they will not be altered further unless x_v^* changes. The concept of *freezing time*, the number of

iterations until the pheromones become frozen, occurs frequently in literature analyzing performance of ACO, as reasoning about the probability of making progress is easier when the pheromones are bounded. The following Lemma from [6] can be used to bound the number of iterations required for the pheromones to become frozen.

Lemma 2 *If the path x_v^* remains unchanged for $O(\log(\tau_{\max}/\tau_{\min})/\rho)$ iterations, the pheromones on arcs leaving v become frozen.*

Proof Consider a situation when pheromone values are already frozen, but a new x_v^* is discovered requiring them to change. The change will be limited to pheromone values on two arcs: the old $\tau_{\max}$ arc being reduced to $\tau_{\min}$, and first arc in x_v^*, being increased from $\tau_{\min}$ to $\tau_{\max}$. As pheromone bounds are chosen such that $\tau_{\max} + \tau_{\min} = 1$, the sum of pheromone values on the two arcs is initially 1, and this property is preserved by the pheromone update mechanism. It is therefore sufficient to consider the number of iterations required to reduce the $\tau_{\max}$ pheromone value to $\tau_{\min}$ by multiplying with $(1 - \rho)$; for instance $\ln(\tau_{\max}/\tau_{\min})/\rho$, as suggested by the proof in [6]:

$$\tau_{\max}(1-\rho)^{\ln(\tau_{\max}/\tau_{\min})/\rho} < \tau_{\max} e^{-\ln(\tau_{\max}/\tau_{\min})} = \tau_{\min}$$

by noting that $(1 - x)^x \leq 1 < e$ for $x \leq 1$.

Altering the pheromone values from one bound to the is the maximum amount of change that might be required – if the pheromone values were not frozen when a new x_v^* is discovered, the pheromone values on old/new arcs are no further from their goal values than they would be if the old values were frozen. Therefore, $\ln(\tau_{\max}/\tau_{\min})/\rho$ iterations without discovering a new x_v^* are sufficient to ensure that pheromones on arcs leaving v are frozen. □

2 Updating after a one-time change to the graph

In a dynamic system, the weights assigned to the arcs of the graph might change – for example, the weights might represent travel times between various destinations in a road network affected by traffic, or the delivery time of packets between various destinations in a computer network. This part of the report examines how a one-time modification of the weight function affects $\text{MMAS}_{\text{SDSP}}$, and establishes upper and lower bounds on the amount of time needed to compute the new shortest paths after a change to the weight function is made.

An interesting result that will be demonstrated is that the magnitude of the change in the weight function (from both the perspective of the number of arcs affected and the total weight change) does not predict the amount of iterations required to rediscover the shortest paths – just as the entire weight function may change without changing *any* of the computed shortest paths, a small change in a single weight may require almost all shortest paths to be rediscovered. Additionally, the number of modified shortest paths also does not provide a clear indication of the number of iterations it might take $\text{MMAS}_{\text{SDSP}}$ to recompute them, as a large number of paths may or may not be updated in parallel depending on the new weight function.

One of the mentioned cases – changing the weights of all arcs while not changing any of the shortest paths – is trivial to imagine: simply multiply the weights of all arcs by a constant $k > 0$. This alters all non-zero arc weights, yet preserves the shortest paths, as the total weight of any path is also simply multiplied by k, so if a path is shorter than another path after the transformation, it would also have been shorter before this transformation. Thus, there's a way to change all arc weights in a graph (as long as they were initially non-0), by an arbitrarily large amount, without changing any of the shortest paths.

The other cases can be illustrated by using the graph used to demonstrate the need for using an ant *colony*, repeated in Figure 2, and the two weight functions w_1 and w_2 shown in (1) below, where $\epsilon > 0$ is a small constant. The shortest paths through the graph using the w_1 weight function avoid taking any arcs to t', while the shortest paths through the graph using the w_2 weight function always immediately visit t'.

$$w_1(a) = \begin{cases} n \cdot \epsilon & \text{if } a = (t', t) \\ \epsilon & \text{otherwise} \end{cases} \qquad w_2(a) = \begin{cases} -\epsilon & \text{if } a = (t', t) \\ \epsilon & \text{otherwise} \end{cases} \tag{1}$$

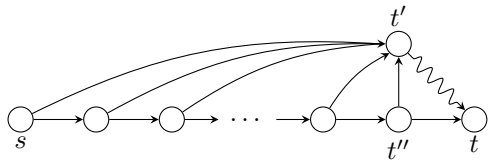


Figure 2: Increasing or decreasing the weight of the wavy (t', t) arc in the above graph results in different optimisation times for $\text{MMAS}_{\text{SDSP}}$.

2.1 An upper bound on expected optimisation time

The worst-case update performance occurs when a change in the weight function alters a large number of shortest paths, and $\text{MMAS}_{\text{SDSP}}$ is unable to rediscover many paths in parallel. The situation is similar to the pessimistic assumptions used to prove an upper bound on the expected optimisation time after the pheromone values have been initialized, and (pessimistically) frozen without discovering any shortest paths. The following theorem states an upper bound result, which is then proven using the same approach as Theorem 4 in [18], which shows an upper bound on expected optimisation time after pheromone initialization[1].

Theorem 3 *The expected number of iterations required by $MMAS_{SDSP}$ to rediscover all shortest paths after a one-time change to the weight function is $O(\ell/\tau_{\min} + \ell \log(\tau_{\max}/\tau_{\min})/\rho)$, where ℓ is the maximum number of arcs in any shortest path to t in the new graph. This also bounds the expected number of iterations required to discover all shortest paths initially.*

Proof It is sufficient to demonstrate that there is a mechanism that would optimise the graph within this expected time. The vertices of the graph can be divided into classes according to the number of arcs on the actual shortest path to the destination vertex t from a given vertex: t itself is in class 0; vertices from which the shortest path to t involves taking a single arc are in class 1, and so on, up to class $\ell < n$. A mechanism that satisfies the theorem's bound simply processes the classes sequentially: it first finds the shortest paths for vertices in class 1, waits for the pheromone values to freeze, then continues to class 2, and eventually up to class n.

[1]The result is further improved in Theorem 7 of [18] by observing that the pheromones do not need to be completely frozen to make progress, which eliminates the $\log(\tau_{\max}/\tau_{\min})$ factor from the freezing time term. This refinement is omitted here to keep the presentation relatively concise.

As the classes are optimized by this process in order, when class i is being processed, the shortest path from any vertex in class i can be discovered by following a single arc with pheromone value at least $\tau_{\min}$ to the correct vertex in class $i-1$, and then following the already-known shortest path from that vertex, where all pheromone values have already been frozen to $\tau_{\max}$. Thus, the probability of a shortest path for a vertex in class i being discovered once all classes $j < i$ have been processed is at least $p_{\min} \cdot {p_{\max}}^{i-1}$. As the pheromone bounds are chosen in accordance with Lemma 1, ${p_{\max}}^{i-1}$ is lower-bounded by a positive constant (as $i - 1 < \ell$), and $p_{\min} > \tau_{\min}/2$. Using the expectation of a geometric distribution with probability $p = \Omega(\tau_{\min})$, the expected number of iterations before a shortest path from a vertex in class i is discovered is $O(1/\tau_{\min})$. After all the shortest paths in a given class are discovered, the pheromone values need to be frozen before beginning work on the next shortest class, which requires $O(\log(\tau_{\max}/\tau_{\min})/\rho)$ waiting time per Lemma 2.

$\text{MMAS}_{\text{SDSP}}$ would need to optimize exactly ℓ classes to discover all shortest paths in this fashion, so the total expected optimisation time is:

$$\begin{aligned} E(\text{Total optimisation time}) &\leq \sum_{i=1}^{\ell} E(\text{Time to optimise class } i) \\ &\leq \sum_{i=1}^{\ell} O(1/\tau_{\min}) + O(\log(\tau_{\max}/\tau_{\min})/\rho) \\ &\leq O(\ell/\tau_{\min} + \ell \log(\tau_{\max}/\tau_{\min})/\rho) \end{aligned}$$

which proves the theorem. □

Inserting $\tau_{\min}$ and $\tau_{\max}$, and the upper bounds $\ell < n$ and $\Delta < n$ yields a few potentially simplified expressions:

$$\begin{aligned} \tau_{\min} = 1/(\ell\Delta): &\quad O(\ell^2\Delta + \ell \log(\ell\Delta)/\rho) \\ \tau_{\min} = 1/(n\Delta): &\quad O(n^2\Delta + n \log(n\Delta)/\rho) \\ \tau_{\min} = 1/n^2: &\quad O(n^3 + n \log(n)/\rho) \end{aligned}$$

A notable consequence of this theorem is that if ℓ is low after the weight function update, $\text{MMAS}_{\text{SDSP}}$ will rediscover the shortest paths quickly. For a practical example of this, consider $\text{MMAS}_{\text{SDSP}}$ finding the shortest paths for the Figure 2 graphs using the w_1 weight function of (1), and a one-time change changing the weight function to w_2. In that case, the shortest paths would be rediscovered in $O(1/\rho)$ iterations, as $\Delta = 2$ and $\ell = 2$ using w_2.

On the other hand, if $\text{MMAS}_{\text{SDSP}}$ initially found the shortest paths for the w_2 function, and then a one-time change switched the weight function to w_1,

the upper-bound on the expected optimisation time is $O(n^2 + n\log(n)/\rho)$ (by observing that $\Delta = 2$). A lower-bound on the optimisation time is needed to assert that $\text{MMAS}_{\text{SDSP}}$ actually requires that many iterations in this situation.

2.2 A lower bound on expected optimisation time

To demonstrate a lower bound on the expected optimisation time, we will show that the mechanism described in the upper bound proof is responsible for making most of the progress during in the optimisation process. This is accomplished by showing that with at least constant probability, $\text{MMAS}_{\text{SDSP}}$ will only discover shortest paths with only one non-reinforced arc during the optimisation process.

Theorem 4 *Certain one-time changes to the weight function may require an expected $\Omega(\ell/\tau_{\min})$ iterations for $MMAS_{SDSP}$ to rediscover all shortest paths, where ℓ is the maximum number of arcs in any shortest path to t in the new graph.*

Proof Assume for the moment that the optimisation process really does proceed by finding the shortest paths in the order of increasing number of arcs, as in Theorem 3, and consider the number of iterations required to find the new shortest paths.

As before, there are ℓ classes of vertices for which a shortest path needs to be discovered, and thus ℓ optimisation phases. In each phase, a specific ant needs to pick a specific arc with pheromone value $\tau_{\min}$, and then follow a path of at most ℓ arcs where all pheromone values are $\tau_{\max}$. For a lower bound on the waiting time, consider the correct shortest path discovered once an ant selects the correct non-reinforced arc. Per Lemma 1, the probability $p_{\min}$ of selecting an arbitrary arc is at most $\tau_{\min}$, so a lower bound on the expected number of iterations T_i required to complete optimisation phase i is $1/\tau_{\min}$.

By assuming that pheromones are frozen immediately after a new shortest path is found (i.e. $\rho = 1$), ℓ phases of waiting for an event occurring with at most $1/\tau_{\min}$ probability result in an $\Omega(\ell/\tau_{\min})$ lower-bound on the expected optimisation time for the entire process, assuming the shortest paths are found in this order. It can then be shown that $\Omega(\ell/\tau_{\min})$ iterations are required with overwhelming probability.

First, consider T_i, the number of iterations spent waiting for the shortest path in class i to be discovered. The path can be discovered with probability at

most $1/\tau_{\min}$ in each iteration, so T_i is geometrically distributed. Assuming the graph is non-trivial, i.e. $\Delta \geq 2$, and hence $\tau_{\min} \leq 1/2$, yields:

$$P(T_i \geq 1/(2\tau_{\min})) = (1-\tau_{\min})^{1/(2\tau_{\min})} \geq (0.25)^{0.5} \geq 1/2$$

so with probability at least $1/2$, T_i is at least $\Omega(1/\tau_{\min})$ iterations.

To find all shortest paths, the shortest paths for vertices in ℓ different classes need to be found, and per the previous assumption specifying that the shortest paths must be found in order, this means that ℓ phases, each lasting at least $\Omega(1/\tau_{\min})$ iterations with probability at least $1/2$ must be performed. Chernoff bounds can be used to bound the combined run time of the algorithm.

Let I_i be the indicator event that $T_i \geq 1/(2\tau_{\min})$ – i.e. I_i is 1 with probability at least $1/2$, and 0 otherwise. Then, $N = \sum_{i=1}^{\ell} I_i$ is the number of phases that require more than $1/(2\tau_{\min})$ iterations, and, per the binomial distribution, $E(N) \geq \ell/2$: at least half the phases are expected to last at least that long. Using Chernoff bounds, it can then be shown that at least a quarter of the phases will require more than that number of iterations with overwhelming probability:

$$\begin{aligned} P\left(N < (1-\delta)\cdot E(N)\right) &< e^{-E(N)\cdot\delta^2/3} \\ P\left(N < \ell/4\right) &< e^{-\ell/24} \\ P\left(N > \ell/4\right) &> 1 - e^{-\Omega(\ell)} \end{aligned}$$

so with overwhelming probability, at least $\Omega(\ell/\tau_{\min})$ iterations (or, as $\ell = \Theta(n)$ in the worst case, $\Omega(n^2\Delta)$ iterations) are needed before all the shortest paths are found, assuming no shortest path is ever found before all of its shorter subpaths are found.

Is the considered order reasonable? In order to deviate from it, a shortest path containing at least two non-reinforced arcs needs to be discovered by some ant. The risk of this is greatest at the very beginning of the optimization process, when there exist undiscovered shortest paths with $2, 3, \ldots, \ell$ non-reinforced arcs. Using the same best-case considerations as before (following the desired number of non-reinforced arcs means finding the shortest path with probability 1), the probability p of an iteration discovering any of these undesirable paths can be bounded using a union bound:

$$p < {\tau_{\min}}^2 \left(1 + \tau_{\min} + {\tau_{\min}}^2 + \ldots + {\tau_{\min}}^{\ell-2}\right) < 2\cdot {\tau_{\min}}^2 \quad \text{as } \tau_{\min} \leq 1/2$$

The probability of no such paths being discovered in $0.5{\tau_{\min}}^{-2} - 1$ iterations is at least:

$$(1-p)^{0.5{\tau_{\min}}^{-2}-1} = \left(1 - 1/(0.5{\tau_{\min}}^2)\right)^{0.5{\tau_{\min}}^{-2}-1} \geq e^{-1}$$

Thus, with constant probability, the simulated ant colony discovers no paths with more than one non-reinforced arc in $\ell^2\Delta^2/2 - 1$ iterations; and if no such paths are discovered within $\Omega(\ell^2\Delta)$ iterations, the ant colony is not done. Therefore, the expected number of iterations required to find all shortest paths after the weight function is changed in a way that invalidates all ℓ shortest paths and does not allow those paths to be rediscovered in parallel, is at least $\Omega(\ell^2\Delta)$.

This approach assumes that paths in all classes are invalidated, so $\text{MMAS}_{\text{SDSP}}$ has to optimize each vertex class in sequence. It is possible to change the weight function to accomplish this invalidation – for instance, changing from weight function w_2 to weight function w_1 of (1) on the graph shown in Figure 2 does invalidate the shortest paths from all vertices except t and t', requiring re-discovery of shortest paths from length 1 up to length $\ell = n - 2$. □

This example serves to illustrate that even relatively minor changes to the weight function can cause the expected number of iterations for $\text{MMAS}_{\text{SDSP}}$ to rediscover the shortest path to be asymptotically equal to the upper bound. While Theorem 4 does not consider choices of ρ when freezing phases are non-instant, it has been shown in Theorem 12 of [18] that the freezing time also affects the lower bound on the expected number of iterations.

3 Using multiple ants

The previous section showed that $\text{MMAS}_{\text{SDSP}}$ solves the single destination shortest path problem in expected polynomial time by randomly constructing paths through the graph, and then updating the pheromones to make construction of shortest paths consisting of increasing number of arcs more likely. Essentially, the optimisation process consists of two types of phases – *discovery* phases, and *freezing* phases – which alternate until all shortest paths are found. This section examines how simulating additional ants can shorten the discovery phases. For the remainder of this section, assume that $\ell = n - 1$ – i.e. there are shortest paths to t with $1, 2, \ldots, n-1$ arcs, depending on the starting vertex.

During *discovery* phases, the pheromone values on the shortest paths already known by the ant colony are set to $\tau_{\max}$, allowing the construction of shortest paths with one additional non-reinforced arc in expected $\Theta({\tau_{\min}}^{-1})$ time. The colony can be thought of as "waiting" for an ant to randomly construct the "next" shortest path in order to continue the optimisation process. This does not necessarily mean that *all* pheromone values remain unchanged during discovery phases, as $\text{MMAS}_{\text{SDSP}}$ may still update the best-so-far paths x_v^* by discovering a shorter, but not the shortest path from v to t.

Once the real shortest path is constructed from a vertex v, the pheromone values on v's outgoing arcs are updated to favor the "correct" outgoing arc in a *freezing* phase. After no more than $\log(\tau_{\max}/\tau_{\min})/\rho$ iterations (as shown in Lemma 2), the pheromone value on the first arc of the discovered shortest path is set to $\tau_{\max}$, and future discovery phases can build upon this path as a known path.

Freezing phases can be avoided by setting $\rho = 1$, which ensures that the pheromone values are set to $\tau_{\min}$ and $\tau_{\max}$ immediately upon discovering a new best-so-far path. With the freezing phases shortened to requiring no iterations, $\text{MMAS}_{\text{SDSP}}$ is able to find the shortest paths through any graph in expected $O(n^3)$ iterations (for $\tau_{\min} \geq n^{-2}$). However, only n of these are "useful" in the sense of advancing the optimisation process – so the colony spends the majority of its expected running time waiting.

The n ants used by the $\text{MMAS}_{\text{SDSP}}$ algorithm are completely independent of each other, and can be simulated on n machines in parallel to improve the performance of the algorithm. This independence property also makes it possible to simulate *additional* ants if additional machines are available, starting multiple ants at each vertex, which increases the probability of discovering a new shortest path during any iteration, which in turn decreases the expected number of iterations before all shortest paths are found.

In some ways, this modification is similar to expanding the number of offspring individuals produced by the (1+1) EA to create a $(1+\lambda)$ and $(1,\lambda)$ EAs[2] to increase the amount of exploration performed by the algorithm before making a choice affecting the next iteration. In the case of the $(1+\lambda)$ EA, the extra offspring individuals may make a difference between exponential and polynomial expected running times on some fitness functions, such as those examined in [5]. However, as the upper bound of the n-ant variant of $\text{MMAS}_{\text{SDSP}}$ is polynomial to begin with, the performance improvements achieved by starting λ ants from each vertex are less dramatic.

3.1 Multiple ants in best-so-far reinforcement

The λ-MMAS algorithm, shown as Algorithm 2 below, is a simple modification of $\text{MMAS}_{\text{SDSP}}$ that starts λ ants at each vertex in each iteration. This modification increases the amount of exploration performed in a single iteration – making each iteration roughly equivalent to λ iterations of $\text{MMAS}_{\text{SDSP}}$ in terms of the probability of discovering new shortest paths with only a single non-reinforced edge.

Algorithm 2 The λ-MMAS algorithm on a directed graph $G = (V, \mathcal{A})$, with pheromone bounds $\tau_{\min}$ and $\tau_{\max}$, and evaporation rate ρ.

Initialize $\tau_a \leftarrow 1/\deg^+(\text{tail}(a))$ for all $a \in \mathcal{A}$
for $i \leftarrow 1, 2, \ldots$ **do**
  for each $v \in V$ **do**
    for $j = 1, 2, \ldots, \lambda$ **do**
      Let $x_{v,j}$ be a new path starting at v
      $p \leftarrow v$, $S \leftarrow \{a \in \mathcal{A} \mid \text{tail}(a) = p \wedge \text{head}(a) \notin x_{v,j}\}$
      while S is not empty and $p \neq t$ **do**
        Select an arc a from S, with probability:
          $p_a = \tau_a / \sum_{s \in S} \tau_s$
        Append a to $x_{v,j}$
        $p \leftarrow \text{head}(a)$, $S \leftarrow \{a' \in \mathcal{A} \mid \text{tail}(a') = p \wedge \text{head}(a') \notin x_{v,j}\}$
    $x_v \leftarrow \arg\min_{x_{v,j}} f(x_{v,j})$
    if $i = 1$ or $f(x_v) < f(x_v^*)$ **then**
      $x_v^* \leftarrow x_v$
  for each $a \in \mathcal{A}$ **do**
    $$\tau_a \leftarrow \begin{cases} \min(\tau_{\max}, (1-\rho)\tau_a + \rho) & \text{if } a \in x^*_{\text{tail}(a)} \\ \max(\tau_{\min}, (1-\rho)\tau_a) & \text{otherwise} \end{cases}$$

[2]The $(1+\lambda)$ and $(1,\lambda)$ EAs create λ randomly mutated offspring before selecting the best one; the former includes the "parent" individual in the selection, while the latter does not.

Recall that the expected number of iterations for $\text{MMAS}_{\text{SDSP}}$ to discover the next shortest path after the pheromones are frozen is $O(1/\tau_{\min})$. Until such a path is discovered, the pheromones along that path remain at the same values as they were in the first iteration after the pheromones were frozen, making the probability of discovering a new shortest path in λ iterations of $\text{MMAS}_{\text{SDSP}}$ identical to the probability of discovering a new shortest path in a single iteration of λ-MMAS. Thus, by picking $\lambda = \Omega(1/\tau_{\min})$, λ-MMAS can discover new shortest paths in expected $O(1)$ iterations, reducing the total expected optimisation time to $O(\ell + \ell \log(\tau_{\max}/\tau_{\min})/\rho)$.

Notably, the freezing time remains unaffected by the modifications in λ-MMAS, and may even overtake the number of iterations required to *discover* shortest paths in the upper bound, as illustrated by the bound above.

The amount of ants can also be increased further, increasing the probability that the colony discovers paths with more non-reinforced edges in any given iteration. This approach is summarized by Theorem 5.

Theorem 5 *A single iteration of λ-MMAS has at least constant probability of discovering a new shortest path with k non-reinforced arcs if $\lambda = \Omega\left((2n^2)^k\right)$.*

Proof The probability that a single ant follows k non-reinforced arcs is at least ${p_{\min}}^k$, and the probability p_c of following the remaining reinforced arcs to the destination vertex is at least a constant (and with the typical choice of $\tau_{\min}$ and $\tau_{\max}$, $p_c \geq 1/e$), so the probability p_k of λ-MMAS discovering a shortest paths with k non-reinforced edges in a single iteration is (2), and by picking an appropriately large λ, p_k can be made at least a constant (3) regardless of input graph:

$$p_k = 1 - \left(1 - {p_{\min}}^k p_c\right)^\lambda \geq 1 - \left(1 - 1/(2n^2)^k \cdot p_c\right)^\lambda \tag{2}$$

$$\lambda = (2n^2)^k/p_c \Rightarrow p_k \geq 1 - e^{-1} \tag{3}$$

which proves the theorem. □

If λ-MMAS proceeds by discovering paths of length k in a constant number of iterations, it'll need to perform no more than ℓ/k discovery phases, reducing the expected number of iterations to find all shortest paths to $O(\ell/k + \ell/k \cdot \log(\tau_{\max}/\tau_{\min})/\rho)$. Taken to the extreme, starting $2^n n^{2n}/p_c$ ants at each vertex will allow λ-MMAS to discover all shortest paths in a constant number of iterations.

While the constant expected number of iterations requires an exponential increase in the amount of parallel computation, making such an approach impractical, picking a constant value of k only requires a polynomial increase in the amount of parallel computation as the graph size increases, which may be more practical. This conclusion is similar to that reached in [5] for the $(1 + \lambda)$ EA: a choice of λ that is roughly reciprocal to the probability of improvement in a single iteration usually provides a reasonable tradeoff between the increased number of fitness function evaluations in a single iteration and the decrease in the total expected number of iterations.

3.2 Iteration-best reinforcement

The λ-MMAS$_{\text{ib}}$ algorithm, adapted to the shortest path problem from the version analyzed on OneMax[3] in [11], is shown as Algorithm 3 below. Similar to λ-MMAS, it starts λ ants at each vertex, but unlike the algorithms considered previously, it only keeps track of he best-so-far path x_v^* within a given iteration, relying on the implicit memory in pheromone values to ensure that the best-so-far paths are likely to be reproduced in the next iteration, making it more similar to a $(1, \lambda)$ EA[4].

[11] shows that, with a sufficiently low evaporation rate and a surprisingly small number of ants, OneMax can be solved by an ant colony in expected $O(n \log n)$ iterations. The approach used demonstrates that there is a positive pheromone drift to the correct arcs in the construction graph. Unfortunately, this does not directly apply to single destination shortest path problems, which are more akin to LeadingOnes[5], as there's only guaranteed to be one shortest path at a time that is easily discoverable by an ant. Nevertheless, the number of ants required for iteration-best reinforcement to succeed may be surprisingly low.

Running the algorithm with a high evaporation rate, $\rho = 1$, simplifies the analysis of λ-MMAS$_{\text{ib}}$, as pheromone values are always frozen at the pheromone

[3] OneMax is a fitness function that maps n-bit strings to the number of 1 bits they contain, and is frequently used as an example function while analyzing performance of evolutionary algorithms. ACO can be used on OneMax in conjunction with a construction graph (the paths through which are mapped to bit strings), see e.g. [13].

[4] The behavior of the $(1, \lambda)$ EA is further examined in [17], which demonstrates that there is a sharp threshold at which the expected optimisation time for the $(1, \lambda)$ EA on a simple fitness function transitions from polynomial running time (similar to the $(1 + \lambda)$ EA) – to exponential running time. This provides an intuition that additional ants might be *required* for λ-MMAS$_{\text{ib}}$ to be effective.

[5] Another common pseudo-boolean fitness function, mapping n-bit strings to the number 1-bits before the first 0-bit. Optimizing it is more difficult than OneMax, as only one bit (namely, the first 0-bit) can be changed to improve the fitness value.

Algorithm 3 The λ-MMAS$_{\text{ib}}$ algorithm on a directed graph $G = (V, \mathcal{A})$, with pheromone bounds $\tau_{\min}$ and $\tau_{\max}$, and evaporation rate ρ.

Initialize $\tau_a \leftarrow 1/\deg^+(\text{tail}(a))$ for all $a \in \mathcal{A}$
for $i \leftarrow 1, 2, \ldots$ **do**
  for each $v \in V$ **do**
    for $j = 1, 2, \ldots, \lambda$ **do**
      Let $x_{v,j}$ be a new path starting at v
      $p \leftarrow v$, $S \leftarrow \{a \in \mathcal{A} \mid \text{tail}(a) = p \wedge \text{head}(a) \notin x_{v,j}\}$
      while S is not empty and $p \neq t$ **do**
        Select an arc a from S, with probability:
          $p_a = \tau_a / \sum_{s \in S} \tau_s$
        Append a to $x_{v,j}$
        $p \leftarrow \text{head}(a)$, $S \leftarrow \{a' \in \mathcal{A} \mid \text{tail}(a') = p \wedge \text{head}(a') \notin x_{v,j}\}$
    $x^*_v \leftarrow \arg\min_{x_{v,j}} f(x_{v,j})$
  for each $a \in \mathcal{A}$ **do**
    $$\tau_a \leftarrow \begin{cases} \min(\tau_{\max}, (1-\rho)\tau_a + \rho) & \text{if } a \in x^*_{\text{tail}(a)} \\ \max(\tau_{\min}, (1-\rho)\tau_a) & \text{otherwise} \end{cases}$$

bounds, with $\tau_{\max}$ pheromone value assigned to the first arc of each of the previous iteration's best paths x^*_v. This removes the need to consider freezing phases of the optimisation process, leaving just two probabilities to consider: the probability of an ant discovering a new shortest path (with exactly one arc with pheromone value $\tau_{\min}$), and the probability of the previous iteration's x^*_v path being *forgotten* – not being constructed by any ant started at v in the next iteration. Forgetting a path with $\rho = 1$ can prove disastrous for the optimisation process, as any longer paths that contain the forgotten path as a subpath might with high probability not be constructed in the next iteration (depending on the choice of λ). The following theorems approach the problem by attempting to make forgetting *any* shortest paths during the entire process unlikely.

Theorem 6 *Starting $\lambda = \Omega(\log n)$ ants at each vertex allows λ-MMAS$_{ib}$ with a high evaporation rate ($\rho = 1$) to find all shortest paths in $O(n^3)$ iterations with high probability.*

Proof λ-MMAS$_{\text{ib}}$'s discovery phases are identical to those of λ-MMAS. In the worst case, with $\lambda = 1$ and $\tau_{\min} = n^{-2}$, the probability of new shortest path being discovered in one iteration is at least $n^{-2}/(2e)$, so each discovery phase lasts an expected $2e \cdot n^2$ iterations. Assuming progress made during the discovery phases is never undone by the iteration-best reinforcement, the $n - 1$

discovery phases finish in expected $O(n^3)$ iterations. Chernoff bounds can be used to show that this result holds with overwhelming probability.

Let I_i be the indicator event of λ-MMAS discovering a new shortest path in iteration i (i.e. $I_i = 1$ if a new shortest path is discovered in iteration i, and 0 otherwise). The probability of a new path being discovered is always at least $p_i = n^{-2}/(2e)$ while the pheromones are frozen and there are undiscovered shortest paths remaining, so the I_i events can be considered independent for the purpose of this proof[6]. Then $N_k = \sum_{i=1}^{k} I_i$ is the number of discoveries in k iterations; setting $k = 4e \cdot n^3$ yields $E(n_k) = 2n$, and per Chernoff bounds:

$$P(N_k < (1-\delta)E(N_k)) < e^{-E(N_k)\delta^2/3}$$
$$P(N_k < n) < e^{-n/6} = e^{-\Omega(n)}$$

so at least n discoveries occur in $4en^3 = O(n^3)$ iterations with overwhelming probability for any choice of λ, as long as no shortest paths are forgotten.

The second element to consider is the probability of forgetting a discovered shortest path. Any shortest path we are concerned about forgetting contains at most ℓ arcs with pheromone value $\tau_{\max}$, and can therefore be constructed by a single ant with probability at least e^{-1} per the usual choice of pheromone bounds. Pessimistically, assume that the shortest path is forgotten if it is not reconstructed by any ant in an iteration[7]; this occurs with probability at most p_{f}:

$$p_{\mathrm{f}} \leq (1-e^{-1})^{\lambda}$$

There are at most n shortest paths that can be forgotten by λ-MMAS$_{\mathrm{ib}}$ in any single iteration, so the probability of failure in a single iteration is at most $n \cdot p_{\mathrm{f}}$, and the probability of failure in k iterations is at most $nk \cdot p_{\mathrm{f}}$ using union bounds. Let $k = c \cdot n^3$, where c is a constant such that k iterations complete the optimisation process with overwhelming probability ($c = 4e$ from above), and select a λ such that the probability of failure is $o(1)$:

$$\lambda = \frac{\log(n^{-5}/c)}{\log(1-e^{-1})} = O(\log n) \quad \Rightarrow$$
$$cn^3 \cdot n \cdot (1-e^{-1})^{\lambda} = cn^4 \cdot n^{-5}/c = 1/n = o(1)$$

[6] This may lead to situations where the indicator events suggest that more discoveries have occurred during the considered iterations than is actually possible. The extraneous discoveries may simply be ignored, and the combined outcome interpreted as the colony having discovered all shortest paths.

[7] In order to negatively affect the optimisation process, not only must the correct shortest path x_v^* not be constructed by any ant started at v, but the constructed path with the best fitness value must start with a different arc out of v – otherwise, the pheromones will not be altered.

Starting $\Omega(\log n)$ ants at each vertex ensures that with high probability no shortest path is forgotten in $4e \cdot n^3$ iterations, and given that no shortest path is forgotten during that number of iterations, all shortest paths are found with overwhelming probability. Therefore, choosing $\lambda = \Omega(\log n)$ ensures that all shortest paths are found in at most $O(n^3)$ iterations with probability approaching 1. □

Imposing the requirement that no paths are forgotten during the entire optimisation process may seem overly pessimistic. Intuitively, λ-MMAS$_{\text{ib}}$ might able to find all shortest paths in polynomial time if forgetting occurs infrequently enough for the colony to be able to rediscover the paths it forgets before forgetting more. Suppose for a moment that this intuition is correct, and is accurately reflected by the requirement in (4)[8]: the probability of forgetting a path is must be lower than the probability of discovering a new shortest path.

Bernoulli's inequality, $(1+x)^r \geq 1 + x \cdot r$ can be used to upper-bound the right side of the requirement inequality (5). Rearranging the equation yields (7), where c is a positive constant:

$$(1 - e^{-1})^\lambda < 1 - (1 - 1/(2n^2))^\lambda \tag{4}$$

$$(1 - e^{-1})^\lambda < \lambda/(2n^2) \tag{5}$$

$$\log \lambda - \lambda \log(1 - e^{-1}) > \log 2 + 2 \log n \tag{6}$$

$$c \cdot \lambda + \log \lambda > \log 2 + 2 \log n \tag{7}$$

Picking $\lambda = \Omega(\log n)$ would satisfy this optimistic requirement. Asymptotically, this is the same lower bound on the number of ants as proved by Theorem 6, so the requirement that no shortest paths are forgotten is reasonable.

3.2.1 Constant evaporation rate

Per Theorem 6, $\Omega(\log n)$ ants are sufficient if the evaporation rate is 1, in which the freezing phases do not exist. When the evaporation rate ρ is a constant, it is possible for the ant colony to fail to reconstruct the newly discovered shortest paths during their freezing phases, where the probability of reconstructing them is lower than the probability of reconstructing an already frozen path. This

[8] This formulation is too optimistic with respect to making progress – it reflects a model where the number of arcs in the longest shortest path known to the colony may be altered by at most 1 in either direction through forgetting/discovery, and optimisation completes if this number ever reaches ℓ. This neglects to consider that sub-paths of the longest known shortest paths may also be forgotten, which can undo large amounts of progress.

section will show that this failure probability is adequately controlled by starting $\Omega(\log n)$ ants at each vertex, as stated by the following theorem.

Theorem 7 *Starting $\lambda = \Omega(\log n)$ ants at each vertex allows λ-MMAS$_{ib}$ with a constant evaporation rate $\rho > 0$ to find all shortest paths in $O(n^3)$ iterations with high probability.*

Proof If the ant colony keeps reinforcing the same arc, a freezing phase is completed in no more than $\log(\tau_{\max}/\tau_{\min})/\rho$ (or $2\log(n)/\rho$ for the usual choice of pheromone bounds) iterations per Lemma 2. In λ-MMAS$_{\text{ib}}$, it is possible that the discovered shortest path will *not* be constructed by any ant during an iteration, and hence will not be reinforced. The pheromone value on the relevant arc during an iteration phase is always at least ρ (following the initial reinforcement after the discovery iteration), so the probability of selecting that arc, and then following the reinforced shortest path to t is always at least $\rho/(2e)$.

A sufficient (but not necessary) requirement to complete a freezing phase successfully is to *always* reinforce the relevant arc in $2\log(n)/\rho$ sequential iterations. Consider the probability p_{f} of any ant failing to construct shortest path being frozen in a single iteration; if $\lambda = c_1 \log n$, this probability is small. As ρ is a constant, c_1 can be chosen based on ρ (and remain independent of n) to make this probability at most n^{-2}, as shown in (8):

$$p_{\text{f}} \leq (1-\rho/(2e))^{\lambda} = \left(\frac{2e-\rho}{2e}\right)^{c_1 \log n} = n^{-c_1(\log(2e)-\log(2e-\rho))}$$

$$c_1 = \frac{2}{\log(2e)-\log(2e-\rho)} \Rightarrow p_{\text{f}} \leq n^{-2} \tag{8}$$

Assuming that no shortest paths are forgotten at any stage of the process, λ-MMAS$_{\text{ib}}$ needs to perform at most n freezing phases of $2\log(n)/\rho$ iterations each. With probability approaching 1, no shortest path is forgotten during the freezing phases, as shown by applying a union bound to the probability p_{f} derived above:

$$(1-p_{\text{f}})^{(n\cdot 2\log(n)/\rho)} \leq 1 - p_{\text{f}} \cdot n \cdot 2\log(n)/\rho \leq 1 - \frac{n\log(n)}{\rho n^2} = 1 - o(1)$$

Thus, starting $\Omega(\log n)$ at each vertex ants is sufficient to ensure a high probability of completing all freezing phases successfully when ρ is constant.

This can then be combined with the proof of Theorem 6. The number of iterations required to *discover* all shortest paths with overwhelming probability is

unchanged, though now the discovery events are followed by the freezing phases before discovery is resumed. The bound on the probability of not forgetting any known (frozen) paths can easily be extended to cover any polynomial number iterations while preserving $\Omega(\log n)$ ant requirement, though this is not needed, as for constant ρ, the number of freezing phase iterations is subsumed by the number of discovery phase iterations allotted by the Theorem. Therefore, all shortest paths are discovered in $O(n^3)$ iterations when $\rho > 0$ is constant and $\lambda = \Omega(\log n)$ ants are started at each vertex with probability approaching 1 as n increases. □

3.2.2 Low evaporation rates

Starting $\Omega(\log n)$ ants at each vertex is sufficient for λ-MMAS$_{\text{ib}}$ to have a high probability of successfully finding all shortest paths within $O(n^3)$ iterations when the evaporation rate is at least constant. If the evaporation rate depends on n, the freezing phases become more difficult to analyze, as there is no longer a positive constant lower bound on the probability of a single ant reconstructing the path.

If the failure to reconstruct a path cannot be prevented entirely, the effects of pheromone evaporation would have to be considered more closely. Notably, the relative effect of pheromone evaporation increases with the increase in pheromone value: while pheromone values are low, it would take many unsuccessful iterations to undo the effects of a single successful iteration, but as the pheromone value increases, this tolerance for failure is reduced. Balancing these effects is difficult.

A simple alternative, albeit a potentially inefficient one, is to start enough ants at each vertex to ensure that every iteration a sufficiently high probability of constructing the "next" shortest path if all shortest paths with fewer arcs are already known.

Let p_1 be the probability that a single ant constructs a shortest path by selecting a single non-reinforced arc and then following reinforced arcs to the destination. Following the approach of Theorem 7, the pheromone value on the first arc of a newly-discovered shortest path after the initial reinforcement is at least $\max(\rho, \tau_{\min})$: for low evaporation rates ρ (e.g. $\rho < n^{-2}$), the bound imposed by $\tau_{\min}$ is better.

p_{c} is then the probability that one of λ ants started at a vertex will construct

the shortest path in this manner:

$$p_1 \geq 1/(2e \cdot \max(\rho, \tau_{\min}))$$
$$p_c \geq 1 - (1 - 1/(2e \cdot \max(\rho, \tau_{\min})))^{\lambda}$$

Picking $\lambda = \Omega(\max(\rho, \tau_{\min})^{-1} \log n)$, or $\lambda = \Omega(n^2 \log n)$ (which works for any choice of ρ) or $\lambda = \Omega(\rho^{-1} \log n)$ (which is a tighter bound for $\rho > \tau_{\min}$) makes p_c approach 1 as the problem size increases, giving each iteration a high probability of constructing the relevant shortest path. Intuitively, this should allow the optimisation process to finish in expected polynomial time with respect to n and $1/\rho$.

4 Periodic changes

The previous sections established upper and lower bounds on the number of iterations needed by various ACO algorithms to find all shortest paths to a particular vertex following a one-time change in the weight function of the graph. This section examines the conditions under which λ-MMAS may be able to keep up with regular changes to the weight function.

If the changes are sufficiently infrequent, i.e. occurring less often than the bounds derived for rediscovering shortest paths after a one-time change as found in Section 2, they are not particularly interesting – λ-MMAS will be able to rediscover the shortest paths within a polynomial number of iterations, which will then remain correct until the weight function changed again. Thus, this section is concerned with periodic changes that are more frequent than that.

When dealing with periodic changes, it is the implicit memory of the previous shortest paths stored in pheromone values that may allow ACO algorithms to adapt to some forms of period changes in the weight function, and find the new shortest paths relatively quickly after each change is made. For instance, [16] demonstrates that an ACO algorithm is able to follow a particular series of periodic changes to the fitness function (on a pseudo-boolean optimisation problem) while an EA is not, essentially relying on a period of fast oscillation between two variants of the fitness function, where the "new" variant is used more often than the one being switched away from, to guide the ACO pheromone values to favor the "new" variant before it is switched to completely.

Notably, oscillation between different fitness functions can be captured by the pheromone values if the evaporation rate is low enough to allow non-frozen pheromone values to persist during the oscillation. In [16], this ensures that even if a solution is constructed for the fitness function unfavored during the oscillation, the results of the pheromone reinforcement will not be able to undo the effects of a large number of successful previous iterations.

The following subsections examine how a low evaporation rate can be useful to ensure that λ-MMAS is able to consistently find shortest paths while the weight function is switched after every iteration; how setting the evaporation rate too low may hinder λ-MMAS in following a slower series of permanent changes; and demonstrate that ACO is not able to efficiently track fitness functions that switch between some weight functions regardless of the choice of evaporation rate.

4.1 Tracking a fast oscillation

The graph shown in Figure 3 can be used to illustrate the effects of selecting the evaporation rate ρ, using the two weight functions shown in (9). Under w_1, the shortest path from s to t does not visit b; under w_2, it does.

$$w_1(a) = \begin{cases} 1 & \text{if } a = (b,c) \\ 0 & \text{otherwise} \end{cases} \qquad w_2(a) = \begin{cases} -1 & \text{if } a = (b,c) \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Consider λ-MMAS, $\lambda = \log^2 n$, running on the graph in Figure 3, with the weight function alternating between w_1 and w_2 every iteration.

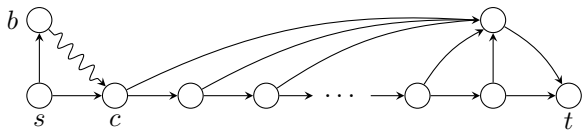


Figure 3: The weight of the wavy (b, c) arc can be adjusted to control whether b will be visited on the shortest path from s to t.

Using these weight functions, the subgraph that follows from c to t serves only to present the ants started at s with $\ell = \Omega(n)$ choices, justifying a non-constant $\tau_{\min}$ (and thus also $p_{\min}$). Whether the ant started at s manages to find a shortest path to t depends only on whether it selects the correct arc out of s, as any path from c to t has weight 0 using either weight function. Notably, because both arcs out of s have equivalent weight, heuristics[9] based on arc weight may not be effective in this situation. As λ-MMAS reevaluates the fitness value of the shortest path for each comparison, it is possible to switch between these two weight functions without concern that the colony would not accept the proper w_1 shortest path after finding the w_2 shortest path, which has a lower weight.

If the evaporation rate is large, the pheromone values will be eventually frozen by λ-MMAS; for $\rho = 1$, the pheromones will be frozen immediately after the first iteration. Once the pheromone values are frozen, and the weight function is changed such that they no longer reflect the true shortest path, one of the $\log^2 n$ ants starting at s will need to make a single pheromone-defying arc selection in order to find the new shortest path. A single single ant makes this selection with probability $p_{\min} = \tau_{\min} \geq 1/(2n)$ (using $\Delta = 2$ and

[9] ACO algorithms may be adapted to use both the pheromone information and external heuristic information to influence arc selection during the construction of random paths through the graph. For more details, see e.g. [4].

pessimistically $\ell \leq n$). Applying a union bound, the probability of any of the $\log^2 n$ ants starting at s discovering the new shortest path in an iteration where one exists is at most:

$$\frac{\log^2 n}{2n} = O(n^{-1} \log^2 n) = o(1)$$

Therefore, with probability approaching 1, λ-MMAS will not discover the correct arc out of s when the weight function changes, and will therefore be wrong approximately half the time if the pheromones freeze.

The following theorem shows that if the evaporation rate is not overly high, pheromone values can be prevented from freezing for any polynomial number of iterations – and therefore each iteration maintains a high probability of constructing the correct shortest path from s.

Theorem 8 *Starting $\lambda = \Omega(\log^2 n)$ ants at s is sufficient is sufficient to ensure that the pheromone values are not frozen by λ-MMAS within a polynomial number of iterations when $\rho = 1/n$.*

Proof If $\rho = 1/n$, the pheromone values will not be frozen immediately. Assuming that the pheromone values are within the range $[1/10, 9/10]$, the $\log^2 n$ ants starting at s will be able to discover the shortest path from s with at least the following probability, even if the pheromones currently favor the longer path:

$$1 - (1 - 1/10)^{\log^2 n} = 1 - n^{-\log(10/9)\log(n)} = 1 - o(1) \tag{10}$$

So with probability approaching 1, as long as the pheromones are within the above range, λ-MMAS will be able to discover the new shortest path and therefore adjust pheromone values towards 1/2.

How long does λ-MMAS manage to keep the pheromone values within this range? Without loss of generality, consider a situation when, after the pheromones were updated using the w_1 weight function, the pheromone value τ_0 on the (s, c) arc satisfies $(1/10)/(1 - \rho) \leq \tau_0 < 1/2$. Pessimistically, the next iteration will decrease the pheromone value further, but the iteration after that, if successful, will update the pheromone value to τ_2:

$$\tau_2 = (1 - \rho)^2 \tau_0 + \rho = \tau_0 + \rho(1 - 2\tau_0) + \rho^2 \tau_0 > \tau_0$$

Thus as long as the next iteration is successful, the pheromone values are adjusted in the right direction and the process can continue. If $\log^2 n$ ants are

started at each vertex, the pheromone values will stay within the $[1/10, 9/10]$ range with high probability for any polynomial number of iterations n^k for a sufficiently high n. For instance, for n such that $\log(10/9)\log(n) \geq k+1$, the probability of all considered pairs of iterations in n^k iterations adjusting the pheromone values towards $1/2$ approaches 1:

$$\begin{aligned}(1 - n^{-\log(10/9)\log(n)})^{n^k} &\geq 1 - n^k \cdot n^{-\log(10/9)\log(n)} \\ &= 1 - n^{k-(k+1)} = 1 - o(1)\end{aligned}$$

Therefore, starting $\log^2 n$ ants is enough, for sufficiently high n, to keep the pheromone values on outgoing arcs from s from being frozen for any polynomial number of iterations. □

This in turn allows the shortest paths from s to be constructed with high probability in each iteration per equation (10).

4.2 Low evaporation rates

The previous section demonstrated an example where using low evaporation rates enables λ-MMAS to keep the pheromone values from being frozen, and therefore reliably able to find the shortest path despite the weight function oscillation. Setting an evaporation rate to a low value is not always beneficial, however.

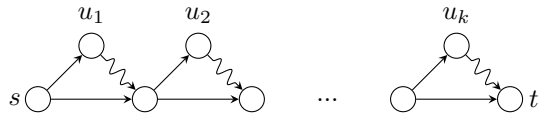


Figure 4: The weights of the wavy arcs from u_i vertices can be adjusted to control whether the u_i vertex is visited on the shortest path from s to t.

Consider a graph of k triangles on the path from s to t (in total, $n = 2k+1$ vertices), as shown in Figure 4, and the family of weight functions w_i for $0 \leq i \leq k$, such that the shortest path from s to t under w_i includes the vertices $u_1, \ldots, u_i$, but not $u_{i+1}, \ldots, u_k$:

$$w_i(a) = \begin{cases} -1 & \text{if tail}(a) = u_j \text{ and } j \leq i \\ 1 & \text{if tail}(a) = u_j \text{ and } j > i \\ 0 & \text{otherwise} \end{cases}$$

Choose $\tau_{\min} = 1/(2n)$, reflecting the maximum vertex degree and a pessimistic assumption about maximum shortest path length. On this graph, with a sufficiently high n, λ-MMAS with $\lambda = 1$ ants finds all shortest paths in $4n^2 + \log(\tau_{\max}/\tau_{\min})/\rho$ iterations with overwhelming probability (this can be shown using Chernoff bounds on the number of discoveries of shortest paths, similar to the approach used in proving Theorem 6).

Suppose that λ-MMAS is allowed to run with the w_0 weight function for $t_0 = 4n^2 + \log(2n)/\rho$ iterations. This is enough to allow it to discover and freeze all shortest paths with overwhelming probability. Then, the next weight functions are used in ascending order for $t = 4n^2 + n\log(2n)$ iterations each – adding the vertices $u_1, \ldots, u_k$ to the shortest path each time. If $\rho \geq 1/n$, λ-MMAS is able to discover and freeze the new shortest paths with overwhelming probability (regardless of the choice of λ); this process is easily repeated $k < n$ times while maintaining overwhelming probability of having successfully frozen the pheromone values of the shortest paths at the end.

If ρ is too low, e.g. $\rho = 1/n^4$, λ-MMAS will fail to find the correct shortest paths at the end of the t iterations after switching to w_k with overwhelming probability, as long as λ is at most polynomial with respect to n.

Proof Recall that the initial t_0 iterations are sufficient to freeze the correct shortest paths for w_0 with overwhelming probability, so when the weight function is first switched to w_i, the pheromone value on the arc leading to u_i is $\tau_{\min}$. Consider the last $k/2$ triangles: the pheromone values on the arcs leading to their u vertices is at most the pheromone value on the arc to $u_{k/2}$.

Optimistically (with respect to the optimisation progress) assume that the correct shortest path including u_i is discovered immediately upon switching to w_i. Thus, after switching to $w_{k/2}$, at most $(k/2) \cdot t$ iterations elapse before the t iterations with w_n are over. The pheromone value on the $u_{k/2}$ arc at the end of this process is not more than:

$$\begin{aligned}\tau_{\min} + \rho \cdot (k/2) \cdot t &< 1/(2n) + n^{-4} \cdot (n/4) \cdot (4n^2 + n\log(2n)) \\ &= \frac{1}{2n} + \frac{1}{n} + \frac{\log(2n)}{4n^2} = o(1)\end{aligned}$$

As correct shortest path from s to t includes $k/2 \approx n/4$ arcs with at most this pheromone value, the probability of constructing it within the t operations after switching to w_k is overwhelmingly small, even if a polynomial number of ants are started at s. □

This example illustrated that a low evaporation rate may negatively impact

the ability of ACO-based algorithms to track permanent changes in the fitness function.

4.3 Impact of oscillating component size

The previous sections have examined periodic changes that only have a minor impact on the shortest paths in the graph – more specifically, only the value of a single "choice" (of whether to visit b and u_i) was altered at a time. It has been shown that if the evaporation rate is chosen appropriately, λ-MMAS is able to track these repeated changes reliably, by either keeping the pheromones close to 0.5 if the oscillation between weight functions is fast, or by correctly adapting to the new shortest paths if the change is permanent. Thus, if the weight functions produce similar shortest paths (as characterized by a low, constant Hamming distance), the implicit memory in pheromones is useful.

Let us consider a situation where the weight functions the fitness function oscillates between do *not* produce similar shortest paths. For instance, consider the graph in Figure 5, which has k columns of 2 vertices, and an additional destination vertex t. For this graph, there exist pairs of weight functions which specify shortest paths with no arcs in common, resulting in a large Hamming distance between shortest paths that also increases with graph size. When the fitness function oscillates between such a pair of functions, it is difficult for λ-MMAS to track the shortest paths correctly.

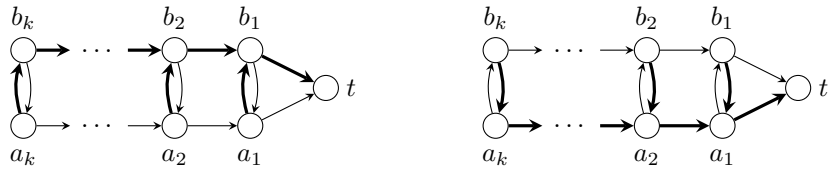


Figure 5: Shortest paths specified by the weight functions in equation (11) are shown in thick arcs. Oscillating between weight functions that specify these shortest paths may make it difficult for ACO algorithms to reliably find the shortest paths.

The following two weight functions can be used to ensure that the shortest paths from any vertex do not share any arcs; here, $C_i = \{(a_i, b_i), (b_i, a_i)\}$ is the

set of arcs within column i:

$$w_1(a) = \begin{cases} 2 & \text{if } a = (a_1, t) \\ \frac{2k-i}{k} & \text{if } a \in C_i \\ 0 & \text{otherwise} \end{cases} \qquad w_2(a) = \begin{cases} 2 & \text{if } a = (b_1, t) \\ \frac{2k-i}{k} & \text{if } a \in C_i \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

Under either weight function, there is a unique shortest path to t from any vertex in the graph: it either follows the non-C_i arcs all the way to t, or takes the C_i arc from the starting vertex and then follows the non-C_i arcs all the way to t. The shortest paths under w_1 and w_2 are shown in Figure 5 on the left and right graph respectively.

Section 4.1 demonstrated that λ-MMAS is able to handle a fast oscillation between two weight functions by keeping the pheromone values close to 0.5, preserving its ability to explore both options being oscillated between with high probability if $\lambda = \log^2 n$ ants are started at each vertex. Even if λ-MMAS is able to keep pheromones on all arcs of Figure 5 close to 0.5, it will fail to construct the shortest path from a_k with overwhelming probability:

$$(1 - 2^{-k})^{\log^2 n} \geq 1 - \frac{\log^2 n}{2^{(n-1)/2}} > 1 - 2^{2-(n-1)/2} = 1 - 2^{-\Omega(n)}$$

On the other hand, if any pheromone values are frozen, then, with high probability, none of the $\log^2 n$ ants started at a vertex will construct a path containing the arc with pheromone value $\tau_{\min}$. Thus, $\lambda = \Omega(\log^2 n)$ ants will not discover the shortest paths at least half the time if the oscillation is fast.

If the oscillation is slower, the pheromone values vertices close to t can freeze to favor the correct shortest path arcs. When the pheromone values are frozen and the weight function is changed, the situation is similar to that considered in Theorem 4: due to the choice of weight functions, only one column at a time is able to construct correct shortest paths with exactly one non-reinforced arc. For an example, consider pheromone values being frozen per w_1, and the weight function switched to w_2: the (b_{20}, a_{20}) arc can only be reinforced after the full shortest path from b_{20} is constructed, as the weight of any two C_i arcs is greater than the penalty on the (b_{20}, t) arc, which is the weight of the w_1's shortest path from b_{20} according to w_2, so the pheromones on the (a_{19}, b_{19}), ..., (a_1, b_1) arcs will need to be reduced to make it easier to construct the shortest path from b_{20}.

If the weight function changes less often than the time it takes to rediscover all of the shortest paths per Theorem 4 (i.e. less often than every $\Omega(\ell \cdot \tau_{\min}^{-1}/\lambda)$ iterations), the shortest paths may be correct some fraction of the time; otherwise, there will, intuitively, almost always be a vertex on which the pheromones favor the wrong arc.

Thus, unless the oscillation is sufficiently slow for λ-MMAS to rediscover all shortest paths before the fitness function changes again, λ-MMAS is not able to keep track of the shortest paths from a_k and b_k reliably, even if using $\lambda = \log^2 n$ ants as in the previous sections. The exact behavior of alternating these weight functions on this graph is examined experimentally in Section 5.2.3.

5 Implementation and Experiments

In order to examine the effectiveness of algorithms based on the ant colony optimisation metaheuristic from a practical viewpoint in addition to the theoretical analyses presented in the previous sections, λ-MMAS and λ-MMAS$_{\text{ib}}$ algorithms have been implemented in a multithreaded C program. While a variety of implementations of algorithms based on the ACO metaheuristic are available on the Ant Colony Optimisation Public Software list[10] for solving various combinatorial optimisation problems, none are targeted at shortest path problems, so a new implementation was necessary to allow experimental evaluation of the algorithms' behavior.

This section describes overall design of the implementation, and presents experimental results that provide additional detail concerning the behaviors predicted by theoretical analyses.

5.1 Implementation overview

The algorithms were implemented in C (C99), using the POSIX threads library (pthreads) for multithreading primitives, the Mersenne Twist pseudorandom number generator[11] for random number generation, and Lua[12] to allow customization of optimisation parameters, graph updates, and output logic.

The initial weight function specifying the graph is read from a user-specified text file, which encodes information about the graph as a series of whitespace-separated numbers. The text file consists of the number of vertices in the graph n, followed by the out-degrees of vertices 1 through $n-1$ (vertex 0 is by convention the destination vertex and has no outgoing arcs), and finally the pairs of head vertex indices and arc weights specifying the arcs in the graph, sorted such that the arc (a, b) appears before (c, d) if $a < c$ or $a = c \wedge b < d$. Multiple arcs and self-loops are disallowed.

A user-specified number of threads is then used to perform the path construction and pheromone updates. To minimize the number of synchronization calls required to ensure that work is performed correctly, all λ ants started from a vertex are simulated by the same thread, and vertices are allocated to threads

[10] `http://www.aco-metaheuristic.org/aco-code/public-software.html`

[11] C/C++ implementation by Geoff Kuenning, `http://www.cs.hmc.edu/~geoff/mtwist.html`

[12] Lua is a powerful, fast, lightweight, embeddable scripting language; `http://www.lua.org/about.html`

in chunks – if a thread is available to do work, will get assigned a chunk of

$$\left\lceil \frac{\text{number of remaining vertices to process}}{\text{total number of threads used} + 1} \right\rceil$$

vertices to compute/reinforce the shortest paths for. This work allocation scheme reduces the size of the allocated chunks as the path construction / reinforcement phase nears completion, in an attempt to minimize the chances that a large number of threads will be waiting for a single thread to finish work on a large assigned chunk. Notably, there is a tradeoff between finer allocation granularity (which may distribute the work more evenly across threads, resulting in less idle time towards the end of the phase) and the number of mutex lock/unlock calls required to allocate vertices to unique threads. The selected strategy performs best for graphs with a relatively large number of vertices n, and a relatively small number of ants λ.

A synchronization barrier is used to ensure that the implementation waits for the construction of all shortest paths to finish before beginning pheromone updates, and vice versa. While the POSIX threads specification includes barriers as an optional component, they are not available on all platforms, so barrier synchronization is implemented using the pthreads mutex and conditional variable primitives that are more widely supported.

Performing path construction requires access to random numbers in order to determine which outgoing arc is selected. The random number generators included in C's standard library are problematic for two reasons: the default granularity of `rand` is relatively low (the standard only guarantees generation of a random integer between 0 and 32767), and the generators often use global state, so multiple threads using the built-in PRNGs will either not get independent random numbers, or will have to contend for access to the PRNG state variables, reducing performance. The Mersenne Twist random number generator solves these issues, providing access to high-granularity pseudorandom numbers and allowing each thread to have a separate PRNG state.

Finally, in order to allow the algorithm parameters to be customized, and to allow the weight functions to be updated dynamically, the implementation runs user-specified scripts written in the Lua language. The scripts can control the number of threads started for the simulation, as well as alter the weight function, pheromone bounds and evaporation rate, pheromone values, and reinforcement mode (best-so-far or iteration-best) while the algorithm is running. This can be used to output the desired information about the current best-so-far path weights or pheromone values, depending on the situation being examined.

Further detail regarding the implementation is available in Appendix A (page 47).

5.2 Experiments

This section presents the results of experiments performed using the implementation. We first verify that the implementation produces expected results in a situation that has been thoroughly analyzed[13]: considering the expected number of iterations to recover from a one-time change, as analyzed in Section 2.

Then, the impact of additional ants on ACO's ability to track a fast oscillation in a fitness function, as discussed in Section 4.1 is examined experimentally. Finally, the implementation is used to demonstrate the behavior of λ-MMAS when the difference between the weight functions being oscillated between is significant, as discussed in Section 4.3.

5.2.1 Recovering from a one-time change

As a basic test case for the implementation, the situation considered in Section 2 can also be simulated using the implementation. The simulation is run on the graph shown in Figure 2 (page 11), with the initial pheromone values set to frozen values so as to favor all arcs leading to t', while the weight function ensures that the shortest paths avoid t' if possible.

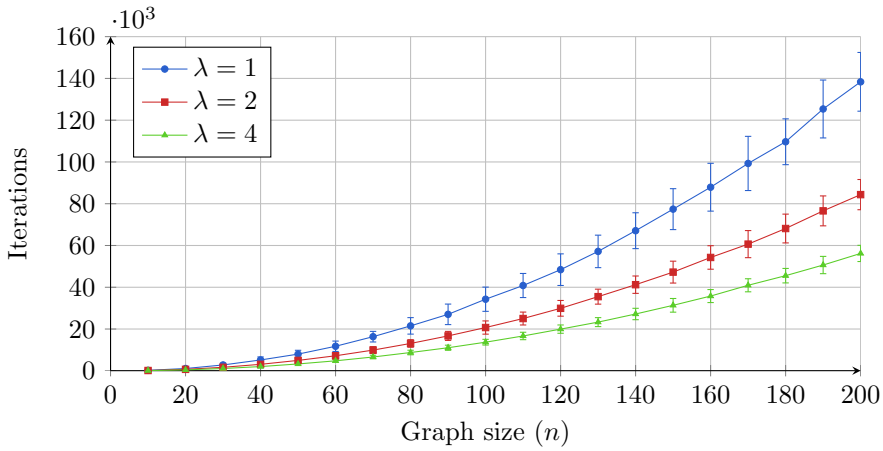


Figure 6: Average number of iterations before all shortest paths in a graph with n vertices are rediscovered by λ-MMAS; error bars indicate the 95% confidence interval based on sample variance.

[13] This is used as a test case for the implementation: if the results do not follow the predicted values, it is likely that the implementation contains an error of some type.

Figure 6 shows the average (over 100 simulations) number of iterations before all shortest paths are discovered by λ-MMAS, with $\rho = 1/n$ and $\tau_{\min} = 1/(n\Delta) = 1/(2n)$ for $\lambda = 1, 2, 4$. These results are consistent with those of Section 2: the increase in the number of iterations is approximately quadratic with relation to n (which is expected given the choice of $\tau_{\min}$), and doubling the number of ants does not quite halve the number of iterations required (also expected, as freezing phases are not shortened by increasing λ).

5.2.2 Tracking a fast oscillation

Consider the situation of Section 4.1: the weight function changes every iteration in such a fashion that the shortest path changes by a single choice (of whether to visit the vertex b in Figure 3, page 27).

The situation as described in that section can be simulated by considering only three vertices: s, b and c, using c as the destination, and altering the evaporation rate ρ and pheromone bounds to reflect an increase in the number of vertices in the original graph. Because all paths from c to t have weight 0, this simplification is equivalent to the original: if the shortest path from s to c is found, a shortest path from s to t is also found.

Figure 7 shows the average number of iterations (over at least 400 simulations) before the pheromone on the (s, b) arc exits the $[1/10, 9/10]$ range for various values of $1/\rho$ and $\lambda = 2, 3, 4$. Notably, while the $\lambda = 2$ graphs appear linear with respect to $1/\rho$, the $\lambda > 2$ graphs are definitely not, illustrating that even a few additional ants can make a significant difference for ACO algorithms.

5.2.3 Effects of oscillating component size

Section 4.3 considered a situation where the Hamming distance between the shortest paths of the weight functions oscillated between is large. The example illustrated that it is difficult for ACO to track repeated changes that alter shortest paths significantly, but was sufficiently complex to make it difficult to predict how exactly the ant colony would behave. In this section, we consider the behavior of λ-MMAS on the graph of Figure 5 (page 31) with $k = 50$ columns, $\lambda = 5$ ants started at each vertex, and evaporation rate $\rho = 1/n$. The results presented in this section are averages over 200 simulations of each set of parameters.

When the oscillation is fast – the weight functions are changed every iteration

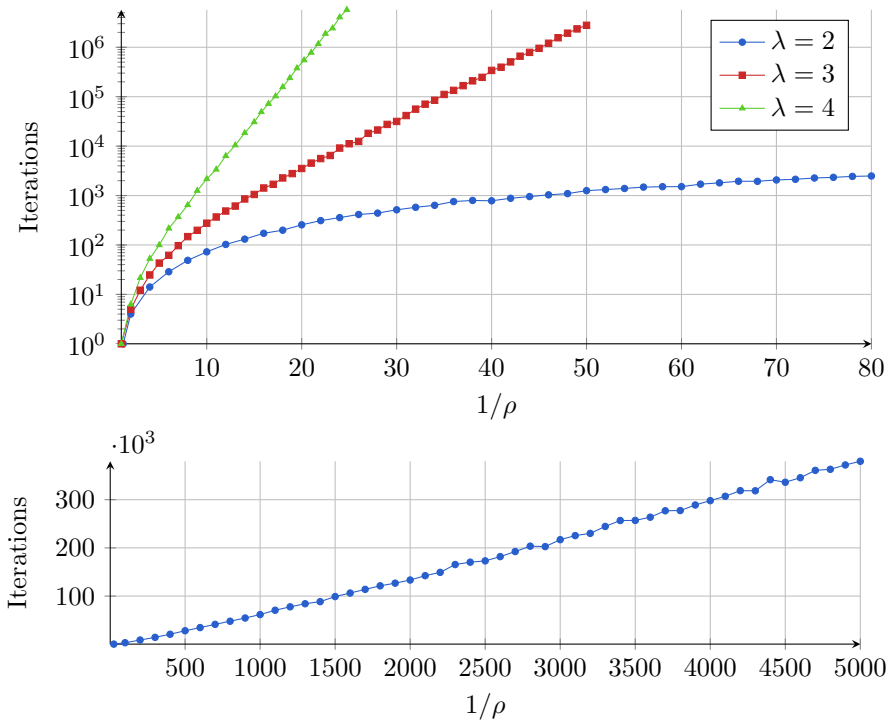


Figure 7: Average number of λ-MMAS iterations before the pheromone values on an arc that's only in the shortest path every other iteration exit the $[1/10, 9/10]$ range.

– λ-MMAS may be able to keep some of the pheromone values from being frozen, and thereby maintain a high probability that both arcs out of a given vertex will be explored by at least one ant. Figure 8 shows how the pheromone values on the (a_i, b_i) arcs develop in these circumstances.

While λ-MMAS is able to keep the pheromone values close to $1/2$ in the columns close to t (where the 5 ants can construct the new shortest paths with high probability), the pheromones do become frozen in columns further away from t. Recall that encountering any frozen pheromone value means that $\log^2 n$ ants started at each vertex will with high probability not explore the non-reinforced arc, and therefore will not construct the shortest paths in at least half the iterations. Thus, λ-MMAS is indeed unable to reliably construct the shortest paths from a_{50} and b_{50} in this case.

When the oscillation is slower – for instance, when the weight functions are

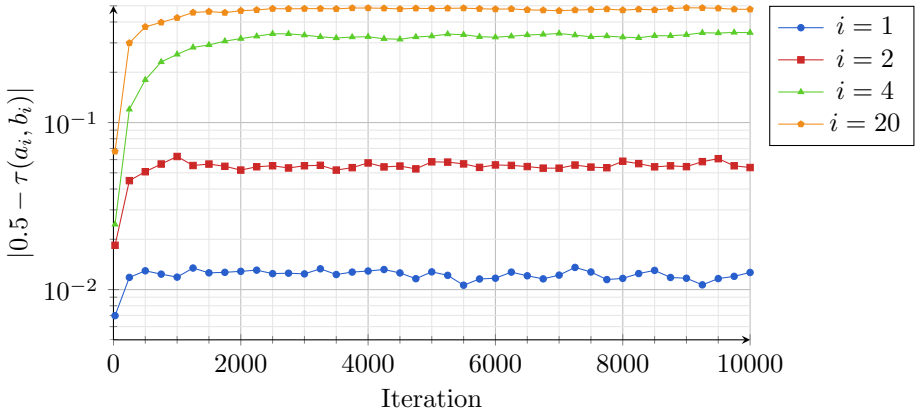


Figure 8: Average observed pheromone deviation from 0.5 on (a_i, b_i) arcs in various columns i, with the weight functions changing every iteration.

changed every 3030 iterations ($15 \times {\tau_{\min}}^{-1}$ iterations); the pheromone values on arcs close to t will be frozen to favor the correct shortest paths. Figure 9 illustrates how the pheromone values develop with this oscillation frequency.

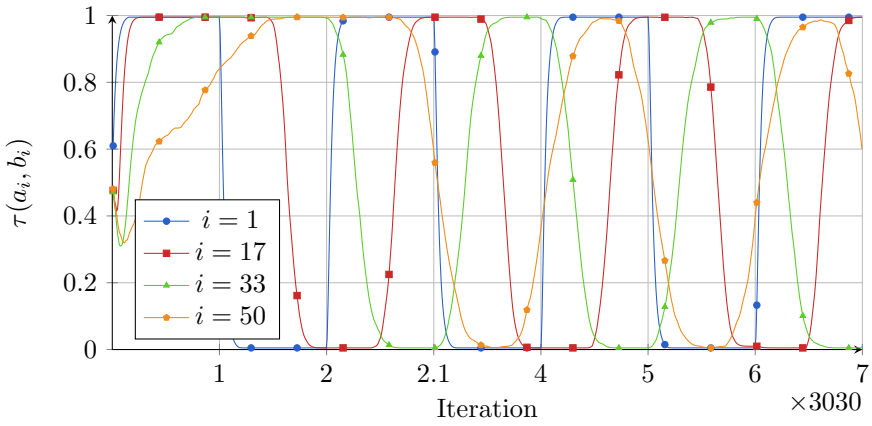


Figure 9: Average observed pheromone value on the (a_i, b_i) arcs when the weight function is changed every 3030 iterations.

Notably, while the pheromone values on the (a_1, b_1) arc are reliably frozen to favor the correct shortest path within relatively few iterations after the weight function is changed, pheromone values on arcs further away from t are slower to adapt – even to the point of changing to favor a particular path *after* the weight function changes. The behavior is perhaps best characterized as waves propagating through the graph – pheromones on arcs close to t are likely to

reflect the current shortest paths, while those on more distant arcs reflect older shortest paths.

Figure 10 shows the probability that the best-so-far path x_v^* is actually the correct shortest path from v in a given iteration, as measured by diving the number of simulations where that was the case by the total number of simulations performed. With this choice of parameters and oscillation frequency, λ-MMAS is able to reliably construct the shortest paths for approximately the first 20 columns before the weight function changes again, and does not appear to be able to construct the shortest paths for the last 15 columns at all before the weight function is changed again.

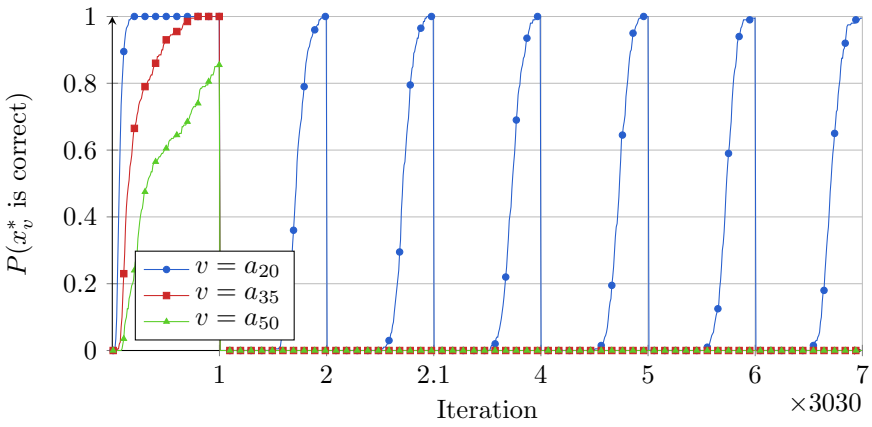


Figure 10: Probability that the best-so-far path x_v^* is the shortest path from v to t when the weight function is changed every 3030 iterations.

Figure 11 shows how many of the best-so-far paths x_v^* are correct shortest paths in a given iteration, as an average over 200 simulations. From it, it can be concluded that λ-MMAS will on average construct less than 50 correct shortest paths (i.e. from the 25 columns closest to t) before the weight function is changed.

From these experiments, it is clear that the original assertion that λ-MMAS with $\lambda = \log^2 n$ ants is unable to reliably construct the shortest paths from the a_k and b_k vertices of the graph is correct. The presented experimental data also revealed non-obvious details about the behavior of pheromones when the oscillation is relatively slow, which could serve as a starting point for future theoretical analysis of the conditions under which ACO algorithms may be able to efficiently handle oscillation between weight functions with significant changes to the shortest paths.

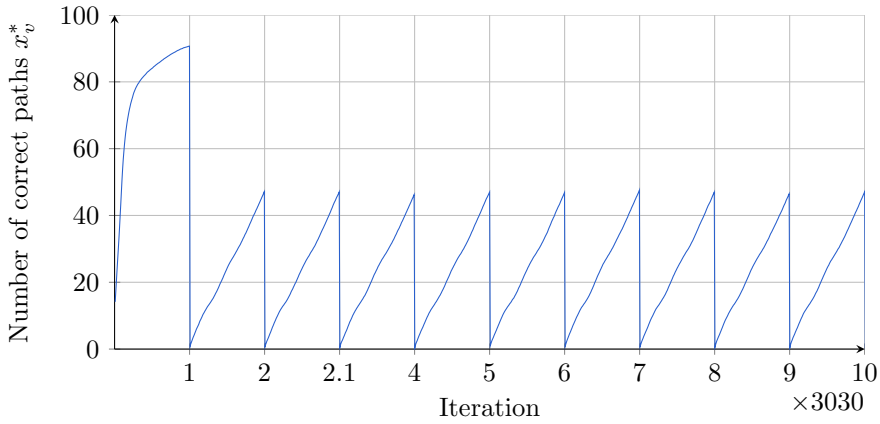


Figure 11: Average observed number of correct (shortest) paths x_v^* when the weight function is changed every 3030 iterations.

6 Discussion

This final section presents a summary of the topics discussed and results presented in the previous sections of this thesis, and provides an outlook over the possible topics for future related work.

6.1 Resume

This thesis examined how variants of the Max-Min Ant System algorithm, based on the Ant Colony Optimization metaheuristic, can be applied to dynamic shortest path problems. It began by demonstrating that an ant colony is needed to solve shortest path problems in an expected polynomial number of iterations (as performance of ACO algorithms is typically measured in iterations, not basic operations). The choice of parameters in the $\text{MMAS}_{\text{SDSP}}$ algorithm was analyzed, and it was shown that choosing the pheromone bounds $\tau_{\min} \leq 1/(\ell\Delta)$ and $\tau_{\max} = 1 - \tau_{\min}$, where ℓ is the maximum length (in arcs) of any shortest path to the destination vertex t and Δ is the maximum vertex degree in the graph, allows construction of a specific path with one arc with pheromone value $\tau_{\min}$, and up to ℓ arcs with pheromone value $\tau_{\max}$ with probability $\Omega(1/\tau_{\min})$. Construction of paths of this type is then shown to be the primary source of progress during the optimisation, which yielded $O\left(\ell/\tau_{\min} + \ell \log(\tau_{\max}/\tau_{\min})/\rho\right)$ and $\Omega\left(\ell/\tau_{\min}\right)$ upper and lower bounds on the expected number of iterations to rediscover all shortest paths after a specific one-time change to the weight function was made.

The optimisation process was then characterized as a series of discovery and freezing phases, and the effects of starting λ ants at each vertex in the λ-MMAS and λ-MMAS_{ib} algorithms were examined. It was shown that $\lambda = \Omega(1/\tau_{\min})$ allows the discovery phases to be completed in expectation in a constant number of iterations, significantly reducing the expected number of iterations required to rediscover the shortest paths. While starting even more ants could allow λ-MMAS to discover shortest paths with up to k non-reinforced arcs, it was shown that doing so would require simulating a number of ants exponential with respect to k. Finally, it was also shown that starting $\Omega(\log n)$ ants at each vertex is sufficient to allow λ-MMAS_{ib}, using iteration-best reinforcement (where the paths x_v^* are only track the best path constructed during the *current* iteration), to rediscover the shortest paths in expected polynomial time if the evaporation rate ρ was at least a constant.

Finally, performance of λ-MMAS was examined in several situations where the weight function was changed regularly. It was shown that λ-MMAS with

$\lambda = \log^2 n$ is able to maintain a high probability of constructing the correct shortest path in each iteration, for any polynomial number of iterations, when a fitness function alternates between two weight functions every iteration, as long as the difference between the shortest paths of the two weight function is relatively minor, and the evaporation rate ρ is not too high. This is accomplished by keeping the pheromone values on the oscillated arcs close to $1/2$. It was then shown that if the fitness function switches between weight functions *permanently*, rather than oscillating between function, evaporation rates that are *too* low may hinder the optimisation process, causing λ-MMAS to lose track of the correct shortest paths for any choice of λ that is polynomial with respect to n. Finally, it was demonstrated that λ-MMAS with $\lambda = \log^2 n$ ants is not able to keep track of a fitness function that quickly oscillates between two weight functions when the Hamming distance between their shortest paths is large, by showing a particular example where, even if the pheromone values were kept close to $1/2$, $\log^2 n$ ants would not be able to construct the shortest paths with overwhelming probability.

The λ-MMAS and λ-MMAS$_{\text{ib}}$ algorithms were then implemented in C, and the implementation was used to provide additional empirical detail to the results predicted in the theoretical analyses by simulating specific scenarios using small graphs. It was shown that using λ-MMAS with $\lambda = 3$ ants is able to the pheromones on an oscillated arc from freezing for significantly longer than with $\lambda = 2$ ants (for which the number of iterations seems to scale reciprocally with the evaporation rate ρ). A more detailed view of the λ-MMAS behavior when the fitness function oscillates between weight functions with shortest paths that have no arcs in common was presented, revealing that if the oscillation is slow enough to allow some of the pheromone values to freeze to favor the correct shortest path arcs, this freezing behavior propagates in waves through the graph – with some pheromone values having been observed to freeze in counter-phase to the actual shortest paths.

6.2 Future work

Some of the bounds presented in the theorems in this thesis could likely be refined further. In particular, it may well be the case that $\log n$ ants are sufficient for the results of Theorem 8, which could be approached using the negative drift theorem (see e.g. [10, Theorem 4.9] or [12, Theorem 2.12]), demonstrating that there exists a drift towards pheromone value $1/2$, that at least a linear number of double-steps away from $1/2$ are required for pheromone values to exit the desired range, and that no large jumps in pheromone value are possible – then, per the theorem, the expected time before the pheromone values first exit the desired range is exponential with high probability.

Theorem 8 could be investigated for fitness functions that switch between k different weight functions (which all differ by one choice) every iteration to determine the influence of k on the required number of ants λ.

The effects of having a large Hamming distance between shortest paths in an oscillation could be examined more closely – in particular, the nature of a wave-like propagation of pheromone values through the graph shown by experimental results is interesting. It would be interesting to consider theoretical basis for this behavior, considering it sometimes updates pheromones in a non-intuitive fashion (changing to favor precisely the wrong arc), as well as experimentally check whether the "waves" continue to exist in larger graphs, or for other pairs of weight functions with the same property of not having the shortest paths share any arcs.

It may also be interesting to consider whether the $\text{MMAS}_{\text{SDSP}}$ algorithm could be improved in any way that affects the expected optimisation time: as presented in Algorithm 1, the algorithm ignores additional information that is available for shortest path problems (e.g. the weights of the subpaths of the constructed path from each vertex), and uses a particularly simple pheromone reinforcement method, which only alters the pheromone values on arcs leaving a vertex v based on the path x^*_v, and not any of the other paths that might pass through v.

Finally, the structure of the $\text{MMAS}_{\text{SDSP}}$ algorithm makes it relatively easy to adapt it to handle multi-objective shortest path problems, where each arc might have several different types weights associated with it, simply by adjusting how fitness functions map the solutions to fitness values (and how those are compared). However, the key property that allowed polynomial expected time optimisation in the single-objective setting no longer applies – shortest paths cannot always be built by adding a single non-reinforced arc to an already known shortest path[14]. Furthermore, multi-objective shortest path problems are known to be NP-hard (per [1]), so efficient optimisation might not be possible using any algorithm. Yet multi-objective optimisation problems are common in nature, and it can be argued that even ant food gathering is one such problem, balancing the distance to food source with the amount of available food and other factors. It may therefore be worth examining if a pheromone-based approach can also be applied to some multi-objective shortest path problems, in a fashion similar to how the performance of a simple evolutionary algorithm on a multi-objective shortest path problem was analyzed in [9].

[14]For an example, consider the problem of finding the cheapest flight connection to reach Reykjavík by midnight: each arc would have both a time and a cost weight. It is not possible to extend already known cheapest connections that satisfy the arrival time requirement by adding additional flights, as doing so might violate the arrival time requirement.

References

[1] M. R. Garey, D. S. Johnson: *Computers and intractability: A guide to the theory of NP-completeness*, W. H. Freeman, New York, ISBN 978-0716710455, 1979.

[2] M. Dorigo: *Optimization, Learning and Natural Algorithms* (in Italian). PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1992.

[3] M. Dorigo and G. Di Caro: *Ant Colony Optimization: A New Meta-Heuristic*. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, editors, IEEE Congress on Evolutionary Computation - CEC'99, pages 1470-1477. IEEE Press, Piscataway, NJ, 1999.

[4] M. Dorigo, T. Stützle: *Ant Colony Optimization*. MIT Press, Cambridge, MA, ISBN 978-0262042192, 2004.

[5] T. Jansen, K. A. De Jong, I. Wegenerr: *On the Choice of the Offspring Population Size in Evolutionary Algorithms*, in Evolutionary Computation, Vol. 13 Issue 4, Winter 2005, pp. 413-440, 2005.

[6] N. Attiratanasunthron, J. Fakcharoenphol: *A running time analysis of an Ant Colony Optimization algorithm for shortest paths in directed acyclic graphs*, Information Processing Letters 105 (2008) pp 88-92, 2008.

[7] L. S. Buriol, M. G. C. Resende, M. Thorup: *Speeding Up Dynamic Shortest-Path Algorithms*, INFORMS Journal on Computing, Vol. 20, No. 2, Spring 2008, pp. 191-204, 2008.

[8] M. Dorigo, T. Stützle: *Ant Colony Optimization: Overview and Recent Advances*, in Handbook of Metaheuristics (eds.: M. Gendreau and J.-Y. Potvin), volume 146 of International Series in Operations Research & Management Science, chapter 8, pages 227-263. Springer, New York, 2010.

[9] C. Horoba: *Exploring the runtime of an evolutionary algorithm for the multi-objective shortest path problem*, Journal of Evolutionary Computation, Vol. 18, Issue 3, Fall 2010, pp. 357-381, 2010.

[10] F. Neumann, C. Witt: *Bioinspired Computation in Combinatorial Optimisation*, Natural Computing Series, Springer, ISBN 978-3-642-16543-6, 2010.

[11] F. Neumann, D. Sudholt, C. Witt: *A few ants are enough: ACO with iteration-best update*, in Proceedings of Genetic and Evolutionary Computation Conference (GECCO '10), ACM Press, pp. 63-70, 2010.

[12] A. Auger, B. Doerr (eds.): *Theory Of Randomized Search Heuristics*, World Scientific Publishing, ISBN 978-9814282666, 2011.

[13] W. Gutjahr: *Ant colony optimization: recent developments in theoretical analysis*, in Theory of Randomized Search Heuristics (eds.: A. Auger, B. Doerr), World Scientific Publishing, pp. 225-254, 2011.

[14] D. Sudholt, C. Thyssen: *A Simple Ant Colony Optimizer for Stochastic Shortest Path Problems*, Algorithmica, Online First^{TM}, DOI 10.1007/s00453-011-9606-2, 2011.

[15] B. Doerr, A. Hota, T. Kötzing: *Ants easily solve stochastic shortest path problems*, in Proceedings of Genetic and Evolutionary Computation Conference (GECCO '12), pp. 17-24, 2012.

[16] T. Kötzing, H. Molter: *ACO beats EA on a Dynamic Pseudo-Boolean Function*, 12th International Conference on Parallel Problem Solving From Nature, pp. 113-122, 2012.

[17] J. E. Rowe, D. Sudholt: *The choice of the offspring population size in the* $(1, \lambda)$ *EA*, in Proceedings of Genetic and Evolutionary Computation Conference (GECCO '12), pp. 1349-1356, 2012.

[18] D. Sudholt, C. Thyssen: *Running Time Analysis of Ant Colony Optimization for Shortest Path Problems*, Journal of Discrete Algorithms, Volume 10 (2012), pp. 165-180, 2012.

A Implementation details

This appendix provides more detailed instructions on how the implementation described in this thesis can be compiled and used to obtain experimental data.

A.1 Using the implementation

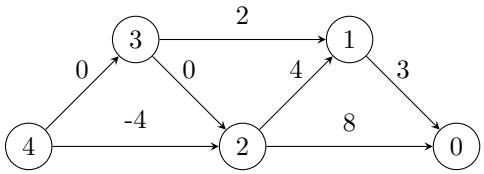
The implementation is written in C99, and has been tested to compile with GCC or LLVM/CLANG.

1. The POSIX threads (pthreads) library is required.
   This is typically available on any Linux/Unix operating system. Pthreads-w32 may be useful on Windows: `http://sources.redhat.com/pthreads-win32/`
2. Lua shared libraries must be available to the C compiler and linker.
   Lua source code can be downloaded form `http://www.lua.org/ftp/`, and includes Makefiles to compile and install the required libraries for most platforms. The implementation has been tested against Lua 5.1.5.
3. The included C source code should be compiled and linked against Lua, pthreads, and the standard math libraries.
   A Makefile is included in the implementation to automate this on some systems.
4. The simulator is invoked by specifying a path to a serialized initial graph and a path to the Lua script to control the simulation:
   `$ ./aco graph.wf script.lua`
   Output is then controlled by the specified Lua script file.
   A number of sample Lua scripts for the experiments presented in Section 5.2 is included. These create their own graph files, and can be started by running them with the standalone Lua interpreter:
   `$ ./lua exp1.lua`

A.2 Graph format example

The initial graph is always read from a serialized representation stored in a file. The Lua script can subsequently modify this graph by altering any existing arc weights, but cannot insert new arcs.

The graph files consist of whitespace-separated numbers: N, the number of vertices in the graph, $\Delta_1, \Delta_2, \ldots, \Delta_{N-1}$ the out-degrees of vertices 1 through $N-1$ (vertex 0 is by convention the destination vertex and has no outgoing arc), and pairs of numbers $H_i W_i$, specifying the head vertex index and arc weight for each arc in the graph. The H_i, W_i pairs are sorted in ascending order of the tail and head vertex indices. This encoding scheme is illustrated by the example in Figure 12




```
5 1 2 2 2
0 3
0 8 1 4
1 2 2 0
2 -4 3 0
```

Figure 12: A simple graph and its serialization.

A.3 Functions available to the Lua environment

Standard Lua table, string, and math libraries are available. The command line arguments to the simulator are accessible through the global `arg` table, indexed such that the simulator executable file path is in `arg[0]`, and the remaining ascending integer indices reflect command line arguments (the first of which is the path to the graph, and the second the path to the Lua script).

The following functions can be used to control algorithm parameters:

`SetNumAnts(numAnts)`
: Sets the number of ants λ started at each vertex.

`SetNumThreads(numThreads)`
: Sets the number of parallel threads used to perform the simulation.

`UseBestSoFarReinforcement(useBF)`
: If `useBF` is truthy, best-so-far reinforcement is used; otherwise, iteration-best reinforcement is used (i.e. the paths x_v^* only reflect the best path of the current iteration).

`SetPheromoneBounds(min[, max])`
: Sets the pheromone bounds to provided values; does not alter existing pheromone values until the next pheromone update. If `max` is omitted, it defaults to $\tau_{\max} = 1 - \tau_{\min}$.

`SetEvaporationRate(rho)`
: Sets the evaporation rate ρ.

`SetCallback("OnPathUpdate", func)`
: Sets `func` to be called after any iteration in which any of the best-so-far paths x_v^* changed. Only one callback can set at a time.

`SetCallback("OnIteration", iterval, func)`
: Sets `func` to be called whenever (iteration mod `interval`) $= 0$. Only one callback can set at a time.

The following functions return information about the current state of the simulation:

`iteration = GetIteration()`
: Returns the total number of iterations performed.

`numVertices, numArcs = GetGraphSize()`
: Returns the number of vertices and the number of arcs in the current graph.

`dst, weight, pheromone = GetReinforcedArcInfo(src)`
: Returns information about the reinforced arc from vertex with index *src*: index of the destination vertex, total weight of the reinforced path, and the current pheromone value on the reinforced arc. If no such path exists, *dst* and *pheromone* are `nil`.

`value = GetPheromoneValue(src, dst)`
: Returns the pheromone value on the (src, dst) arc, or `nil` if no (src, dst) exists.

The following functions modify the current state of the simulation:

`SetPheromoneValue(src, dst, value)`
: Sets the pheromone value on the (src, dst) arc to $\min(\tau_{\max}, \max(\tau_{\min}, \texttt{value}))$.

`SetArcWeight(src, dst, weight)`
: Sets the weight on the (src, dst) arc to `weight`.

`StopSimulation()`
: Halts the simulation; no further iterations will be performed, and the program will exit after the Lua callback completes.