TRAFFIC SIGNAL CONTROL WITH ANT COLONY OPTIMIZATION

A Thesis
presented to
the Faculty of California Polytechnic State University,
San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Electrical Engineering

By
David Renfrew
2009

COMMITTEE MEMBERSHIP

TITLE:	Traffic signal control with ant colony optimization
AUTHOR:	David Renfrew
DATE SUBMITTED:	November 2009
COMMITTEE CHAIR:	Helen Yu, Associate Professor
COMMITTEE MEMBER:	Fred DePiero, Professor
COMMITTEE MEMBER:	Franz Kurfess, Professor

ABSTRACT

Traffic signal control with ant colony optimization

David Renfrew

Traffic signal control is an effective way to improve the efficiency of traffic networks and reduce users' delays. Ant Colony Optimization (ACO) is a metaheuristic based on the behavior of ant colonies searching for food. ACO has successfully been used to solve many NP-hard combinatorial optimization problems and its stochastic and decentralized nature fits well with traffic flow networks. This thesis investigates the application of ACO to minimize user delay at traffic intersections. Computer simulation results show that this new approach outperforms conventional fully actuated control under the condition of high traffic demand.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

With the ever-increasing traffic demand, congestion has become a serious problem in many major cities around the world. ATMS (advanced traffic management system) is a systematic effort toward the design of an integrated transportation system with new technologies. By regulating the traffic demand at each intersection in the network, the goal is to avoid traffic conflict and shorten the queue length at a stop light.

Many different approaches to the traffic signal control problem have been proposed by researchers over the years. Some of the earliest, large scale adaptive traffic signal control systems, such as TRANSYT (traffic network study tool) [1], SCOOT (split, cycle and offset optimization technique) [2], and SCATS (Sydney coordinated adaptive traffic system) [3], utilize pre-calculated off-line timing plans for signal cycles based on the current traffic conditions. More recent developments in traffic signal control employ artificial intelligent technology, such as neural networks and fuzzy logic [4]. Algorithms using Petri nets and Markov decision control have also been investigated in recent years [5].

Ant colony algorithm is a meta-heuristic approach for solving computationally hard combinatorial optimization (CO) problems [6] [7]. Inspired by the behavior of the ants in real world, ant colony algorithm is a multi-agent system, in which each single agent is called an artificial ant. It is one of the most successful examples of swarm intelligent systems and has been applied to solve many different types of problems, including the classical traveling salesman problem, path planning and network routing.

In nature, when searching for food, real ants wander randomly until they find food [8]. As an ant returns to the colony with food, it deposits pheromone, a chemical used for communication. These pheromone trails guide other ants as they continue their search for food. As more pheromone is deposited, the ants' paths become less random and are biased toward the paths with higher pheromone concentration.

In the ant colony algorithm, artificial ants search the solution space probabilistically to create candidate solutions. These candidate solutions are then evaluated and used to update pheromone concentrations. Over time the pheromone concentrations on paths evaporates. Only the paths that have been reinforced with additional pheromone are left.

In this research, a new approach to finding the optimal signal timing plan for a traffic intersection is investigated using ant colony optimization algorithms. The ACO is used with a rolling horizon algorithm to achieve real-time adaptive control. Computer simulation results indicate that this new approach is more efficient than traditional fully actuated control (discussed in Section 5.3), especially under the conditions of high, but not saturated, traffic demand.

This thesis is organized as follows: chapter two presents a literature review of current traffic control strategies and new developments in the field. Chapter three explains traffic dynamics, assumptions made for the traffic signal problem and the calculation of vehicle delays. Chapter four gives background on Ant Colony Optimization (ACO) algorithms. Chapter five explains how ACO is implemented in the traffic signal control problem. Chapter six contains simulation results. The convergence rates of ant colony algorithms to finding optimal solutions are investigated. Then

comparisons of vehicle delays in ACO algorithm and traditional fully-actuated control are made. Chapter seven gives conclusions and an outline of future works.

Chapter 2 Literature Review

Traffic networks are an integral part of any city's infrastructure, but increased vehicle use causes traffic congestion, leading to decreased flow rates. Two major causes of congestion are overall demand and disturbances in traffic flow from accidents, special events and illegal parking. Once traffic movements are saturated, traffic flow at the upstream link stops and vehicles cannot cross intersections on green lights. Additional congestion is then created on other movements, leading to gridlock and devastating urban traffic flow. Congestion leads to excess vehicle delays, reduced safety and increased air pollution and petroleum use. Additionally, congestion can cost governments billions of dollars a year [9].

Expansion of traffic networks is expensive and due to available resources, it is often not feasible. As a result, increasing the efficiency of traffic networks with current facilities is essential. Improving traffic signal control strategies is a very effective way to improve traffic management. Advances in low power sensors and actuators, as well as low cost and reliable means of communication and computers, have made real-time adaptive traffic control systems feasible [5].

There are many difficulties in dealing with traffic control. Traffic movements are stochastic and non-linear; so many conventional control techniques cannot yield optimal results. Additionally, traffic conditions can change quickly, so control strategies must be highly responsive. As traffic networks grow in size, finding the optimal strategy becomes a complex combinatorial problem, making real time implementation very difficult. Thus, advanced techniques in control and optimization must be employed.

For large scale networks, traffic control systems can be classified as centralized or decentralized control [5]. To implement traffic control on large networks, the network is divided into smaller subsystems. In centralized control systems, a central controller creates the control policy and sends control signals to each subsystem. Centralized control can achieve global optimality because of the high level of communication between subsystems, but it is computationally intensive and very sensitive to a malfunctioning central controller or broken communication links. The central controller is also unresponsive to time-varying traffic dynamics.

Due to centralized control's limitations, recent research has focused on decentralized control. In decentralized control, neighboring subsystems communicate for coordination purposes; but control decisions are made on a local level, involving only a few signals [10]. This approach is less computationally complex, more robust and responds to changes in traffic dynamics quickly. Because optimization is made on a local level, the global optimal solution might not be found.

To evaluate the effectiveness of control strategies, different performance measures are available. Some common measures are average vehicle delay, intersection queue lengths and number of vehicle stops. These measures are interrelated but optimizing one does not necessarily lead to optimization of the other criteria [11].

Traffic control strategies fall into one of three categories: fixed time, semi-actuated and fully-actuated control. Fixed time control is an open loop control strategy because signal cycles are computed off-line and do not consider current traffic dynamics. The preset cycle lengths are determined from past traffic data and heuristic information. Signal coordination is easily achieved with fixed time controllers. For this reason, fixed

time control is generally used in high volume business districts where signal coordination is required and turning lanes are not given their own green light phase [9]. If traffic flow fluctuates a lot, then fixed time controllers may cause long delays. Intensive off-line work is required to compute signal splits, offset and cycle length. Additionally, control rules must be monitored and updated due to long term changes in traffic dynamics.

Actuated control, also called traffic responsive control, is a closed loop control strategy because control policies respond to current traffic demand. In semi-actuated control, one or more, but not all of the movements are actuated. In fully actuated control, all movements are actuated. Signal actuation can be achieved in many different ways, making optimal actuation methods an area of active research.

Traffic networks are classified into three categories: isolated intersections, arterial streets (including freeways), and closed network intersections. On isolated intersections, arrivals are assumed to arrive randomly and are uncorrelated with the arrivals at other intersections. In arterial and closed networks, vehicle arrivals between neighboring intersections must be considered and the signals at neighboring intersections must be coordinated for optimal control.

For the remainder of this chapter, popular traffic control algorithms are presented. These methods utilize historic traffic data, expected flow rate and statistical distribution on arrivals as well as standard control practices to create traffic control policies.

SCOOT (Split, Cycle and Offset Optimization Technique) [2] and SCATS (Sydney Coordinated Adaptive Traffic System) [12] are on-line control strategies based on the off-line optimization techniques. Detectors monitor traffic flows and predict future arrivals by creating flow profiles. The flow profiles are used to evaluate incremental

changes to the signal's splits, offsets and cycle times. If the changes are beneficial they are implemented by the controller.

Dynamic programming is a mathematical technique for solving optimal control problems [13]. Time is broken into small intervals and the optimal control policy for each interval is found. Dynamic Programming finds the optimal policy, but is often very computationally intense and requires more information on future arrivals than is typically available. These limitations make real-time implementation of Dynamic Programming difficult; but its off-line results can be used for comparison with other methods. Control strategies can be evaluated by how well they approximate the results of dynamic programming, but with less information and computation.

Two approaches for on-line optimization that are similar to off-line dynamic programming are binary choice logic and sequential approach. Binary choice logic, proposed by Miller [14], divides time into short, successive fixed intervals. At the beginning of each interval a choice is made to extend the signal or change the signal. The drawback of this technique is the time periods are very short (3-6 seconds), so optimal performance over longer time periods is not guaranteed.

In sequential approach, a longer period of time, generally 50-100 seconds, is considered. During this time period, 1 to 3 signal changes in one signal cycle can be made. All possible signal cycles over this time interval are sequentially evaluated to find the optimal switching times. The optimal policy is then implemented over the entire time period. This technique yields close to optimal results when the system is in steady state. But, because the decisions are made over longer periods of time this technique does not respond fast enough to time-varying traffic dynamics.

The advantages of the above two techniques are combined in model-based optimization methods. OPAC, PRODYN, CRONOS and RHODES are examples of rigorously developed model-based methods [9]. These models incorporate a rolling horizon for optimization. A long time interval (usually 60 seconds) is considered and the optimal control policy is found but only implemented over a short period (usually around 4 seconds). Each algorithm uses a similar length for the rolling horizon but different methods to find the optimal control policy. After the signal has been implemented for the shorter interval, the process is repeated.

More recent traffic research has introduced fuzzy logic and artificial intelligence into traffic control. Fuzzy logic has been successfully applied to a single intersection and groups of intersections [15]. Fuzzy logic controllers create membership functions between their inputs and output. Then the controller chooses an output that is acceptable to all membership functions. In traffic control, the inputs are the intersection's current state, which includes the waiting time of vehicles that are currently waiting, queue length and arrival rates. The output is the traffic control law, usually green extension times or optimal cycle time and splits [4]. The use of fuzzy controllers allows for different objectives to simultaneously be optimized by specifying a minimum level of acceptability for each objective. Optimizing the fuzzy logic rule bases is often difficult, but neural networks and genetic algorithms can be used to efficiently create the rule bases.

Another new technique in traffic network control is artificial neural networks. Neural networks are very powerful in mathematical modeling, and their nonlinear mapping ability makes them ideal for predicting the highly nonlinear traffic flow [16]. Accurate traffic prediction and modeling is essential for choosing signal switch times in

optimal control. Neural networks use sensors to measure vehicle arrivals and departures along with the current queue to predict future queue lengths. The advantage of neural networks is that no assumption on an analytic model for traffic flow is made. Additionally, neural networks can easily be integrated into hardware. Unfortunately, neural networks design procedures can be lengthy and size of an effective network is hard to determine. Neural network training can take a long time and require a large amount of data [17].

The use of neural networks in traffic modeling can be improved by adding fuzzy logic. The fuzzy logic is first used to classify traffic patterns into different sets. Then each set has its own set of rules for traffic prediction. This allows for better generalization and faster training times over conventional neural networks [18].

In this research, Ant Colony Optimization (ACO), a technique in Swarm Intelligence, is used to for the adaptive control of an isolated intersection. A rolling-horizon approach with variable length is employed.

ACO algorithms have successfully been applied to many computationally complex combinatorial problems; the traffic signal problem is addressed very naturally as a combinatorial optimization problem. Traffic signals form large, complex networks and advanced methods must be used to optimize signals lengths.

The ability of the ACO to incorporate heuristic information about traffic networks makes it efficient. Additionally, ant colony optimization has successfully been applied to other traffic related problems, such as the vehicle routing problem (VRP), with positive results.

Chapter 3 Traffic Dynamics

In this chapter, the traffic signal problem and traffic terminology is presented. Then the rules for vehicle arrivals and departures are established. The assumptions on vehicle arrivals and departures, as well as traffic flows are presented. Finally, the method of evaluating traffic signals by computing vehicle delays is presented.

3.1 Problem Statement

Consider a traffic intersection with four external approaches. Movements 1 and 3 are opposite each other so they share green times; similarly movements 2 and 4 also share green times. For simplicity, each movement consists of a single lane and turning lanes are not considered. Video detectors are located at the intersection to count the queue lengths and there are no detectors outside the intersection. An estimate on the volume of traffic is assumed (see [5] and [19]). The vehicles are homogenous and leave the intersection at the same speed.

The objective is to find the traffic signal switch times that minimize the average delay of the vehicles at the intersection.

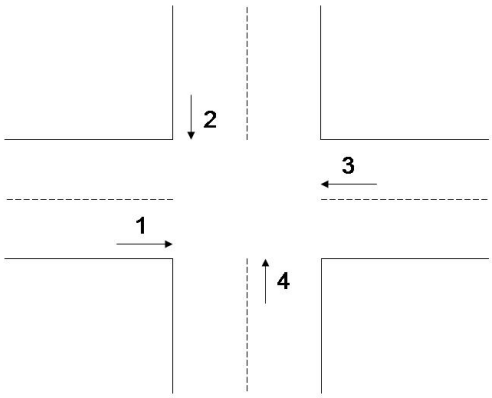

Figure 3-1. An isolated intersection with four movements

3.2 Traffic Terminology

An intersection consists of a number of movements and a crossing area. The number of vehicles waiting on a movement is called the queue. A signal cycle is a complete series of green lights for each movement. Its length is called the cycle time. The split is the percentage of green time for a movement with respect to the total cycle time. The all red time is the length of the time when all movements have a red signal. The all red time occurs between phase changes for safety. The minimum distance between vehicles is measured in seconds and called the minimum headway. Once a vehicle arrives at the intersection, the time it takes for the next vehicle to arrive is called the inter-arrival time. A vehicle's delay is its time of departure minus its time of arrival.

3.3 Departures

At a given time, t, the queue length on movement i is denoted as $q^i(t)$. The number of vehicles initially in the queue that leave movement i during a time interval (t_1, t_2) is denoted $q^i_{out}(t_1, t_2)$. The time intervals correspond with signal phases. The output $q^i_{out}(t_1, t_2)$ is a function of the signal choice and the queue length at t_1.

$$q^i_{out}(t_1, t_2) = \begin{cases} \min[q^i(t_1), 1 + Int\left(\dfrac{t_2 - t_1}{hw}\right)] & \text{if } u_{(t_1,t_2)} = \text{ green} \\ 0 & \text{if } u_{(t_1,t_2)} = \text{ red} \end{cases} \tag{3.1}$$

where hw is the headway between vehicles as they leave the intersection, $u_{(t_1,t_2)}$ is the signal choice and $Int(\cdot)$ gives the integer part of its argument. The output function means that when a signal turns green, the first car in the queue leaves immediately. Then, each successive vehicle leaves the intersection hw seconds after the vehicle in front of it until all vehicles are released or the traffic light changes to red.

3.4 Arrivals

Vehicles arrive from the external links randomly and uncorrelated, making the Poisson process a good model for the arrivals. In a Poisson process, inter-arrival times follow the exponential distribution [20]. Meaning the probability density function on the inter-arrival times is:

$$f_X(x) = \lambda e^{-\lambda x} u(x) \tag{3.2}$$

where λ is the arrival rate in vehicles per hour per movement and $u(\cdot)$ is the unit step function. The step function is required because negative inter-arrival times do not make sense. A graph of the exponential probability density function with $\lambda = 1/6$ is shown in Figure 3-2.

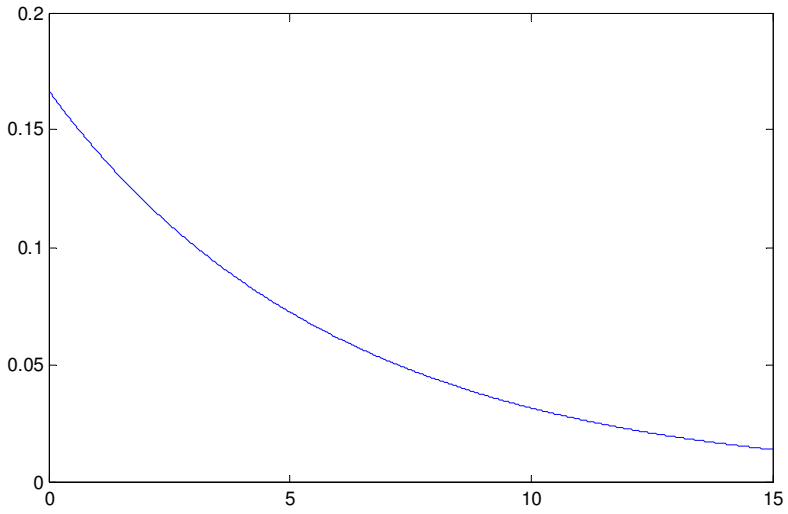

Figure 3-2. Exponential probability density function with $\lambda = 1/6$

As shown in equation (3.2), the exponential distribution allows for instantaneous inter-arrival times. Due to geometric and physical considerations of traffic network, there must be a minimum headway between vehicles. To avoid this problem, the shifted

exponential distribution is used for inter-arrival times [21]. That is, the probability density function for the inter-arrival time is:

$$f_X(x) = \frac{\lambda}{1-\lambda hw} e^{-\frac{\lambda}{1-\lambda hw}(x-hw)} \mathrm{u}(x-hw), \tag{3.3}$$

where hw is the minimum headway between vehicles in seconds. The graph of the shifted exponential probability density function with $\lambda = 1/6$ and $hw = 2$ is shown in Figure 3-3.

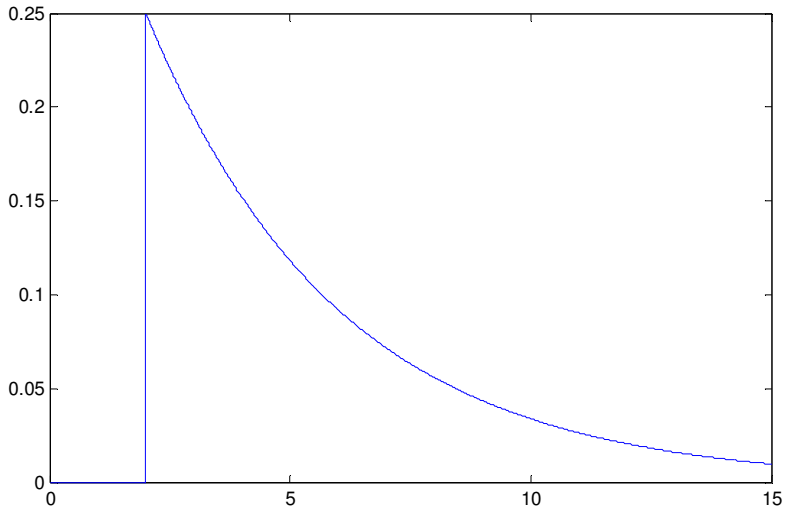

Figure 3-3. Shifted Exponential probability density function with $\lambda = 1/6$

This probability distribution acts very similar to the exponential distribution but the minimum inter-arrival time is now hw, instead of 0. If the headway is set to zero this formula gives the standard exponential distribution. The shifted exponential distribution gives the same expected inter-arrival time of $1/\lambda$ as with the exponential distribution with arrival rate λ. It should be noted that in order for this probability density to make sense $1 - \lambda hw > 0$. This requirement means that the expected inter-arrival time is greater than the minimum headway.

Vehicle arrival times are generated by:

$$at_2 = at_1 + hw - \left(\frac{1}{\lambda} - hw\right) * \mathrm{Log}(r) \tag{3.4}$$

where at_2 is the next arrival time, at_1 is the previous arrival time, Log is the natural logarithm function and r is a uniformly distributed random number in $(0, 1)$ [22].

The expected number of arrivals during a given time interval is required to compute the expected vehicle delay during a signal cycle. This is difficult to do with the shifted exponential distribution because the minimum headway causes the distribution to lose its Markov property [21]. The probability of an arrival at a given time, t, is not solely based on the state of the system at t, but also on the arrivals during the time interval $[t - hw, t)$. Fortunately, if the signal length is several times larger than the minimum headway, then the expected number of arrivals can be approximated by the Poisson distribution. That is, the probability of n vehicles arriving in Δt seconds is approximated by:

$$P(n) = \frac{(\lambda \Delta t)^n e^{-\lambda \Delta t}}{n!} \tag{3.5}$$

where n is a non-negative integer representing the number of arrivals, and Δt is the duration of time period. The Poisson distribution gives the probability of a given number of arrivals during a time interval of a Poisson process. This approximation is sufficient because the probability densities of the exponential and shifted exponential distributions behave similarly and have same the expected inter-arrival time; so they give similar traffic dynamics.

In the Poisson distribution, the expected number of new arrivals in Δt seconds is $\lambda \Delta t$.

3.5 Traffic Flow

When the signal turns green for a movement, vehicles are released from the queue. The first vehicle leaves when the traffic light changes. Then each of the following vehicles leaves hw seconds after the vehicle before it. If additional vehicles arrive before the queue is empty, then they are added to the end of the queue. New arrivals are also released hw seconds after the vehicle in front of them is released. This process continues until either the queue is empty or the traffic light changes to red. If the queue is empty before a switch to red then additional vehicles pass freely through the intersection with zero delay. At the end of the green phase, the signal is red on all movements. The length of this period is the all red time. When a movement has a red signal, new arrivals are added to the queue and wait until the next green phase to be released.

3.6 Vehicle Delays

Ant Colony optimization is used to determine the traffic signal control that minimizes vehicle delay at the intersection. At the beginning of each signal phase, the algorithm evaluates candidate signal cycles by computing the total expected delay of the signal cycle. This computation has a deterministic and probabilistic part. The deterministic part is the delay of the vehicles already in the queue, and the probabilistic part is the expected delay of future arrivals. Because the actual arrival times of future vehicles is unknown, only the expected delay of future vehicles can be minimized. In order to calculate the expected delay, the arrival rate of new vehicles is assumed, as stated in the problem statement. The probability distribution of their inter-arrival times is also a necessary assumption to compute the expected delay.

3.7 Delay of vehicles initially in queue

Given a green phase of length $(t_2 - t_1)$, $q^i_{out} = q^i_{out}(t_1, t_2)$ of the initial vehicles will be released on the green movements. Let $n \leq q^i_{out}$ denote the position in the queue of a vehicle that will be released on the current queue. At the beginning of a green phase, the current delay of the n^{th} vehicle in the queue is $t_1 - at_n$, where at_n is the arrival time of the n^{th} vehicle. After the traffic light turns green, the additional delay is $(n-1)hw$ seconds. Thus, the combined delay of the q^i_{out} vehicles is

$$\sum_{n=1}^{q^i_{out}} (\text{wait time of car n}) = \sum_{n=1}^{q^i_{out}} (n-1)hw + (t_1 - at_n)$$

$$= [q^i_{out}(q^i_{out} - 1)/2]hw + \sum_{n=1}^{q^i_{out}} (t_1 - at_n). \tag{3.6}$$

3.8 Delay of vehicles not released on current phase

As new cars arrive at an intersection, they are added to the movement's queue. If they arrive during a sufficiently long green phase they will be released, otherwise they must wait to be released during a future green phase. The expected delay for a vehicle that is released during the phase it arrives is calculated differently than those that are not.

First, consider a signal phase in which vehicles arrive but do not leave. Let Δt denote the length of this phase. This situation occurs on a red phase or an over-congested green phase that does not allow new arrivals to exit the intersection.

In a Poisson process, the expected number of new arrivals in Δt seconds is $\lambda \Delta t$. Since the probability distribution of the inter-arrival times is identical, vehicle arrivals are uniformly distribution in the phase. So the expected delay for each new vehicle is $\Delta t/2$

[23]. Multiplying the expected number of vehicles by the average expected delay gives the expected total delay for new arrivals during an interval of length Δt as

$$\lambda \Delta t \times \frac{\Delta t}{2} = \frac{\lambda \Delta t^2}{2}. \tag{3.7}$$

3.9 Delay of vehicles released on current phase

Alternatively, consider a green phase where all new arrivals exit the intersection on the current phase. The delay of a new arrival is dependant on what position in the queue the vehicle arrives, similar to the delay of vehicles initially in the queue at the beginning of a green phase. Once a vehicle arrives, its delay is the minimum headway multiplied by its position in the queue. Therefore, future expected queue lengths are necessary to calculate the expected wait time of future arrivals.

As vehicles arrive and depart the length of the queue changes. If vehicles inter-arrival times follow the exponential distribution, then nearly instantaneous inter-arrival times occur. Short inter-arrival times cause fluctuations in the queue and make its length increase, making the expected queue difficult to determine. Fortunately, instantaneous inter-arrival times are not physically possible. As a result, an approximation to the exponential distribution on the vehicle arrivals is used to accurately approximate average expected vehicle delays.

This approximation is made by partitioning the signal cycle into time intervals of length equal to the minimum vehicle headway. During each interval a vehicle arrives with probability λhw. The probability of having multiple arrivals in an interval is zero. The exponential distribution can be derived from this approximation by taking the length of the intervals to zero, giving a continuous probability distribution for the inter-arrival

times [24]. Since the approximation does not take this limit, the minimum headway requirement is preserved.

During each time interval one car from the queue will leave, according the established rules Section 3.3, and at most one car arrives. This makes the expected queue length easier to describe because queue lengths cannot increase.

Depending on the first new vehicle's arrival time, it can wait up to $(q-1)hw$ seconds before it reaches the front of the queue. Since vehicles depart faster than they arrive, the delay of each successive vehicle decreases. The expected delay decreases linearly until a new arrival waits close to zero seconds before it reaches the front of the queue. So the average time to the front of the queue is $(q-1)hw/2$. Additionally, a vehicle must wait minimum headway seconds once it reaches the front of the queue before it is released. So the average time each car waits until it is released is:

$$(q-1)hw/2 + hw = (q+1)hw/2. \tag{3.8}$$

Equation (3.8) gives the average wait time of new vehicles, but the expected number of new arrivals before the queue is empty must also be determined. This is done by computing the expected time to an empty queue and then multiplying this time by the vehicle arrival rate.

3.10 Time until queue is empty

The expected time required to empty the queue is $\frac{(q-1)hw}{1-\lambda hw}$. This time is computed iteratively. First, the time until the vehicles initially in the queue are released is computed. Then, the expected number of new arrivals during this time interval is computed, along with the time until these new vehicles are released. This process of

computing the expected number of new arrivals during the last groups release times continues. Recall that the expected number of vehicles is not an integer and will get smaller with each iteration. Then, the time intervals are added up to give the expected time when the queue is empty.

The initial vehicles are released at time $t = t_1 + (q-1)hw$, and $\lambda(q-1)hw$ additional vehicles are expected to arrive in this time. An additional $\lambda(q-1)hw^2$ seconds are required to release these vehicles. This cycle of vehicles arriving and being released continues. The sum of all time intervals gives that the queue is expected to empty after $\sum_{n=0}^{\infty}((q-1)hw)(\lambda hw)^n$ seconds. This sum a geometric series, so the total time to an empty queue can be written in closed form as:

$$\frac{(q-1)hw}{1-\lambda hw} \tag{3.9}$$

The above summation is convergent if $\lambda hw < 1$; this condition is assumed to be true; otherwise, the arrival rate would be higher than the release rate.

The expected number of new vehicles is found by multiplying the length of this time interval by the arrival rate; giving an expectation of $\frac{(q-1)hw}{1-\lambda hw}\lambda$ new vehicles.

3.11 Vehicle delay during a signal phase

For the rest of this section, the expected total delay for a signal cycle is developed for a single movement using equations (3.6), (3.7), (3.8), and (3.9). The total expected delay for a signal cycle is the sum over all four movements. Because only one movement is considered at a time, notation can be simplified by letting $q = q^i(t_1)$ be the number of vehicle initially on movement i at time t_1.

To compute the total expected wait for on a green phase three different cases are considered. The first is if $t_2 - t_1 \geq \frac{(q-1)hw}{1-\lambda hw}$, where no vehicles are expected at the end of the phase. The second is if $\frac{(q-1)hw}{1-\lambda hw} > t_2 - t_1 \geq (q-1)hw$, where all initial vehicles are released but the queue is not expected to be empty at the end of the phase. The third is if $(q-1)hw > t_2 - t_1$, where not all vehicles in the initial queue are released.

Case 1: The queue is empty at the end of the phase.

If $t_2 - t_1 \geq \frac{(q-1)hw}{1-\lambda hw}$, then no vehicles are expected in the queue at the end of the phase.

If the queue is initially empty then the delay for the movement during this phase will be zero. Otherwise, the total expected delay from t_1 to t_2 is:

$$J_{green}(t_1, t_2) = \frac{q(q-1)}{2}hw + \frac{(q-1)hw}{2}\frac{(q-1)hw}{1-\lambda hw}. \tag{3.10}$$

The first term is the delay of the initial vehicles (3.6). The second term is the average wait of future vehicles (3.8) multiplied by the expected number of cars that will arrive before the queue is empty (3.9). Once all vehicles are released from the queue, new arrivals pass through the intersection with no wait.

Case 2: All initial vehicles are released but the queue does not empty.

If $\frac{(q-1)hw}{1-\lambda hw} > t_2 - t_1 \geq (q-1)hw$, then all initial vehicles are released. The number of vehicles that are expected to arrive is $\lambda(t_2 - t_1)$ and the number of vehicles released is $1 + Int\left(\frac{t_2 - t_1}{hw}\right)$. Therefore, $q + \lambda(t_2 - t_1) - \left[1 + Int\left(\frac{t_2 - t_1}{hw}\right)\right]$ vehicles are expected at the end of cycle. Thus

$$rel = 1 + Int\left(\frac{t_2 - t_1}{hw}\right) - q \tag{3.11}$$

of the new arrivals were released. This phase should be viewed as partly a phase that releases new vehicles and partly as a phase that does not. Vehicles that arrive during the first $rel \times hw$ seconds are released and vehicles that do not are not released.

The total expected delay from t_1 to t_2 is:

$$J_{green}(t_1, t_2) = \frac{q(q-1)}{2}hw + \frac{(q-1)hw}{2}rel + \frac{\lambda(t_2 - t_1 - rel \times hw)^2}{2}. \tag{3.12}$$

The first term is the delay of the initial vehicles (3.6). The second term is the average wait of future released vehicles (3.8) multiplied by the expected number of cars that will arrive and be released (3.11). The final term is the delay of the cars that are not released (3.7).

Case 3: Not all vehicles in the initial queue are released.

If $t_2 - t_1 < (q-1)hw$, then $q_{out} = 1 + Int\left(\frac{t_2 - t_1}{hw}\right)$ vehicles will be released. During this phase, an additional $\lambda(t_2 - t_1)$ vehicles are expected to arrive; none of them are released. Once again, at the end of the phase $q + \lambda(t_2 - t_1) - \left[1 + Int\left(\frac{t_2 - t_1}{hw}\right)\right]$ vehicles are expected.

The total expected delay from t_1 to t_2 is:

$$J_{green}(t_1, t_2) = \frac{q^i_{out}(q^i_{out} - 1)}{2}hw + (q - q^i_{out})(t_2 - t_1) + \frac{\lambda(t_2 - t_1)^2}{2}. \tag{3.13}$$

The first term is the delay of the released vehicles (3.6). The second term is the delay of initial vehicles not released and the third term is the expected delay of future arrivals (3.7).

The total expected delay of a signal cycle also includes the delay on the red movements. During a red phase, no vehicles are released and $\lambda(t_2 - t_1)$ vehicles are expected to arrive. Therefore, $q + \lambda(t_2 - t_1)$ vehicles are expected at the end of the red phase.

The total expected delay from t_1 to t_2 is thus:

$$J_{red}(t_1, t_2) = q(t_2 - t_1) + \frac{\lambda(t_2 - t_1)^2}{2}. \tag{3.14}$$

The first term is the delay of the initial vehicles and the second term is the expected delay of future arrivals (3.7).

3.12 Total vehicle delay during a signal cycle

When choosing a signal cycle length and split, it is important to not only reduce the delay of vehicles released during the current cycle, but also ensure that the vehicles released during future signal cycles have short delays. If the queue is too long at the end of a cycle, then the delay of vehicles in the queue will be large. Additionally, future arrivals will also face longer delays. When evaluating candidate signal cycles the delay of the vehicles left in the queue at the end of the signal cycle, t_3, needs to be considered. Since the phase lengths after time t_3 are not chosen, a lower bound for the future additional delay is computed. Ideal release times are considered in the computation of a lower bound. That is, the green signal durations are assumed to be t_{max} seconds and red signal durations are assumed to be t_{min} seconds. This is not possible because the green phase lengths on one movement must be the same as the red phase length on its conflict movements, but it does favor reduced queues without intense computation. This lower bound on the delay for movement i is denoted as $LB^i(t_3)$.

Equations (3.10), (3.12), (3.13), and (3.14) are combined to compute the total expected delay of a signal cycle beginning at t_1 with signal phase changes at t_2 and t_3 as:

$$J(t_1, t_2, t_3) = \sum_{i=1}^{4} \left(J^i_{u_{(t_1,t_2)}}(t_1, t_2) + J^i_{u_{(t_2,t_3)}}(t_2, t_3) + \sum_{k=1}^{q^i(t_1)} (t_1 - at^i_k) + LB^i(t_3) \right) \quad (3.15)$$

Once again u denotes the signal choice on the movement. At any given t_1, one set of parallel movements will be green and the other set will be red.

Note that equation (3.15) is always positive since each term is always greater than or equal to zero, and the queue on at least some of the movements is has positive probability of being non-empty.

Chapter 4 Ant Colony Optimization

Ant colony optimization (ACO) is a metaheuristic for solving computationally hard combinatorial optimization problems. The optimization problem is defined on (S, f, Ω) where S is a finite set of candidate solutions, f is the objective function to be minimized and Ω is a set of constraints. The goal is to find a globally optimal solution, $s^* \in S$, that minimizes f [6].

ACO is used to solve combinatorial NP-hard problems. It was first tested on the Traveling Salesman Problem. ACO is also used to solve Routing Problems, Quadratic Assignment Problems and Scheduling Problems [6].

4.1 Biological Ants

ACO is based on the methods of foraging ant colonies [8]. When searching for food, ants wander randomly around their nests. Once an ant finds food, it evaluates the food source and then returns to the nest. On the return trip, the ant deposits pheromone, a chemical used for communication. The amount of pheromone deposited is based on the quantity and quality of the food. The pheromone trail guides other ants as they continue to search for food. As more pheromone is deposited, the ants' paths become less random and are biased toward paths with higher pheromone concentration. As time progresses pheromone evaporates; so unless a path is traversed frequently, the pheromone on it disappears. Along with finding the best food sources, ants also find the shortest paths to food. Shorter paths are traversed faster, so pheromone is deposited on them more frequently. This leads to faster pheromone accumulation.

The phenomenon of ants using pheromone to communicate and discover optimal paths is observed in the Double Bridge Experiment [25]. A path between an ant's nest and food with a double bridge is laid out. In the first trial, the length of each bridge is equal, as shown in Figure 4-1. At first, ants move freely between the nest and food, choosing either path randomly. As time progresses, due to random fluctuations, one of the paths gains a higher pheromone concentration; this larger amount of pheromone attracts more ants. The increased number of ants deposits more pheromone on this path. A positive feedback loop is created and the number of ants that choose this path increases until all the ants are using it.

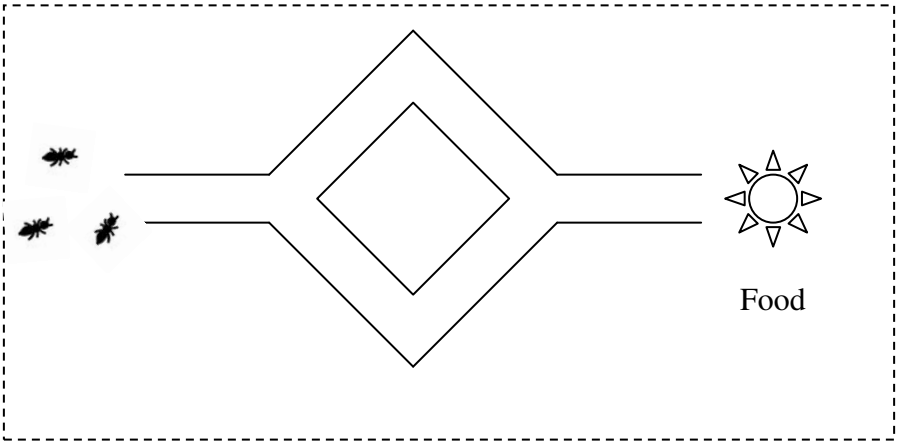

Figure 4-1. Double bridge with equal lengths

In the second trial, one of the paths is made twice as long as the other path, as shown in Figure 4-2. Once again, ants start by randomly using both bridges, but soon more pheromone is concentrated on the shorter path. Eventually, the higher pheromone level causes all ants to travel along the shorter path.

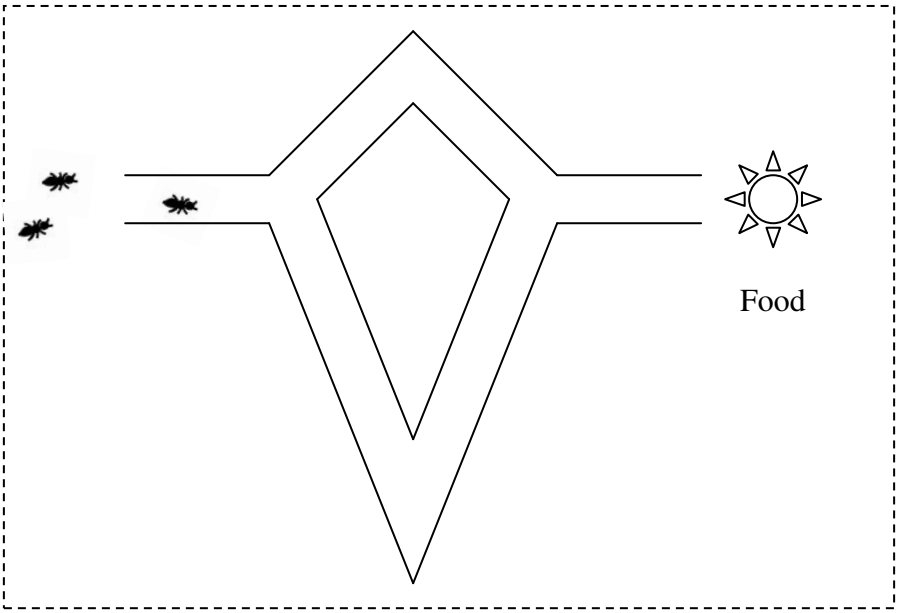

Figure 4-2. Double bridge with unequal lengths

4.2 Ant Colony Optimization framework

The movements of real ants are modeled by artificial ants in ant colony optimization. In ACO algorithms, artificial ants probabilistically search a graph, with the guidance of the pheromone, to create candidate solutions. Candidate solutions are then evaluated and used for pheromone updates. Many different versions of the ACO have been developed, but they all follow the same idea of solution construction guided by pheromone levels. The framework for ACO algorithms is as follows [7]:

1) Initialize Pheromone Values. The pheromone values on each path are set to the same constant value.

2) Solution Construction. Each ant begins on a start node and constructively builds a solution based on the pheromone values. A solution is an ordered set of nodes. Ants move from node i to node j with probability:

$$p_{ij} = \begin{cases} \dfrac{\tau_{ij}^{\alpha}\ \eta_{ij}^{\beta}}{\sum_{l \in N_i} \tau_{il}^{\alpha}\ \eta_{il}^{\beta}} & \text{if } j \in N_i \\ 0 & \text{if } j \notin N_i \end{cases} \tag{4.1}$$

where N_i is the neighborhood of node i. The neighborhood of node i is the set of all nodes that an ant can move to when at node i. The pheromone value between node i and j is τ_{ij} and η_{ij} is a heuristic value. The values of α and β are nonnegative; and they weight the relative importance of the pheromone and heuristic values respectively.

3) Update Pheromone. The pheromone update is the key difference between most ACO algorithms; but the general framework still holds. First pheromone is evaporated by the rule:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij}, \forall (i,j) \in A, \tag{4.2}$$

where $\rho \in (0,1]$ is the evaporation coefficient.

Then pheromone on some of the paths is increased by:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}. \tag{4.3}$$

Where the pheromone update, $\Delta\tau_{ij}$, is algorithm specific.

4) The solution construction and pheromone update are repeated until the stop condition is met.

Because each algorithm updates pheromone differently, different values are used for their parameters. Each parameter is chosen specifically for the application. But, in common applications, like the Traveling Salesman, formulas have been developed to give the range of parameter values [6].

4.3 Specific Ant Colony algorithms

In this section, the Ant System, Elitist Ant System and Rank Based Ant System algorithms [6] are discussed. Each algorithm uses m ants to construct solutions; and the initial pheromone deposit and solution construction steps are the same. Their differences lie in the pheromone update step.

Each artificial ant begins at a start node and constructs a solution. The solution constructed by ant k is denoted by T^k. Each solution T^k is given a cost, denoted by C^k, which is related to the objective function being minimized. Ant system is the simplest method and the easiest to implement, but it is usually not sufficient in applications. Other ACO algorithms are modifications of the ant system algorithm. Specific algorithms are chosen based on the problem of interest.

Ant System:

In this algorithm, all ants are considered equally. After each ant has constructed a solution, the pheromone levels on all arcs are evaporated with the same rate, as shown in equation (4.2). Then each ant adds pheromone to each link it took by:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \qquad \forall (i,j) \in A \tag{4.4}$$

where m is the total number of ants, $\Delta\tau_{ij}^{k}$ is the pheromone deposited by ant k on the arc between i and j, is defined by:

$$\Delta\tau_{ij}^{k} = \begin{cases} 1/C^k & \text{if arc } (i,j) \text{ belongs to } T^k \\ 0 & \text{otherwise} \end{cases}. \tag{4.5}$$

Elitist Ant System:

In this algorithm, extra weight is given to the best-so-far solution, denoted as T^{bs}. As in the Ant System algorithm, pheromone is first evaporated as in equation (4.2), then ants deposit pheromone by

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + e\Delta\tau_{ij}^{bs}, \qquad \forall (i,j) \in A \tag{4.6}$$

where $\Delta\tau_{ij}^{k}$ is defined the same as in the Ant System algorithm, e is the weight for the best-so-far path and $\Delta\tau_{ij}^{bs}$ is defined by:

$$\Delta\tau_{ij}^{bs} = \begin{cases} 1/C^{bs} & \text{if arc } (i,j) \text{ belongs to } T^{bs} \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

where C^{bs} is the cost related to the best-so-far solution.

Rank-Based Ant System:

In this algorithm, the ant's solutions are sorted in order of increasing cost before the pheromone is deposited. Only the $(w-1)$ best-ranked ants and the best-so-far ant are allowed to deposit pheromone. The best-so-far solution is weighted by w, the r^{th} best ant is weighted by $\max\{w-r, 0\}$. Thus the pheromone update rule is:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{r=1}^{w-1} (w-r)\Delta\tau_{ij}^{r} + w\Delta\tau_{ij}^{bs}, \qquad \forall (i,j) \in A \tag{4.8}$$

where $\Delta\tau_{ij}^{r}$ and $\Delta\tau_{ij}^{bs}$ are as defined above. The pheromone evaporation stage is performed before the update, as in the other methods, but less pheromone is generally evaporated on each step. The rank-based update biases away from bad solutions, allowing for more conservative evaporation.

4.4 Ant Colony applications

Ant colony algorithms have successfully been applied to very difficult combinatorial optimization problems. They were first applied to the Traveling Salesman Problem (TSP) by Marco Dorigo [26]. Some other successful applications of note are vehicle routing problems [27] and quadratic assignment problems.

The Traveling Salesman Problem is based on a salesman who is given a set of customer cities and a starting location. The salesman wishes to find the shortest path that visits each city exactly once. Customer cities and the paths between them are represented as a weighted graph. Each city is a vertex on the graph and each path is an edge, weighted by its length. The TSP has practical applications in printed circuit boards and the positioning of X-ray devices [6].

When applying ACO to the TSP, ants begin at the starting point and randomly traverse the graph. Once an ant has visited every city, it updates the pheromone on edges it traversed. Ants deposit pheromone inversely to the total length of their trip. The heuristic information is usually inversely proportional to the length of an edge. A straight-forward choice is weighting the path between node i and j by $\eta_{ij} = 1/\mathrm{dist}(i, j)$. Ant colony algorithms have been shown to find optimal solutions to the TSP in fewer iterations than other naturally inspired algorithms, such as genetic algorithms and simulated annealing [26].

The vehicle routing problem involves resource allocation. A set of customers must receive deliveries from a central depot with a given number of delivery vehicles. Each delivery vehicle has a fixed capacity and each customer has a nonnegative demand. A vehicle cannot serve more customer demand that its capacity allows. The goal is to

determine the most efficient route for each vehicle so that each customer is visited exactly once and delivery vehicles do not exceed their capacity [28].

This problem is also represented as a weighted graph. The nodes are the central depot, denoted node 0, and each of the customers. The arcs represent the paths that connect the customers to each other and the central depot. Once again, each arc is weighted by the distance between the nodes it connects. To apply ACO to this problem each ant begins at the central depot and constructs a vehicle's route until it reaches the vehicle's capacity. Then the ant returns to the central depot and begins to construct a route for another vehicle. It continues to construct routes until each customer is served. Then it updates the pheromone on each node it took and starts over. This problem is harder than TSP because it involves solving TSP for each vehicle. Heuristic information for this problem is similar to the TSP but a customer's distance from the central distance is also considered. A typical heuristic weight is $\eta_{ij} = d_{i0} + d_{0j} - ad_{ij} + b|d_{i0} - d_{0j}|$, where $a \geq 1$ and $b > 0$ are parameters. The quantity $d_{i0} + d_{0j} - d_{ij}$ is the distance saved by going straight from node i to node j instead of visiting the central depot first. The extra factor of $(a-1)d_{ij}$ discourages moving to nodes that are far away and the final term keeps the distance from the central depot from changing rapidly [27].

Chapter 5 Ant Colony and Traffic Optimization

This chapter begins with the motivation behind using ACO for traffic signal control. Then, the second section describes how the ACO is implemented for traffic signal control.

5.1 Motivation

The traffic signal problem is addressed very naturally as a combinatorial optimization problem. As traffic networks grow, the complexity of the finding an optimal solution becomes much more difficult. Total enumeration of all solutions becomes intractable very quickly, so advanced methods must be used [9]. ACO algorithms have successfully been applied to many computationally complex combinatorial problems, making ACO a good choice for solving the traffic signal problem.

The ACO ability to incorporate heuristic information about traffic networks makes it more efficient. For example, in the isolated traffic signal problem the maximum queue length currently at the signal is accounted for. On more complicated traffic topologies, other heuristic measures can be incorporated, such as distances between signals.

Ant colony optimization has successfully been applied to other traffic related problems, such as the vehicle routing problem (VRP), with positive results. Although the VRP has a different setup and objectives, similar heuristics and objective functions are used in both cases.

As will be discussed later in this section, the ACO can be used to optimize rolling horizon control. Rolling horizon control has successfully been used in traffic signal control [13]. Some of the advantages of this approach were discussed in Chapter Two.

Additionally, ACO requires very few restrictions on the cost function. For example, many optimization techniques rely on computing a gradient. This requires the existence of a gradient and can be computationally expensive. ACO algorithms are not dependant on the form of objective function; if the objective function is changed the algorithm works the same. This allows the heuristic information, intersection topology, and vehicle arrival rates to be easily changed. Thus, the ACO robustly conforms to new situations.

5.2 Ant Colony implementation

The rolling horizon method is used to implement the optimal signal cycles. The length of the horizon is variable and set equal to the length of the signal cycle chosen. The length of a full signal cycle is used for the horizon; giving all movements equal weight in the decision process. Once the optimal policy is found, it is implemented for one phase. Then, the beginning of the horizon is advanced to the next signal switch time and the optimization process is repeated. Figure 5-1 shows an example of signal choices and time advances.

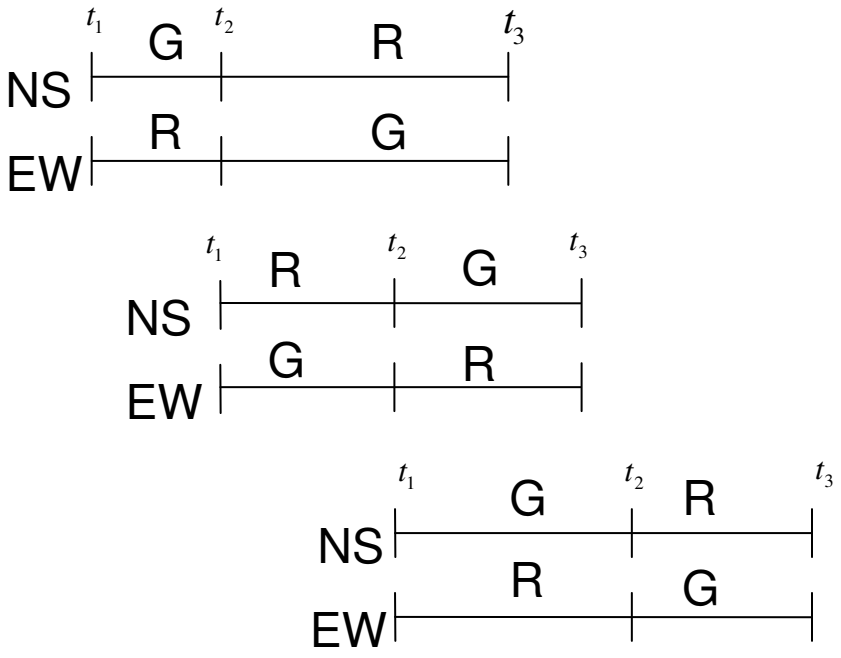

Figure 5-1. Example of Rolling Horizon Control

Ant colony optimization uses artificial ants to evaluate candidate solution and find the optimal signal cycle switch times. The length of a green signal in a candidate solution is bounded between the predetermined minimum green time t_{min} and maximum green time t_{max}. Additionally, to make the set of candidate solutions finite, time is discretized into one second time intervals. With these constraints, a graph is constructed for the ants to traverse, as shown in Figure 5-2. When an ant is at time t the set of admissible nodes that it can move to is $\{t + t_{min}, t + t_{min} + 1, t + t_{min} + 2, \ldots, t + t_{max}\}$. All ants start at the current time, then they move right to a new node, representing the next signal switch time. From there they move to the right again to another node, representing the next signal switch time. This creates a full signal cycle. At a given time, t_1, the set of candidate solutions is all the possible admissible combinations of the next two switch times, t_2 and t_3.

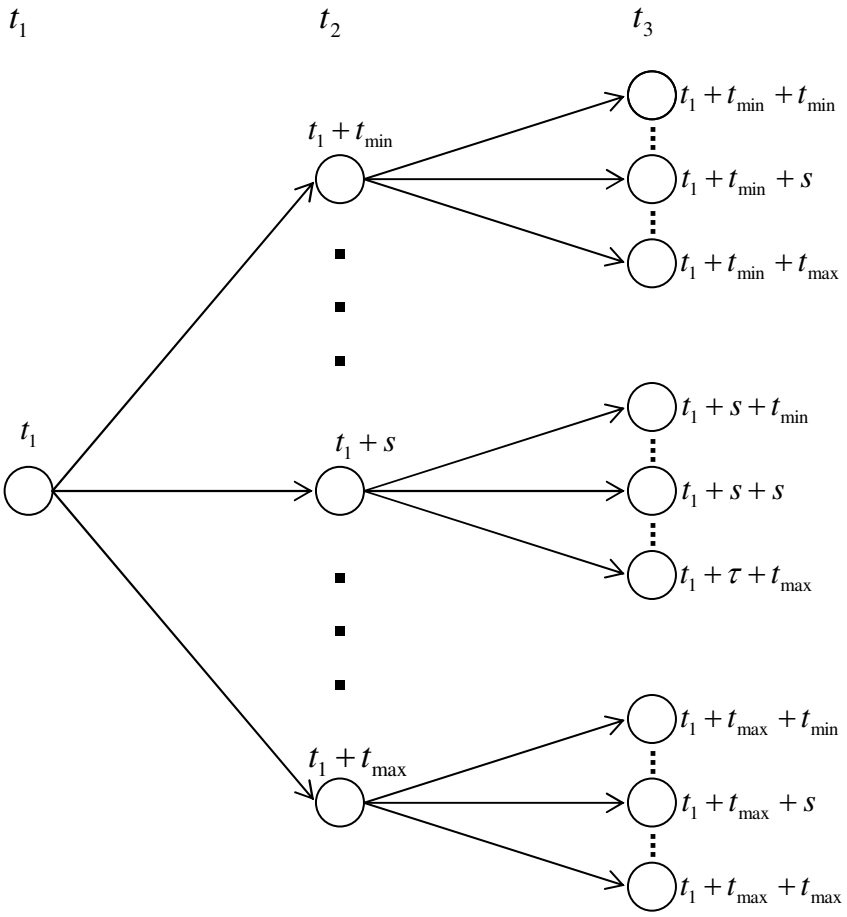

Figure 5-2. Graph that ants traverse

After an ant creates a signal cycle, t_2 and t_3, the expected total delay, $J(t_1, t_2, t_3)$, of the ant's solution is computed, using equation (3.15). But $J(t_1, t_2, t_3)$ is not quite the cost function that needs to be minimized. Shorter time intervals tend to have smaller total expected delays because fewer vehicles enter the intersection during the cycle. Therefore, the expected delay of fewer vehicles is being summed. Short cycle lengths are suboptimal when they create long queue lengths. To avoid long queues, the total expected delay is divided by the length of the cycle multiplied by the traffic volume plus the number initial vehicles. This gives the expected average delay per vehicle. This value, not the total

expected delay, is minimized. Thus the cost of the solution associated with the pheromone update in equation (4.5) is:

$$C^k = \frac{J^k(t_1, t_2, t_3)}{4\lambda(t_3 - t_1) + \sum_{i=1}^{4} q^i(t_1)}. \tag{5.1}$$

This pheromone value is added to the edges which ant k traversed to create its solution.

Shown in Figure 5-3 is the flow chart for the Ant Colony code.

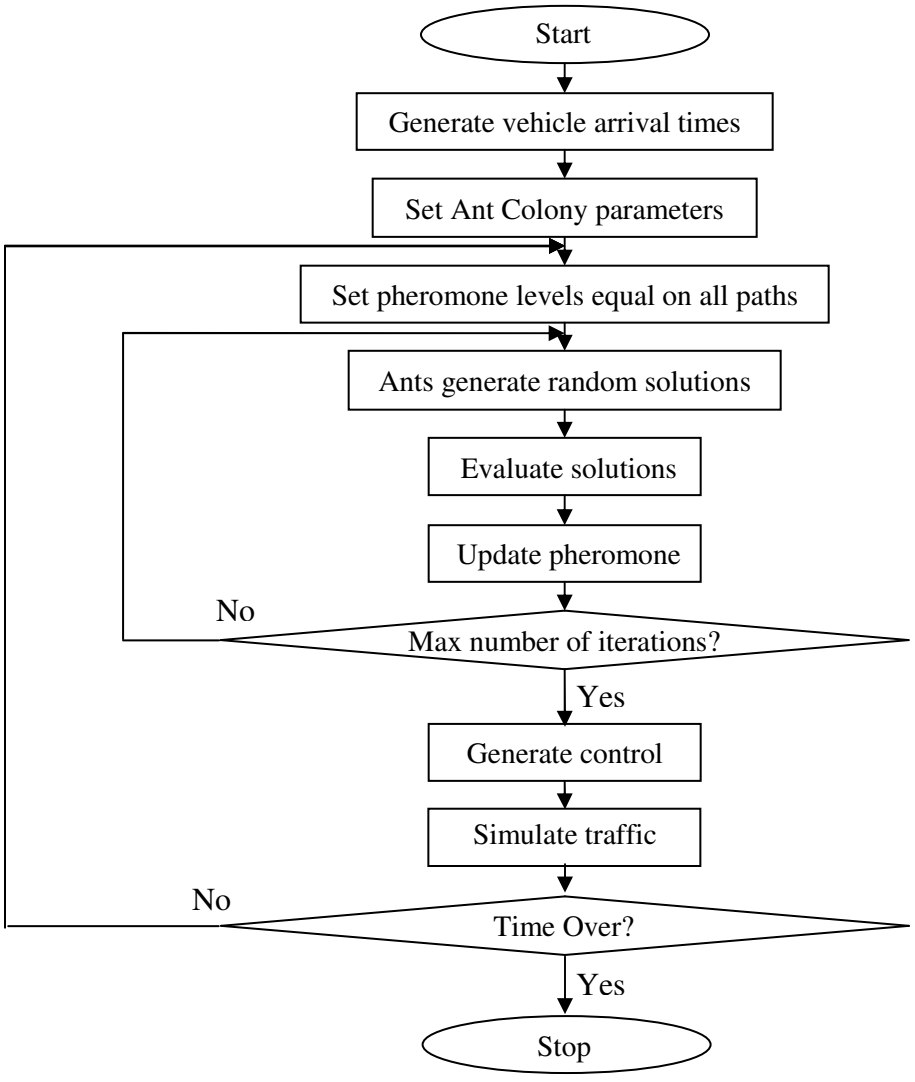

Figure 5-3. Computational Flow Chart

An advantage of the ACO is its ability to incorporate heuristic information about the solution space being searched [6]. In the traffic signal problem, releasing all vehicles in the queue usually results in smaller waiting times. So the green phase length should be set accordingly. For small queues, releasing the current queue and then switching the signal is optimal. For longer queues, additional time is optimal because, with high probability, additional vehicles will arrive before all vehicles are released. In either case, the optimal solution is near the time when all vehicles are released. This time is $t_1 + (q^g(t_1) - 1)hw$, where $q^g(t_1)$ is the length of the largest queue on the green movements at time t_1. To bias the search towards switch times near this time, the pheromone levels are weighted by the heuristic value of

$$\eta_{t_1 t_2} = Exp[-\frac{|(q^g(t_1) - 1)hw - (t_2 - t_1)|}{c}], \tag{5.2}$$

where c is a positive constant. The value of c is chosen experimentally. The exponential function is used because it has a sharp peak at its maximum. Any function with a local maximum can be used and choice of function is determined experimentally. A graph of η with $c = 5$, $q^g(t_1) = 7$ and $hw = 2$ is shown in Figure 5-4.

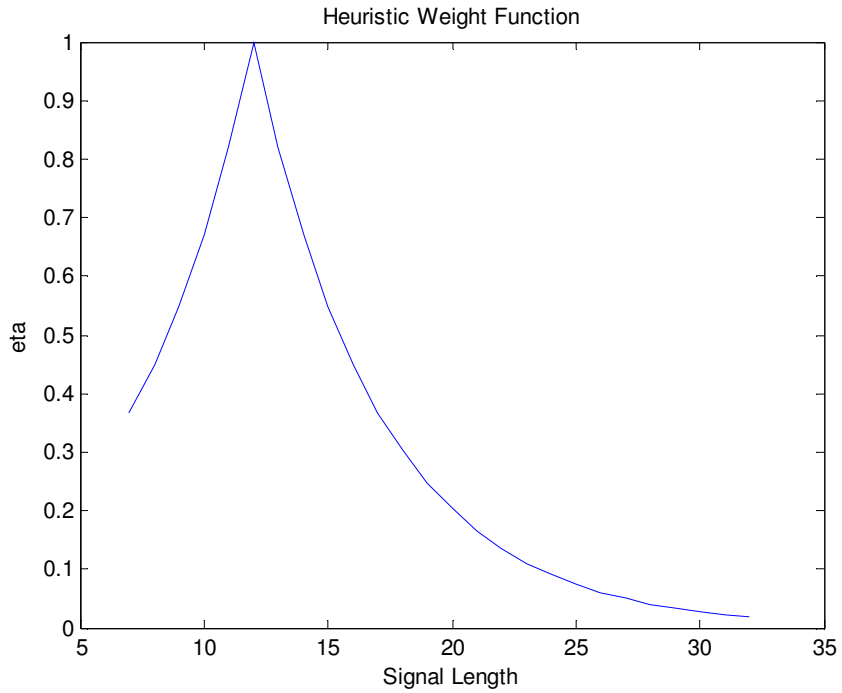

Figure 5-4. η with seven vehicles in queue

One problem with the ACO is its tendency to accumulate pheromone on near optimal solutions [6]. At initialization, all paths are chosen with equal probability. If a near optimal solution is randomly chosen more than any other path at initialization, then its paths will have the most pheromone. The positive feedback of the ant colony algorithm can cause pheromone to accumulate rapidly on this near optimal solution. As a result, the optimal path may not be found. When using an ACO algorithm with the best-so-far ant this stagnation became especially apparent. Simulations in this research demonstrate this stagnation, see section 6.1. To avoid stagnation, a search of the solutions near the best-so-far solution can be added. This local search can be performed in many different ways and is dependant on the problem being optimized.

In the traffic signal problem this is accomplished by replacing every n^{th} iteration of the random solution search with a local search. In this local search the search space is

replaced with a neighborhood of size T of the best-so-far solution. On a normal iteration, the possible choices next switch times for an ant at time t is the set $\{t + t_{min}, t + t_{min} + 1, t + t_{min} + 2, \ldots, t + t_{max}\}$. In a local search, the search space is restricted to $\{t + t_{bt} - T, t + t_{bt} - T + 1, \ldots, t + t_{bt} + T\}$ intersected with the set of allowable signal settings. Where t_{bt} is the best so far signal switch time. If a better solution is found, it replaces the old best-so-far solution. Pheromone evaporation and update is not performed on the local search iterations. This process leads to less stagnation and faster convergence.

5.3 Fully actuated control

ACO algorithms are compared to a traditioned fully actuated control strategy. In is sometimes called the vehicle-interval method [9]. In this strategy the minimum and maximum green times, as well as an extension time are given.

First, the minimum green time is assigned to a set of movements. If another vehicle arrives on a green movement during the minimum green time, then the controller extends the green signal by the extension time. The controller continues to extend the green signal if new vehicles arrive until the maximum green time is reached. If no vehicle arrives on the movements during an extension period, then the controller checks for a vehicle in the queue of the red movements. If there is a vehicle, this movement is given the minimum green time; otherwise, the signal remains the same.

Chapter 6 Simulation Results

In this chapter, the results of different ant colony algorithms and parameter choices are presented and compared. First, converge rates of pheromone concentration to the optimal solution using different algorithms and parameters are examined. Then, the average vehicle delay of the Ant Colony control is compared with a traditional fully actuated control algorithm. Table 6-1 shows the traffic parameters in seconds used in simulations. The parameters were chosen consistent with literature [5].

Table 6-1. Traffic Simulation Parameters

Parameter	Value
Minimum green time (s)	5
Maximum green time (s)	30
All red time (s)	2
Minimum headway (s)	2
Extension time (s)	1

6.1 Convergence of pheromone levels to best solution

An accurate way to determine how long an ACO algorithm typically takes to find the optimal solution is required to evaluate it. Because the algorithm uses a probabilistic search, the correct solution can be found on the first iteration or never. Fortunately, there are many methods to ensure the possibility of never finding the optimal solution is ruled out [6]. In order to estimate algorithm convergence rates, algorithms are tested on examples where the optimal solution has a priori been calculated. Then the pheromone concentration on the optimal path is recorded. Eventually the pheromone becomes so concentrated on a path that running further iterations does not change the pheromone levels.

To see the qualitative behavior of the pheromone convergence, a simple case is considered. Each movement of the intersection initially has zero vehicles in their queue. In this case, it is optimal to switch the signal after the minimum green time. The figures that follow show the percent of the total pheromone lying on the optimal signal setting.

For each choice of ant colony parameters, one hundred trials are run. In each trial, every edge of the ants' graph begins with equal pheromone. During an iteration of the trial, every ant constructs a solution and accordingly makes a pheromone update. The pheromone level on the optimal phase is plotted after every iteration; showing how the pheromone concentrates on the correct phase. The pheromone convergence from each trial is plotted on the same graph. Then, the average over all trials is compared for different numbers of ants. In the cases where many of the trials choose the wrong signal switch times, the average gives a picture of what percentage of trials found the optimal solution.

The number of ants used in ACO is an important implementation issue. As little as one ant could be used, but this does not take full advantage of the algorithm. When more ants are used, more exploration is done during each iteration. As a result, more pheromone is released per iteration, decreasing the chance of biasing towards poor solutions. But, increasing the number of ants increases the computational work done per iteration. Additionally, the large amount of pheromone deposited does not allow significant bias towards one path. As a result, the pheromone levels change slower and it is difficult to tell when the optimal solution is found.

In the following simulations, the size of the local neighborhood is $T = 4$. Meaning, on a local search step, the solution search space is restricted to paths that are

distance four of less from the best so far solution. The local neighborhood of four is chosen because the typical wrong solution without the local search is within four of the optimal solution. The local search is performed every third iteration. ACO is pretty robust to how often the local search is performed, but being performed more often is preferable [6]. In formula (5.2), the heuristic weight function, the constant is $c = 5$. In formula (4.1), the exponents α and β are both 1, more explanation on the choices of α, β, and c are given in Section 6.9. In the elitist ant update, formula (4.6), the elitist weight is $e = 10$. The choice of the elitist weight should be on the same order of magnitude as the rest of the ants [6]. In the rank based system the top ten ants are used to update. The pheromone evaporation coefficients of $\rho = .4$ and $\rho = .2$ are compared. The number of ants used and iterations taken is shown in the graphs. The vehicle arrival rate is 800 vehicles per hour per movement.

6.2 Ant System

First, the original ant system algorithm is examined with ten, twenty-five and fifty ants. As seen in the Figure 6.1-8, when only ten ants are used, the pheromone levels tend to concentrate faster. This accumulation is even more apparent if $\rho = .4$, when more pheromone is evaporated on each iteration, causing the pheromone levels converge faster. Unfortunately, most trials do not converge to the optimal solution.

As more ants are used the optimal path is chosen more often, but too many iterations are necessary and the number of trials that chose the optimal path is not sufficient. As a result more advanced algorithms must be used.

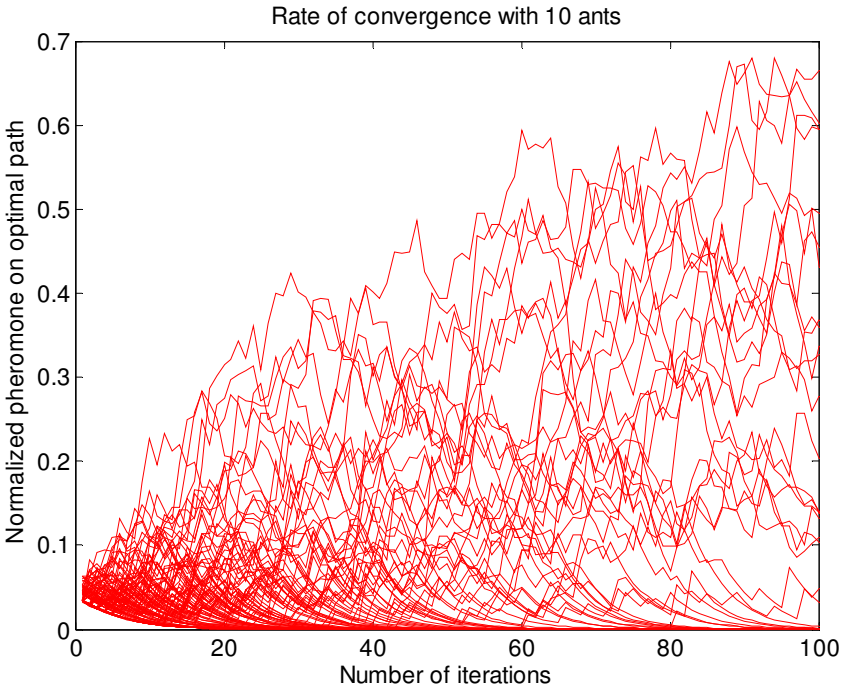

Figure 6-1. Ant System rate of convergence with 10 ants, $\rho = .2$

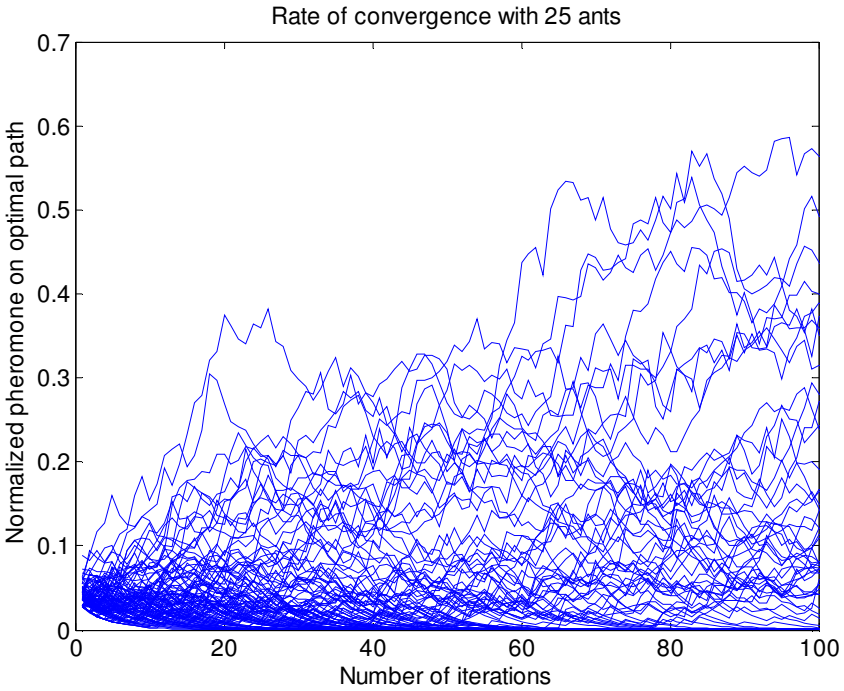

Figure 6-2. Ant System rate of convergence with 25 ants, $\rho = .2$

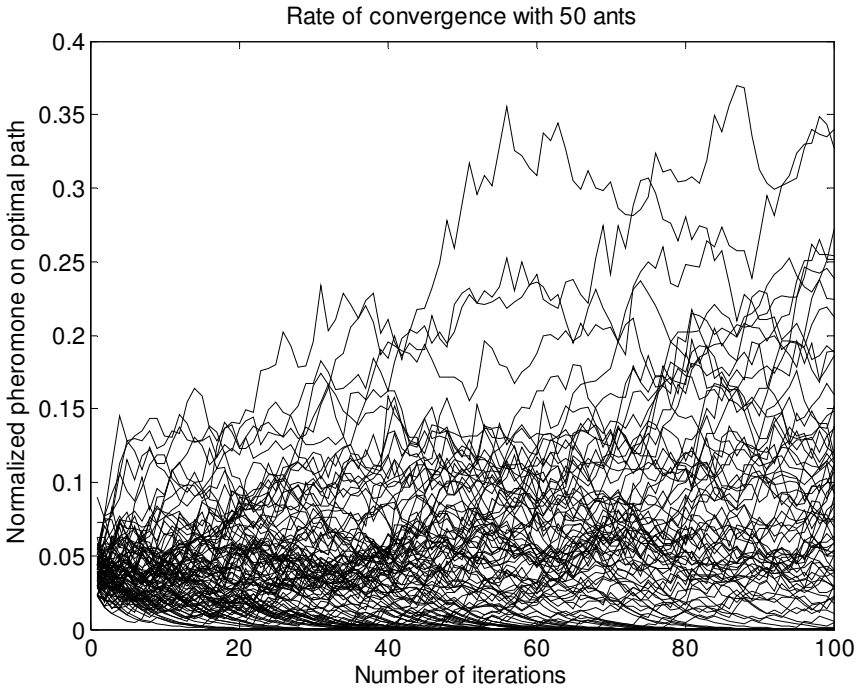

Figure 6-3. Ant System rate of convergence with 50 ants, $\rho = .2$

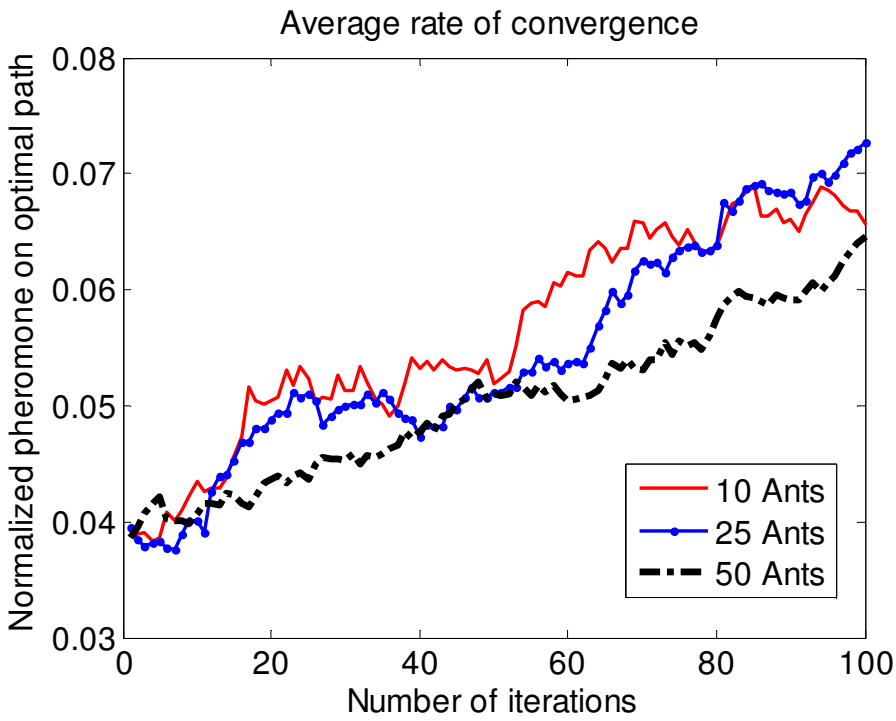

Figure 6-4. Ant System average rate of convergence with $\rho = .2$

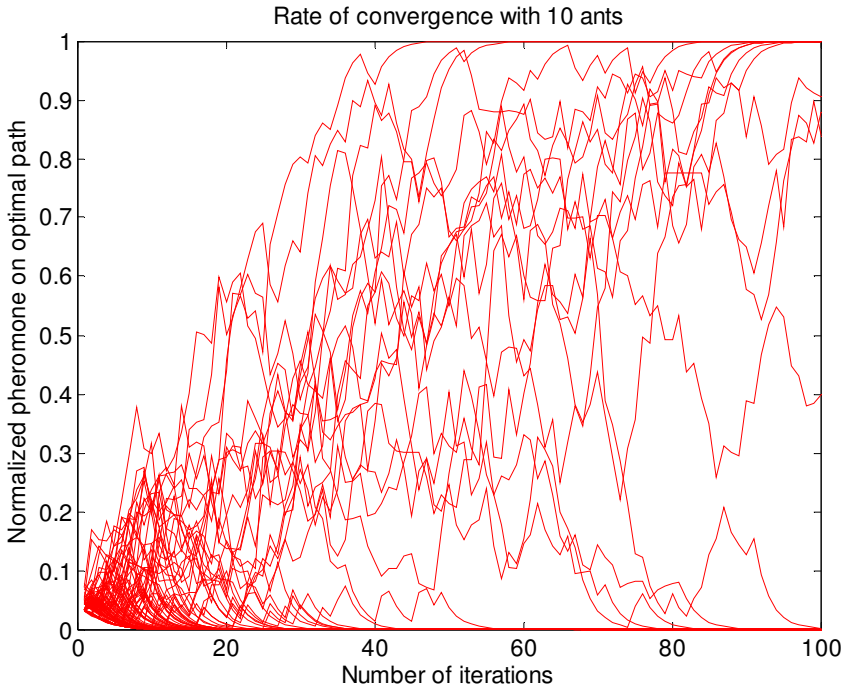

Figure 6-5. Ant System rate of convergence with 10 ants, $\rho = .4$

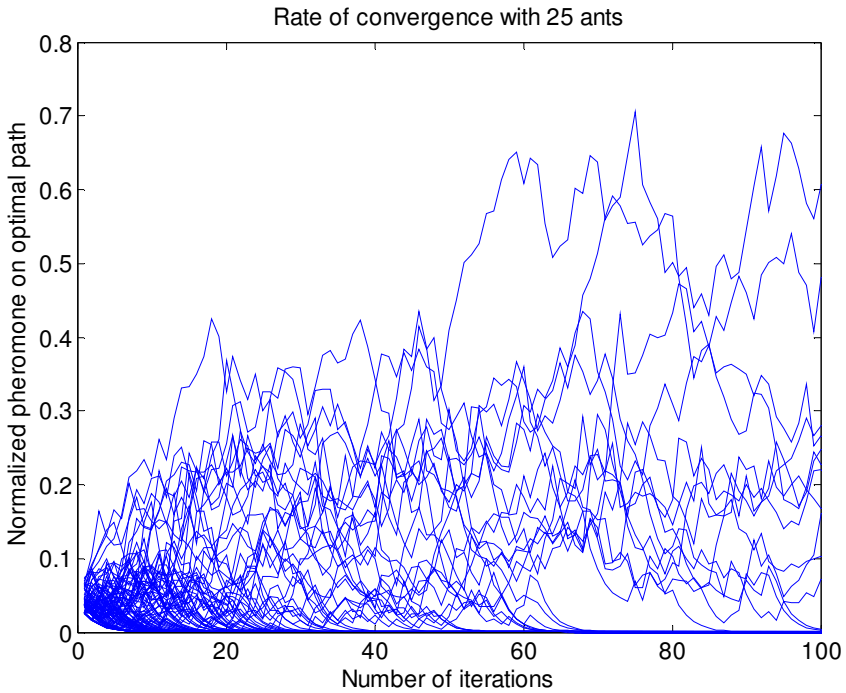

Figure 6-6. Ant System rate of convergence with 25 ants, $\rho = .4$

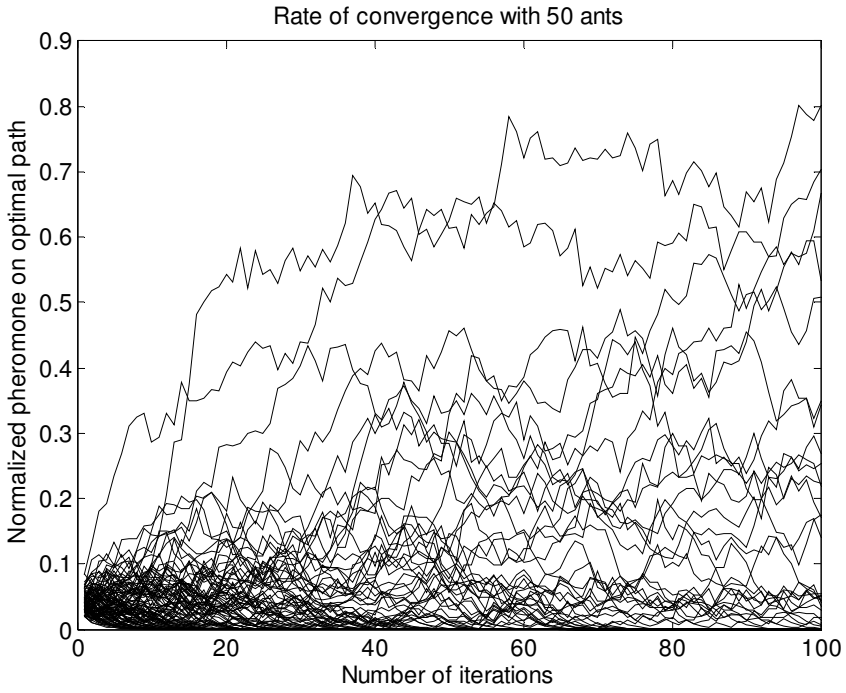

Figure 6-7. Ant System rate of convergence with 50 ants, $\rho = .4$

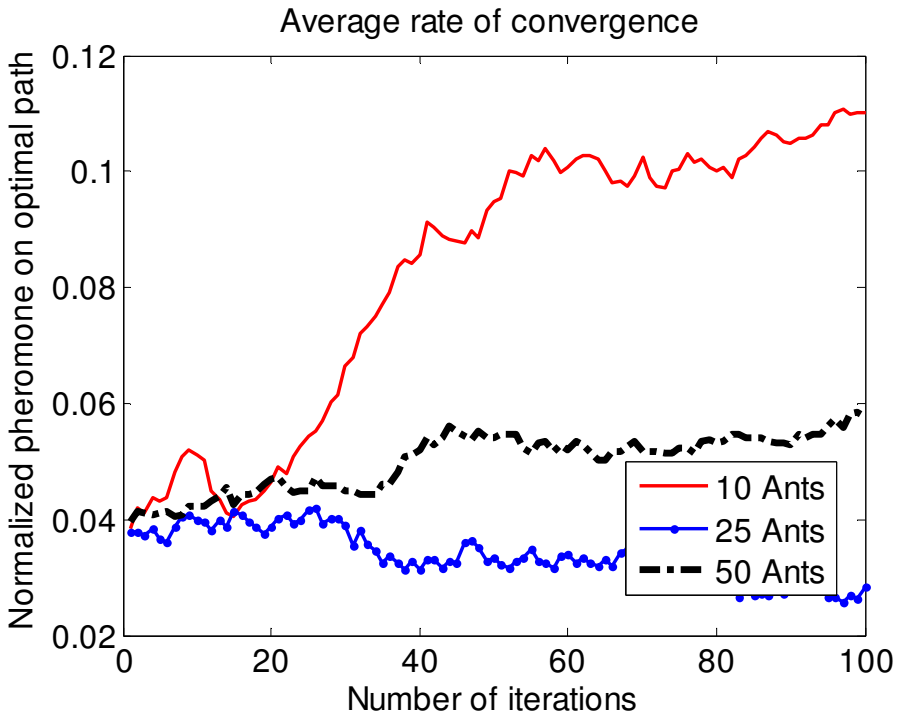

Figure 6-8. Ant System average rate of convergence with $\rho = .4$

6.3 Local search

As described in chapter five, adding a local search can improve convergence. Unfortunately, as shown in the figures below, when it is used alone it does not help. The following graphs behave very similarly to the graphs in Section 6.2. The pheromone addition on the local search step is not sufficient to create a significant bias towards an optimal solution.

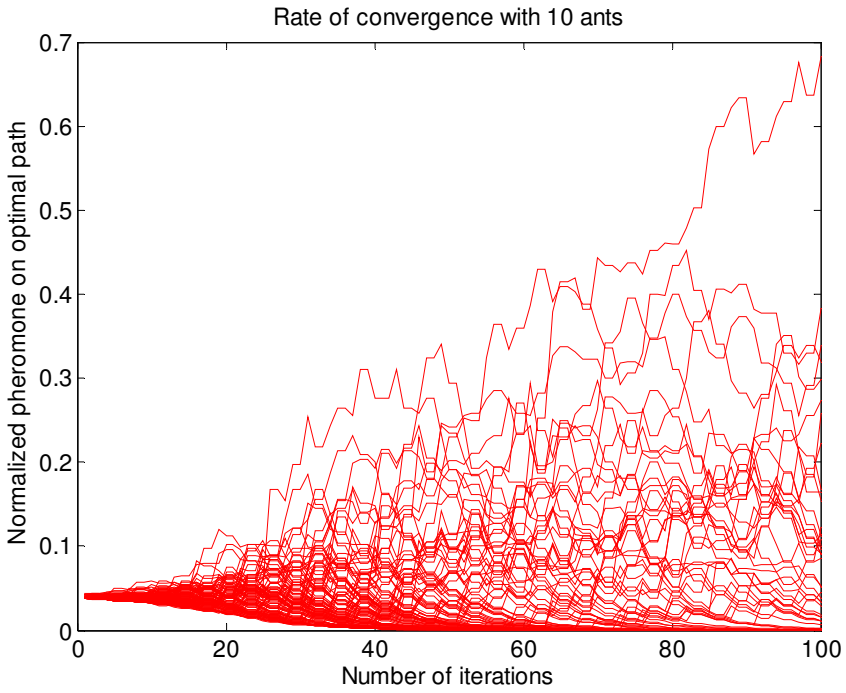

Figure 6-9. Ant System with local search rate of convergence with 10 ants, $\rho = .2$

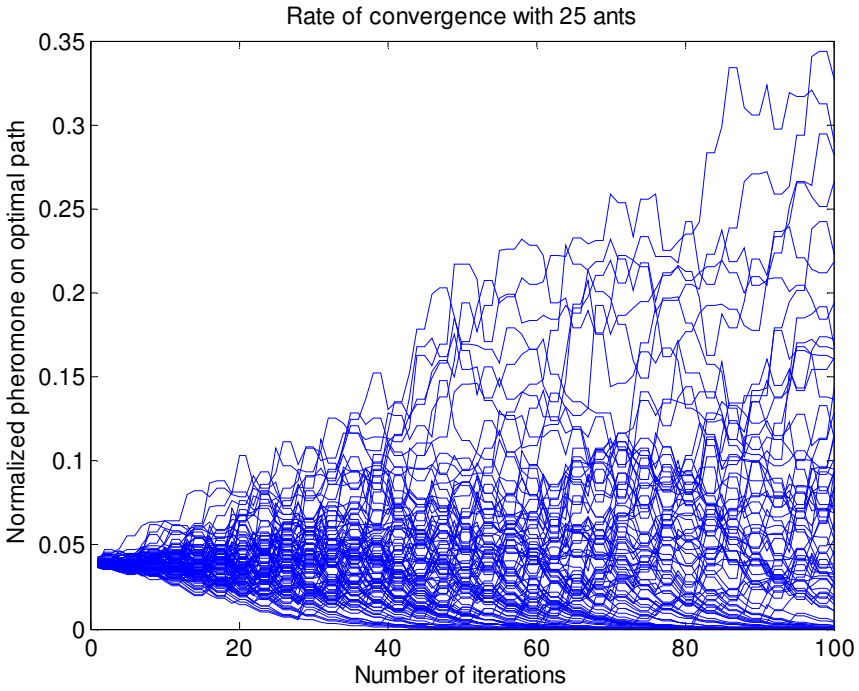

Figure 6-10. Ant System with local search rate of convergence with 25 ants, $\rho = .2$

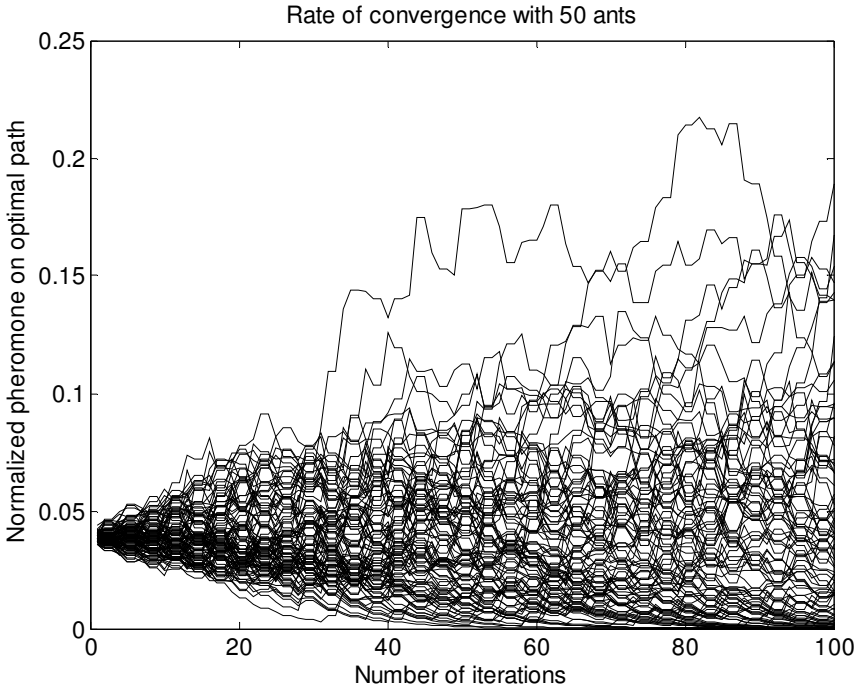

Figure 6-11. Ant System with local search rate of convergence with 50 ants, $\rho = .2$

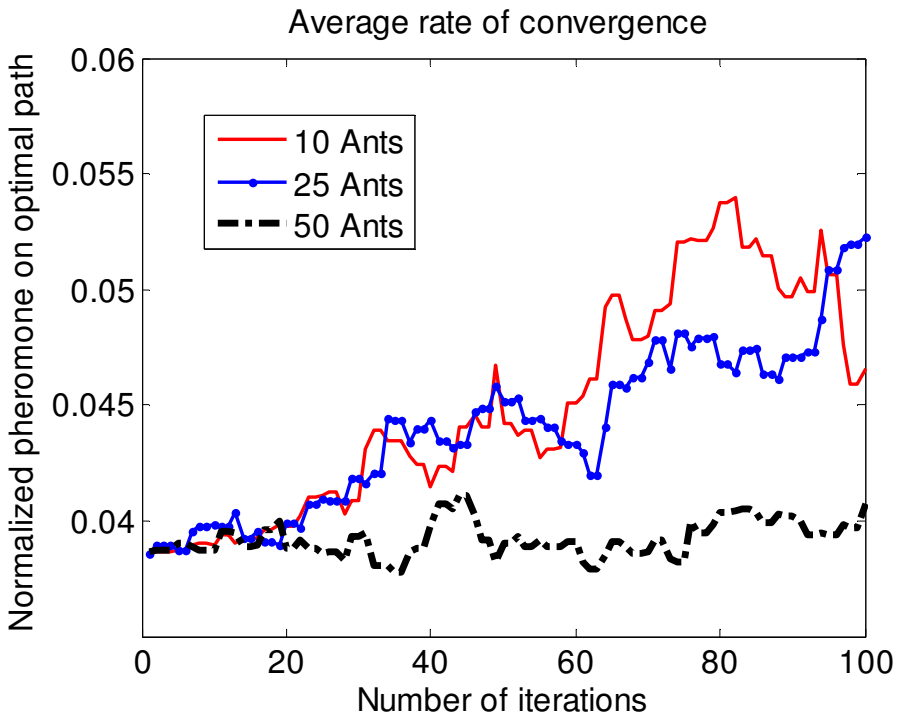

Figure 6-12. Ant System with local search average rate of convergence with $\rho = .2$

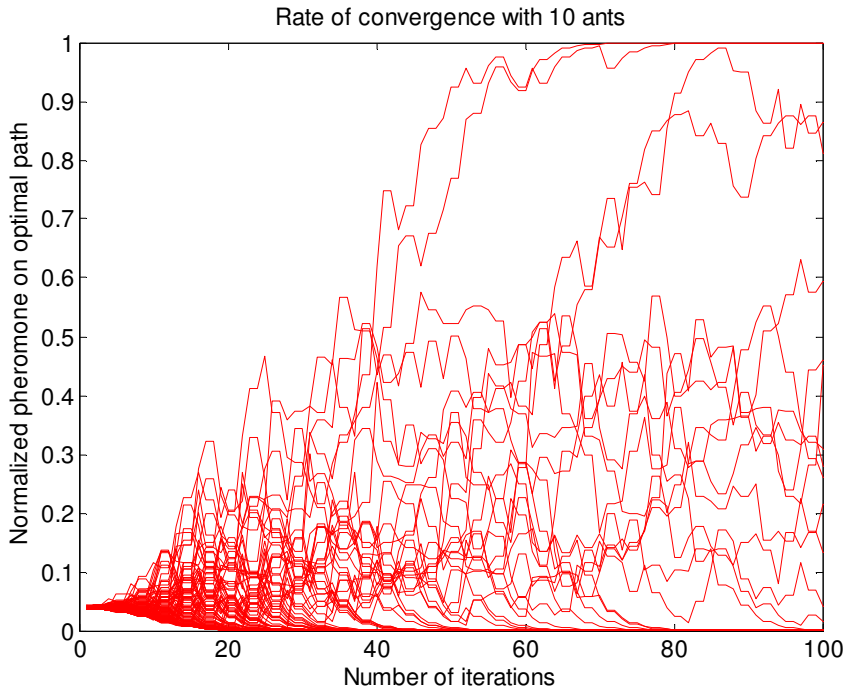

Figure 6-13. Ant System with local search rate of convergence with 10 ants, $\rho = .4$

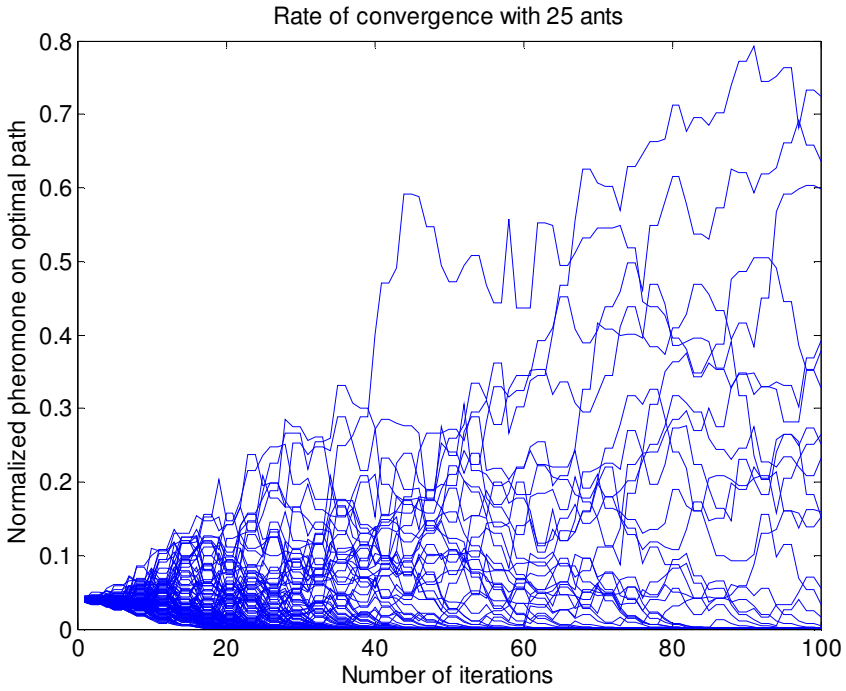

Figure 6-14. Ant System with local search rate of convergence with 25 ants, $\rho = .4$

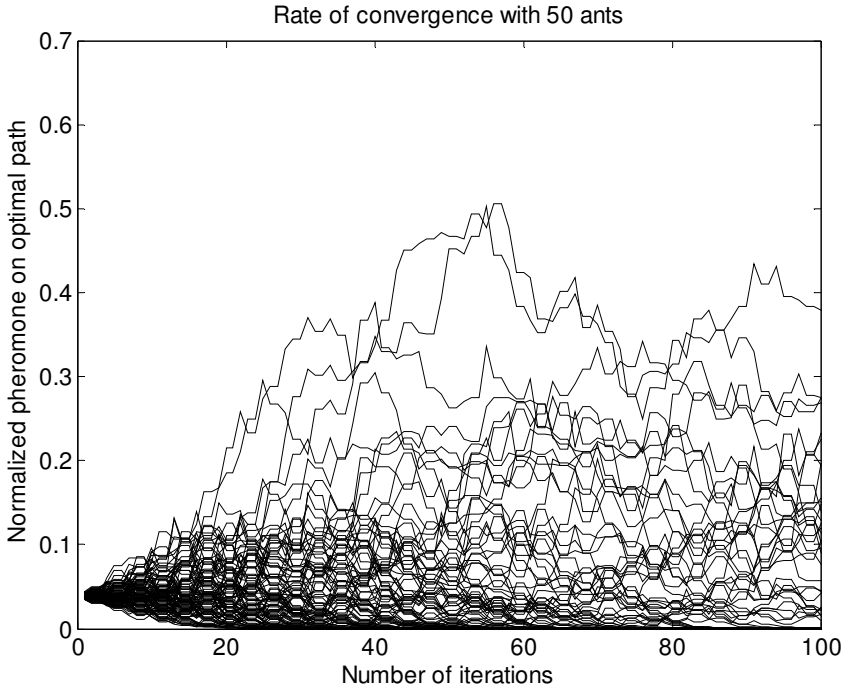

Figure 6-15. Ant System with local search rate of convergence with 50 ants, $\rho = .4$

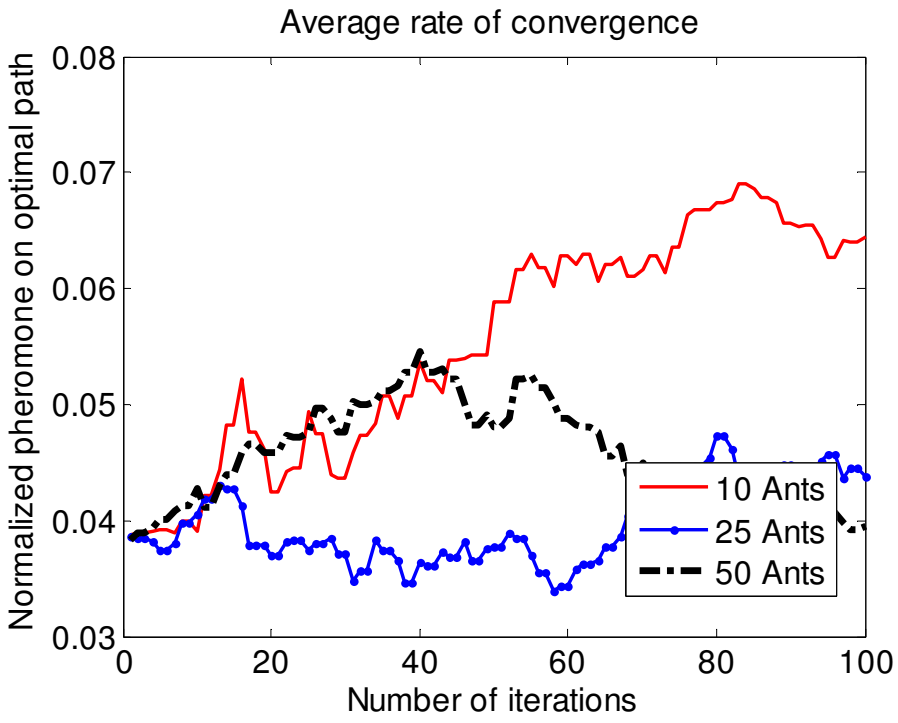

Figure 6-16. Ant System with local search average rate of convergence with $\rho = .4$

6.4 Elitist Ant System

To improve the speed of convergence, the best-so-far path is weighted heavier in the Elitist Ant System. If bad solutions are chosen at the beginning too often, they will get too much pheromone and cause pheromone to concentrate on the wrong path. Alternatively, if the optimal solution is chosen, then convergence will be faster. The speed of convergence is very dependant on pheromone evaporation. Using more ants reduces the effects the elitist update. The pheromone concentrates slower on bad paths, but also does not indicate as fast when a good solution is found. Additionally, the number of ants required to offset bad updates from the elitist ant is too large.

When $\rho = .2$, the average over all iterations looks similar for the different number of ants. In the cases with fewer ants, there is a smaller chance of choosing the optimal solution, but if the correct solution is found convergence is faster. On the other hand, when more ants are used the pheromone concentrates slower, even after the optimal solution is found.

When $\rho = .4$ and only ten ants are used, the optimal solution is not found early very often and evaporation occurs rapidly. As a result, the optimal solution is not found very frequently.

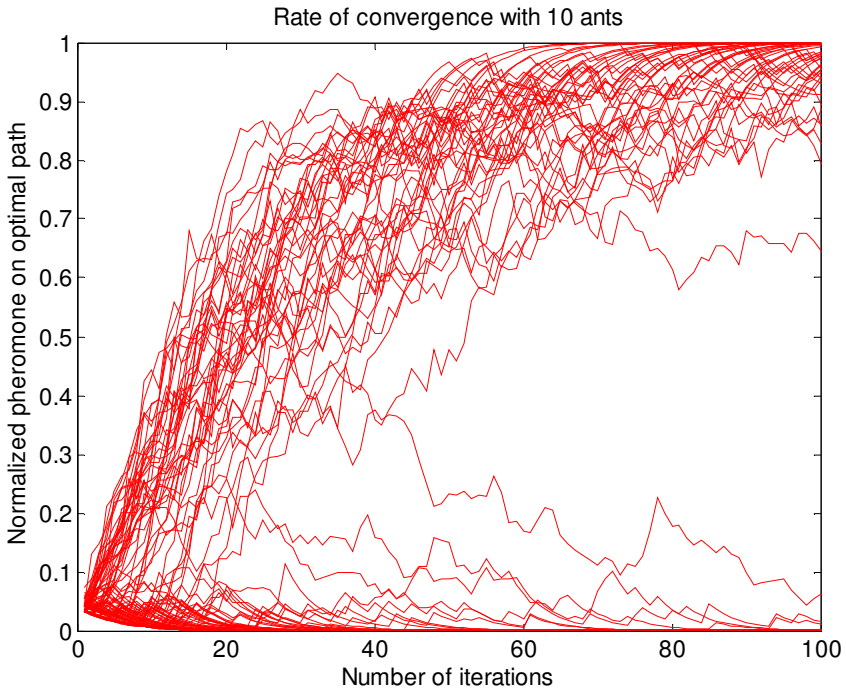

Figure 6-17. Elitist Ant System rate of convergence with 10 ants, $\rho = .2$

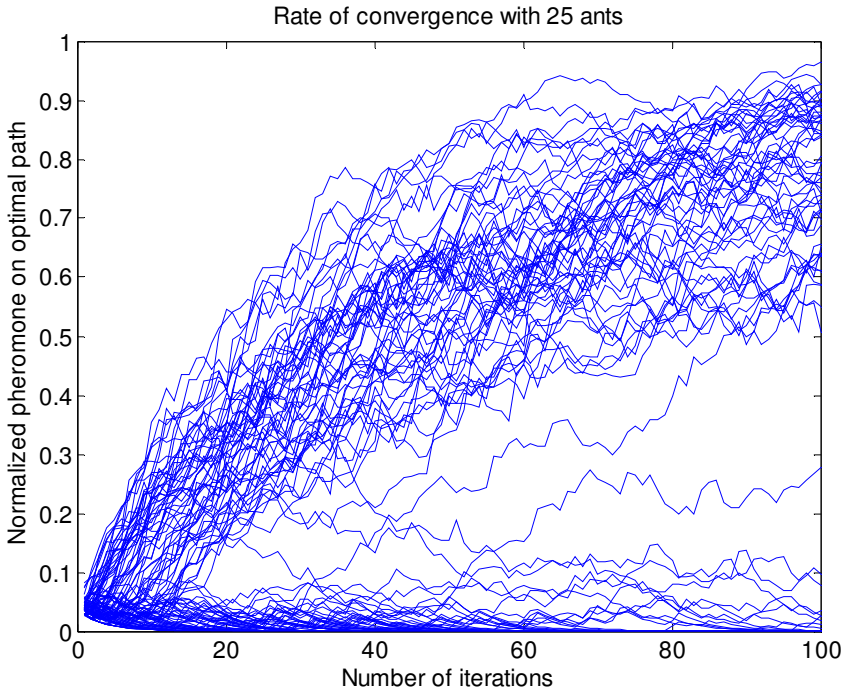

Figure 6-18. Elitist Ant System rate of convergence with 25 ants, $\rho = .2$

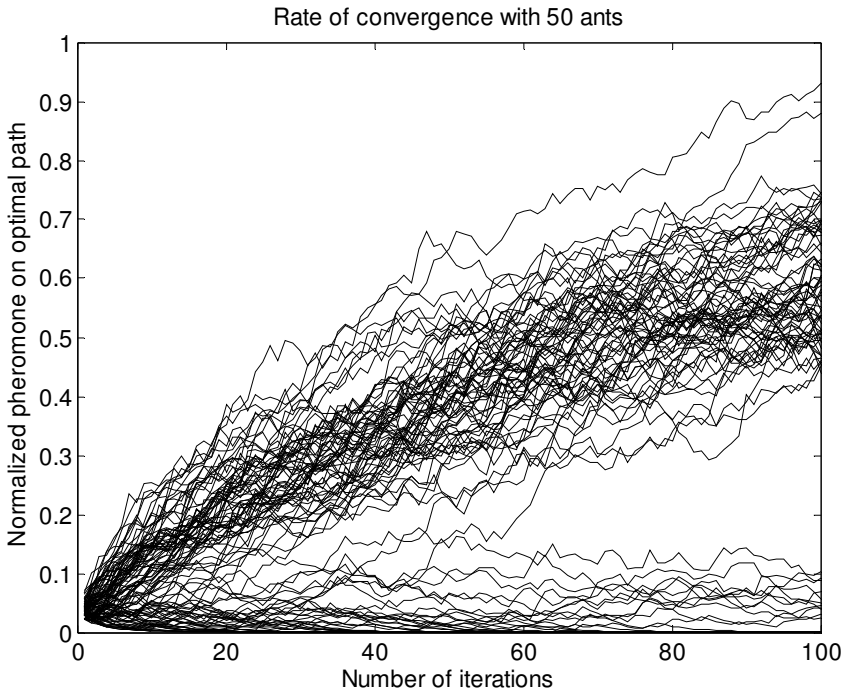

Figure 6-19. Elitist Ant System rate of convergence with 50 ants, $\rho = .2$

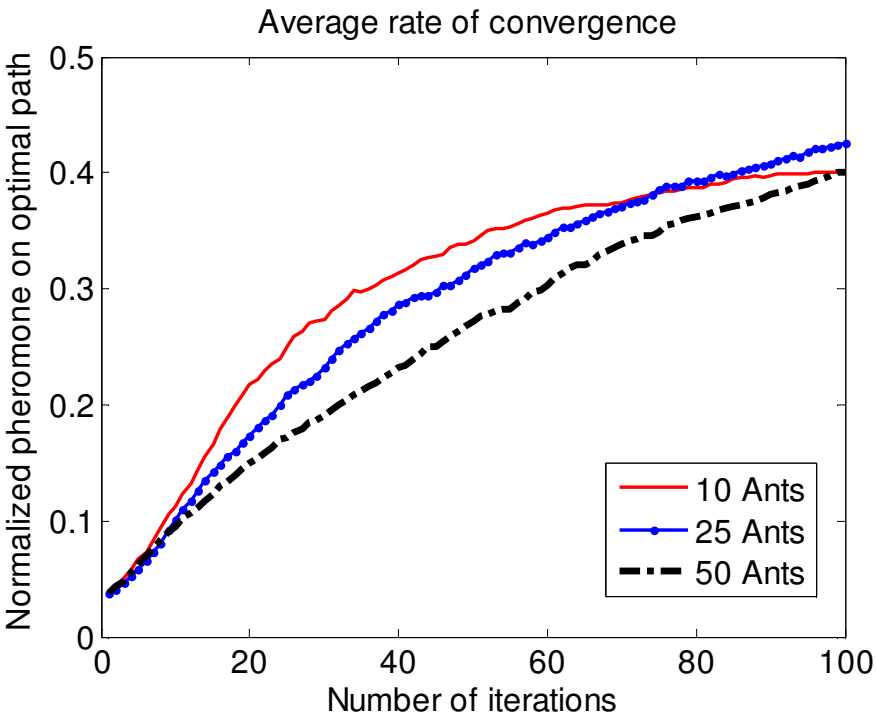

Figure 6-20. Elitist Ant System average rate of convergence with $\rho = .2$

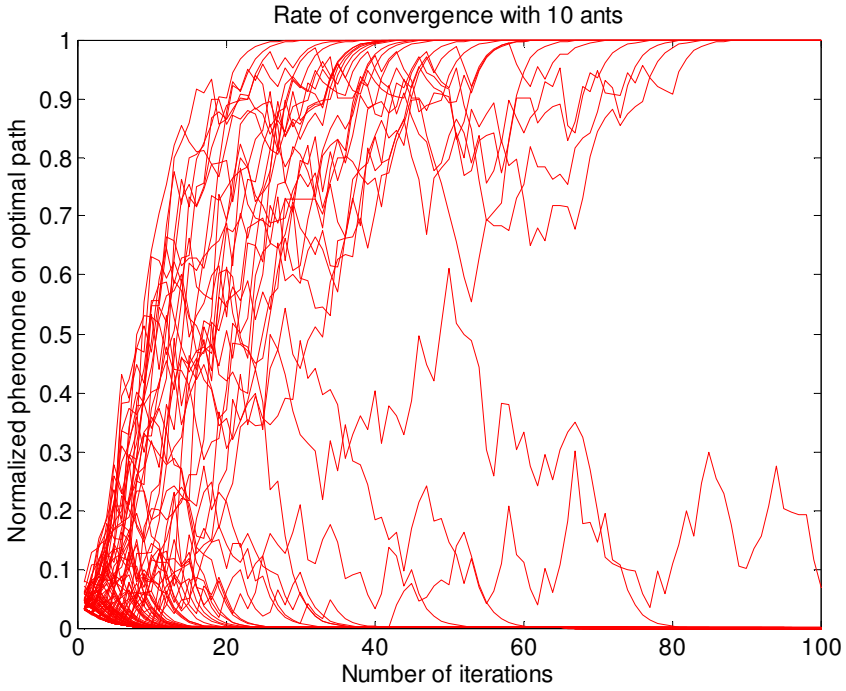

Figure 6-21. Elitist Ant System rate of convergence with 10 ants, $\rho = .4$

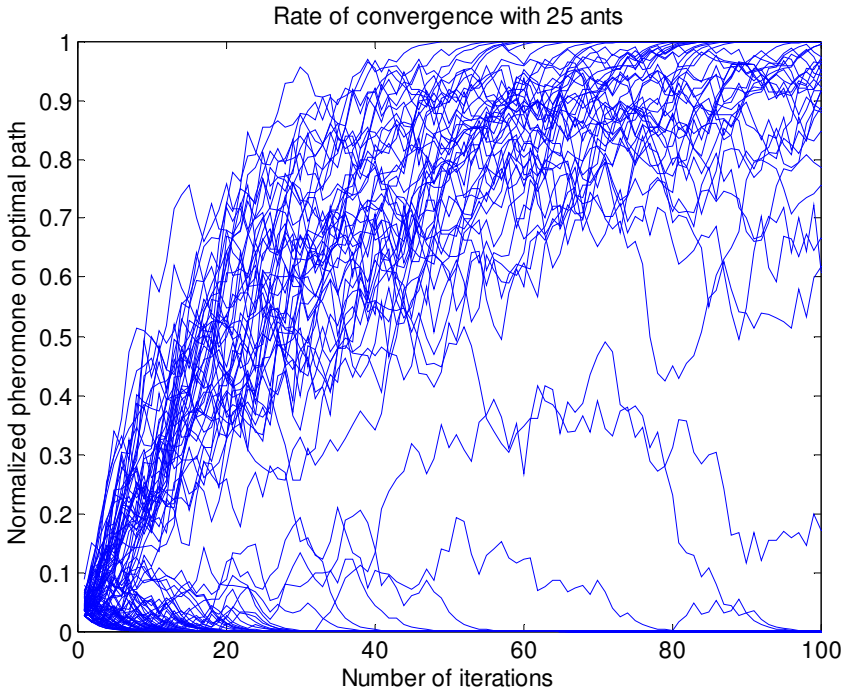

Figure 6-22. Elitist Ant System rate of convergence with 25 ants, $\rho = .4$

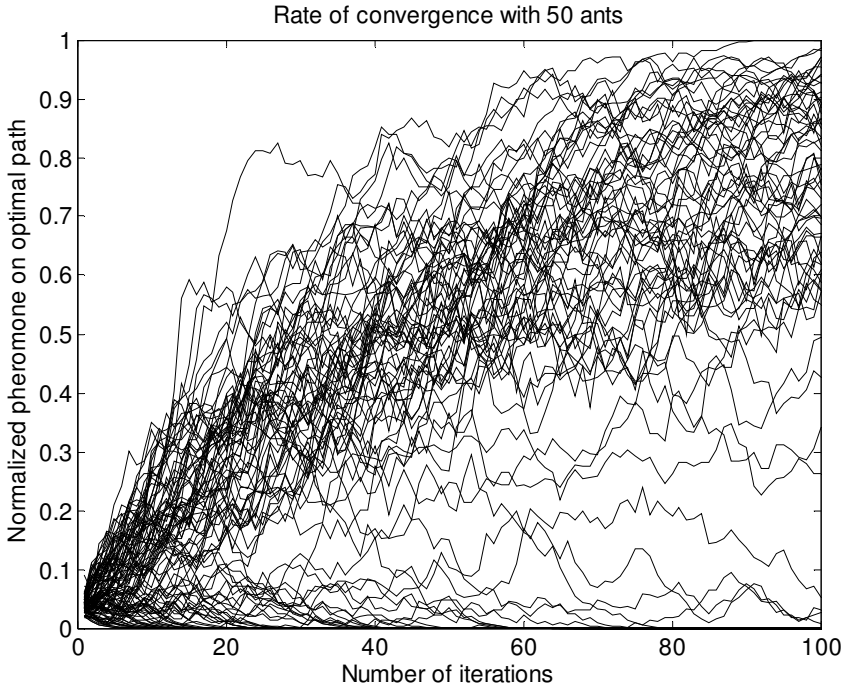

Figure 6-23. Elitist Ant System rate of convergence with 50 ants, $\rho = .4$

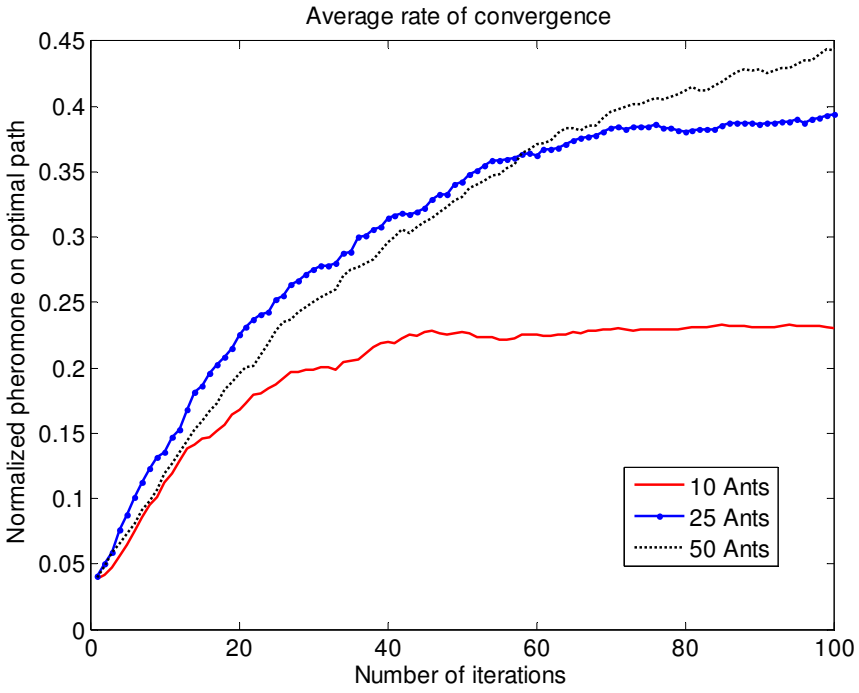

Figure 6-24. Elitist Ant System average rate of convergence with $\rho = .4$

6.5 Elitist Ant System with local search

When the Elitist ant system is used, the local search helps to find the best path and the elitist ant will update this path more often. The local search was performed every three iterations. In the figures below, the pheromone levels can increase significantly on multiples of three, because the optimal setting has been found and then the elitist ant updates this path on each iteration. Fewer ants can be used and still guarantee convergence to the optimal path. In these situations, using ten ants is sufficient most of the time, but there are too many trials where the optimal path is found later in the simulations. When the local search is used, the probability of not finding the optimal solution decreases significantly. As a result, the use of less pheromone evaporation is not

needed to reduce the probability of bad convergence. The slower evaporation only slows convergence without significant advantages.

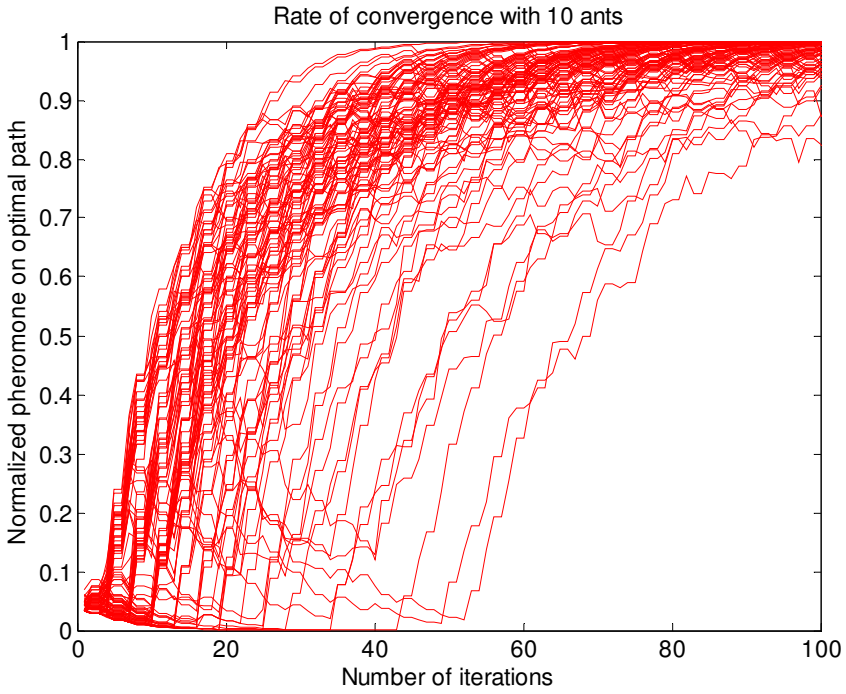

Figure 6-25. Elitist Ant System with local search rate of convergence with 10 ants, $\rho = .2$

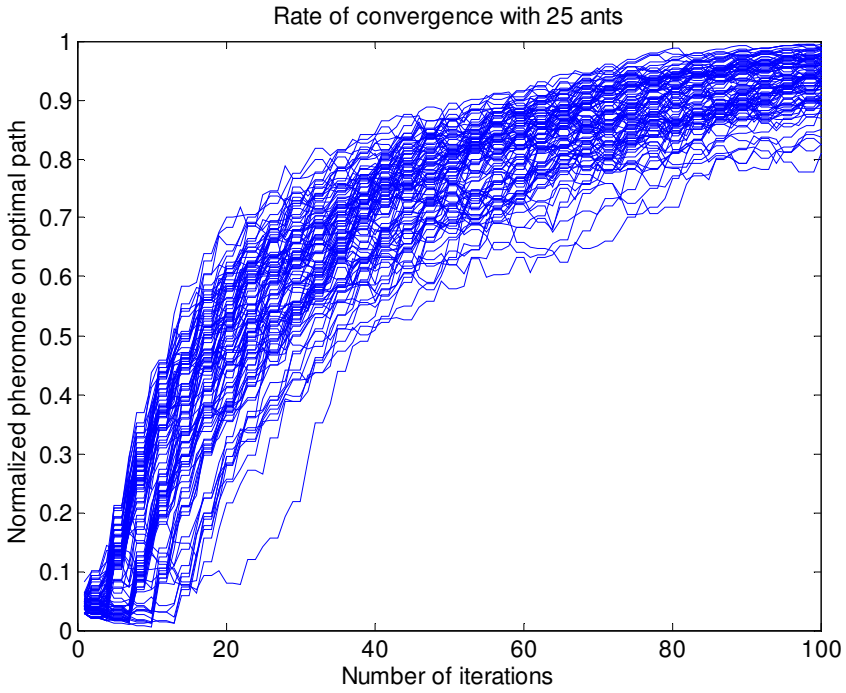

Figure 6-26. Elitist Ant System with local search rate of convergence with 25 ants, $\rho = .2$

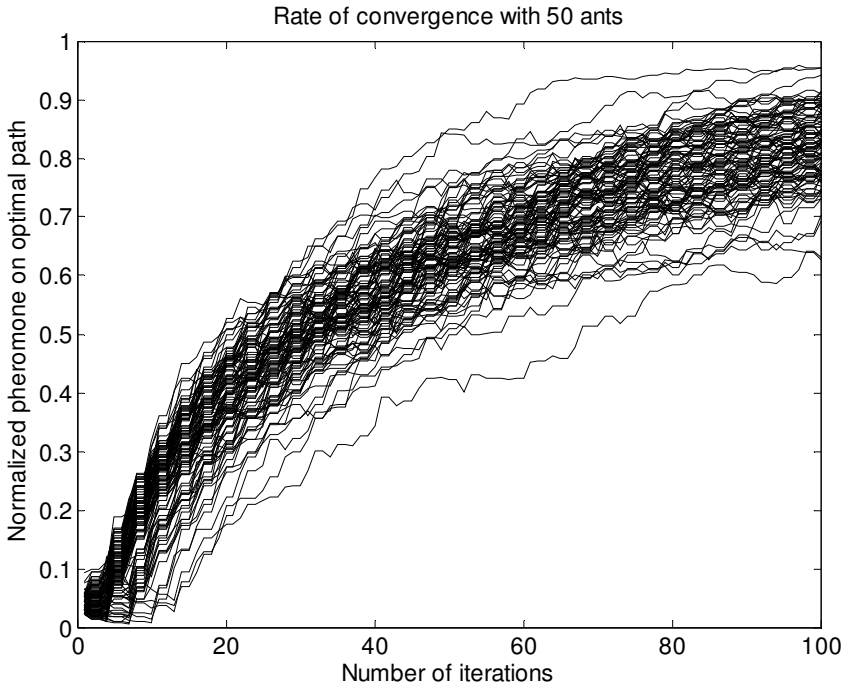

Figure 6-27. Elitist Ant System with local search rate of convergence with 50 ants, $\rho = .2$

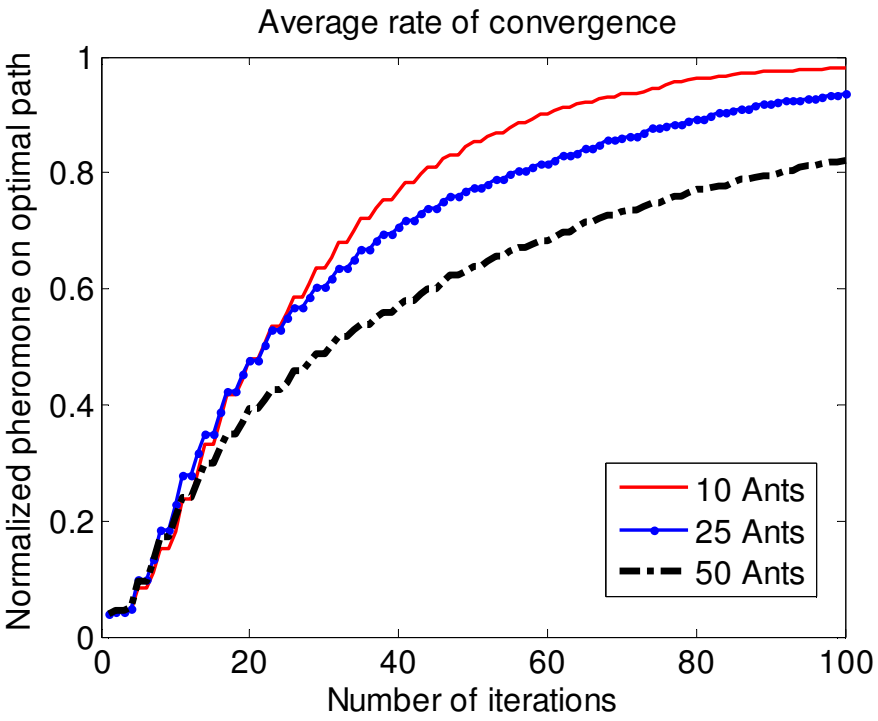

Figure 6-28. Elitist Ant System with local search average rate of convergence with $\rho = .2$

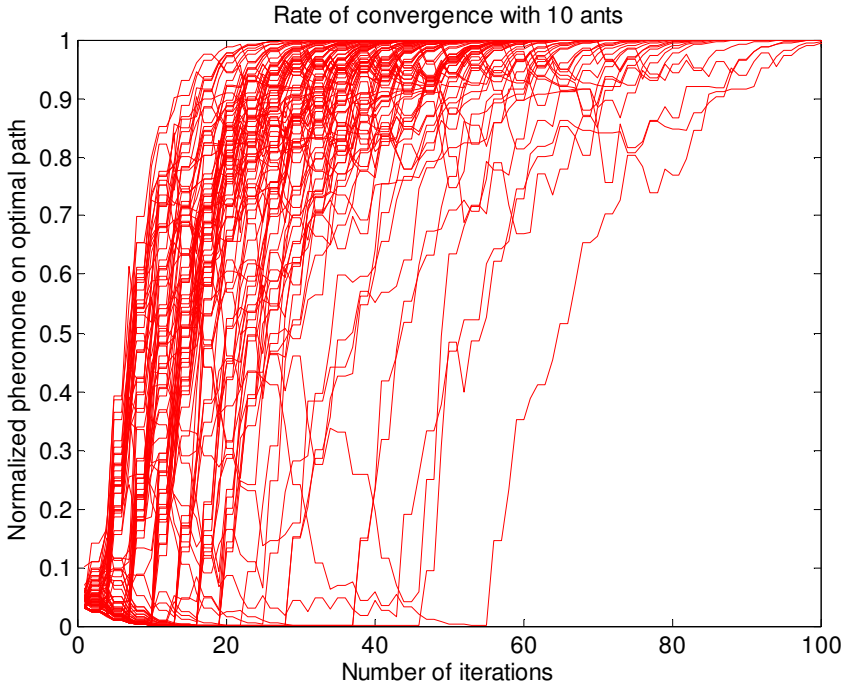

Figure 6-29. Elitist Ant System with local search rate of convergence with 10 ants, $\rho = .4$

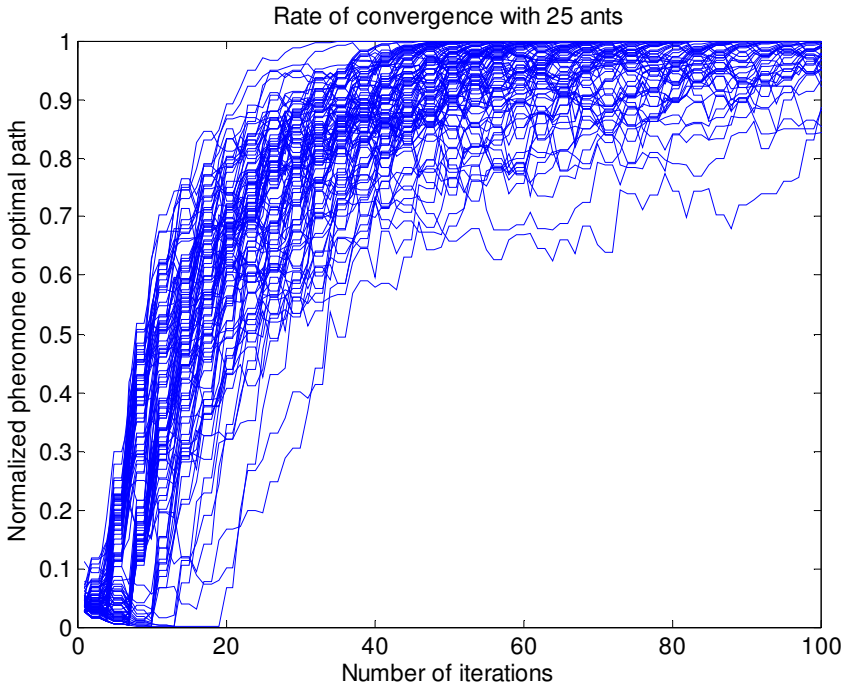

Figure 6-30. Elitist Ant System with local search rate of convergence with 25 ants, $\rho = .4$

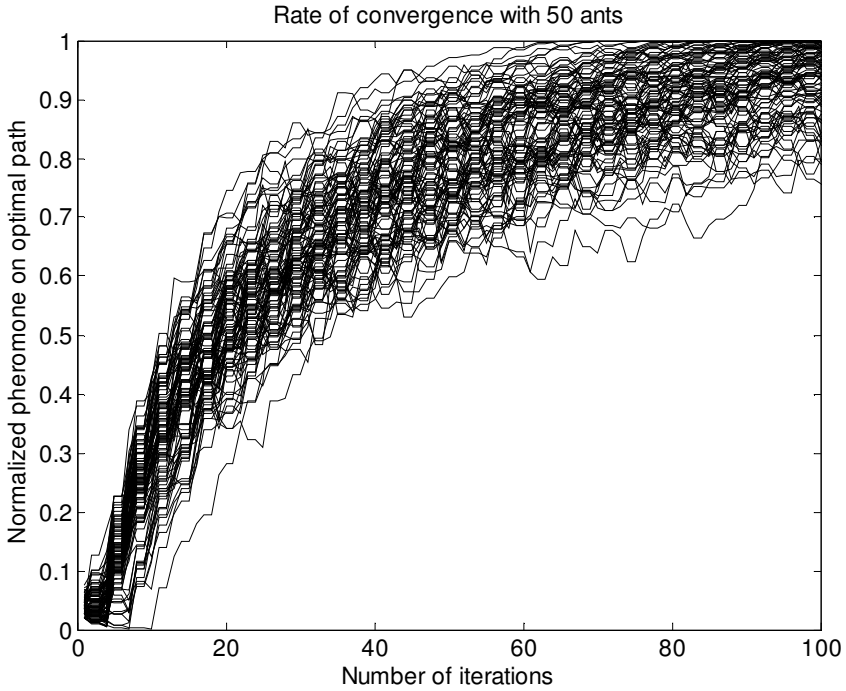

Figure 6-31. Elitist Ant System with local search rate of convergence with 50 ants, $\rho = .4$

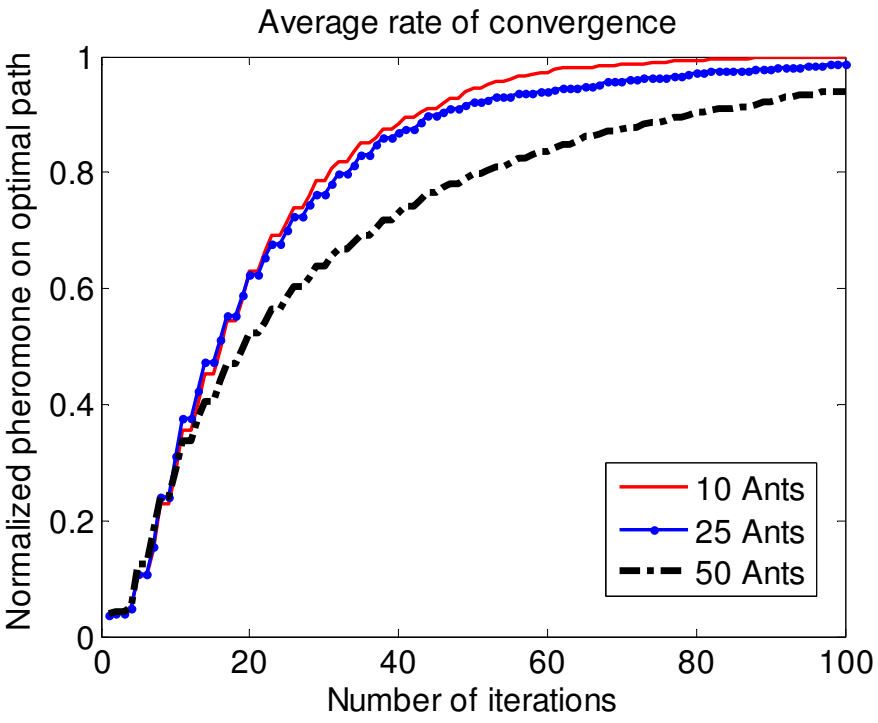

Figure 6-32. Elitist Ant System with local search average rate of convergence with $\rho = .4$

6.6 Heuristic Information

By adding the heuristic in formula (5.2) convergence to the optimal is even faster and 10 ants are sufficient for simulations. Once again, evaporating pheromone slower is not needed to find the correct solution; it only makes the convergence slower.

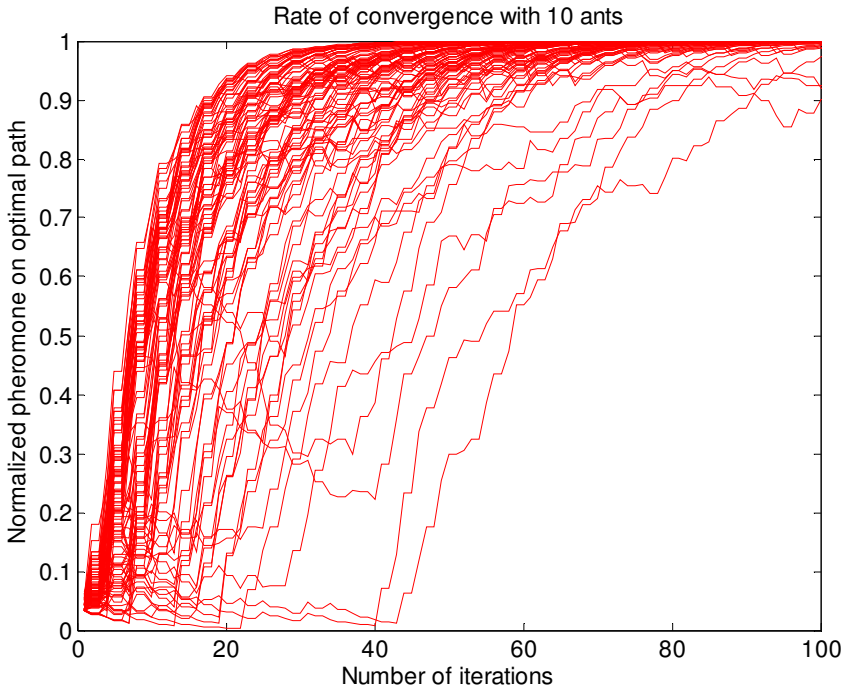

Figure 6-33. Elitist Ant System with local search and heuristics rate of convergence with 10 ants, $\rho = .2$

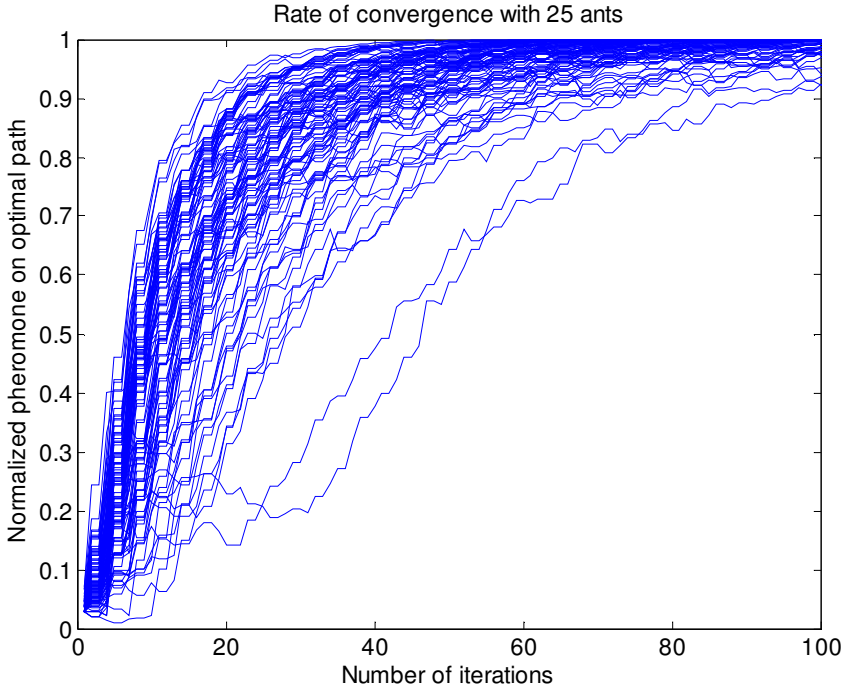

Figure 6-34. Elitist Ant System with local search and heuristics rate of convergence with 25 ants, $\rho = .2$

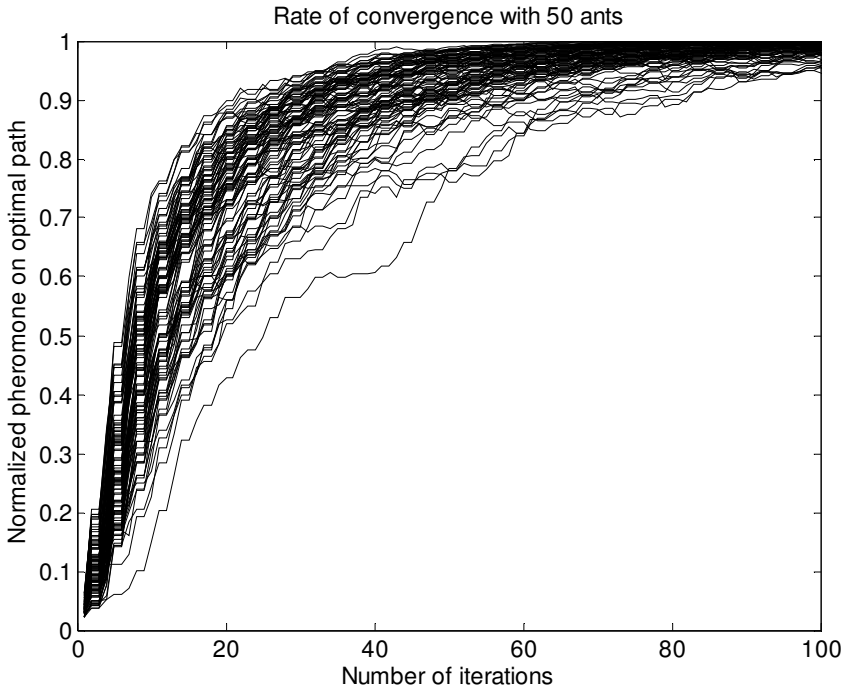

Figure 6-35. Elitist Ant System with local search and heuristics rate of convergence with 50 ants, $\rho = .2$

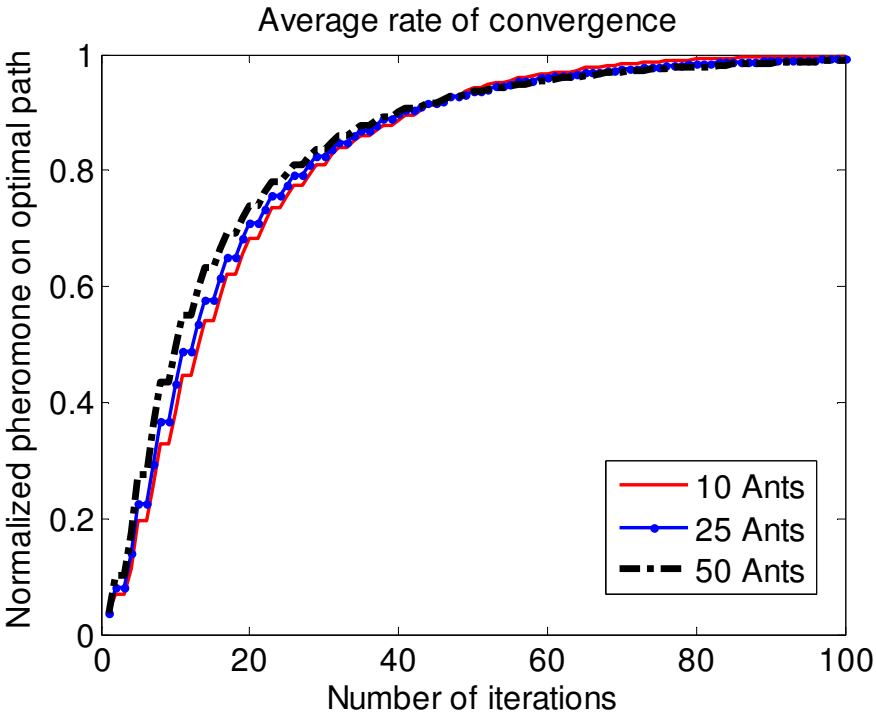

Figure 6-36. Elitist Ant System with local search and heuristics average rate of convergence with $\rho = .2$

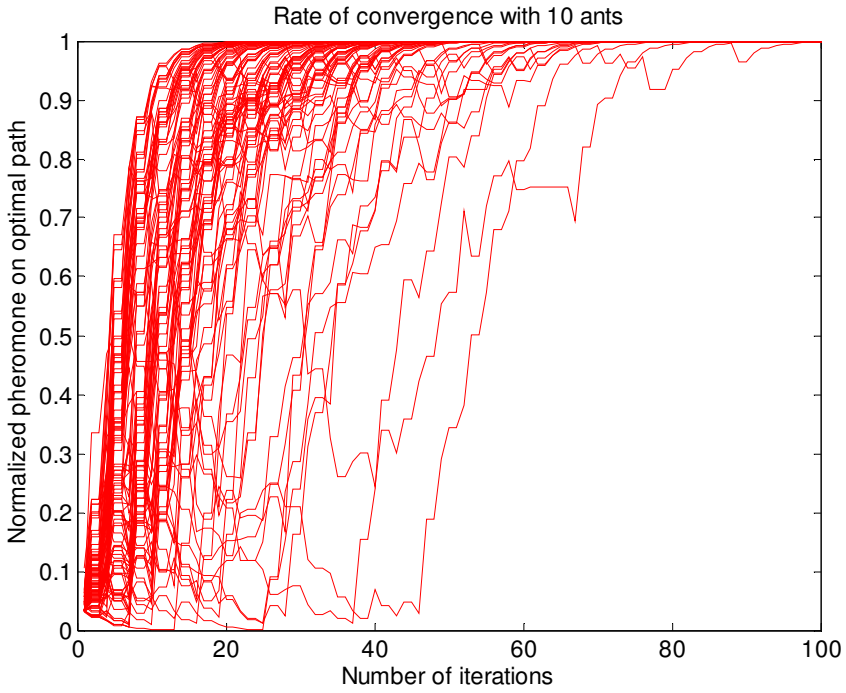

Figure 6-37. Elitist Ant System with local search and heuristics rate of convergence with 10 ants, $\rho = .4$

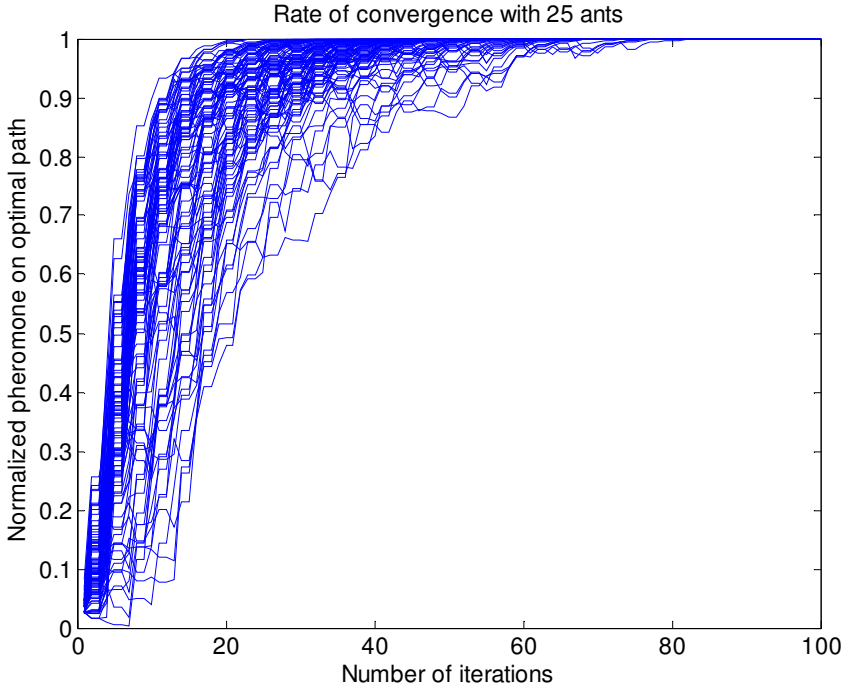

Figure 6-38. Elitist Ant System with local search and heuristics rate of convergence with 25 ants, $\rho = .4$

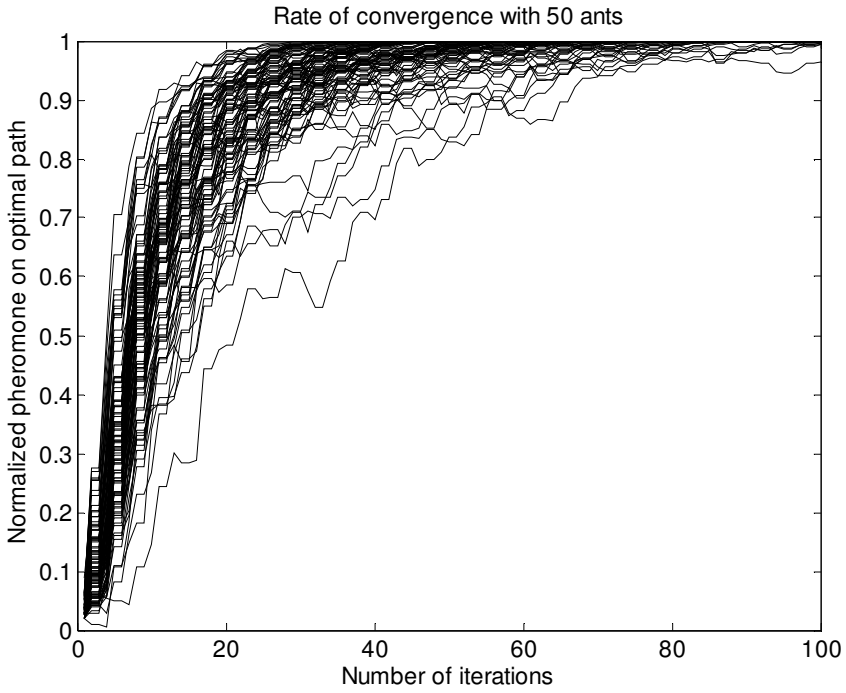

Figure 6-39. Elitist Ant System with local search and heuristics rate of convergence with 50 ants, $\rho = .4$

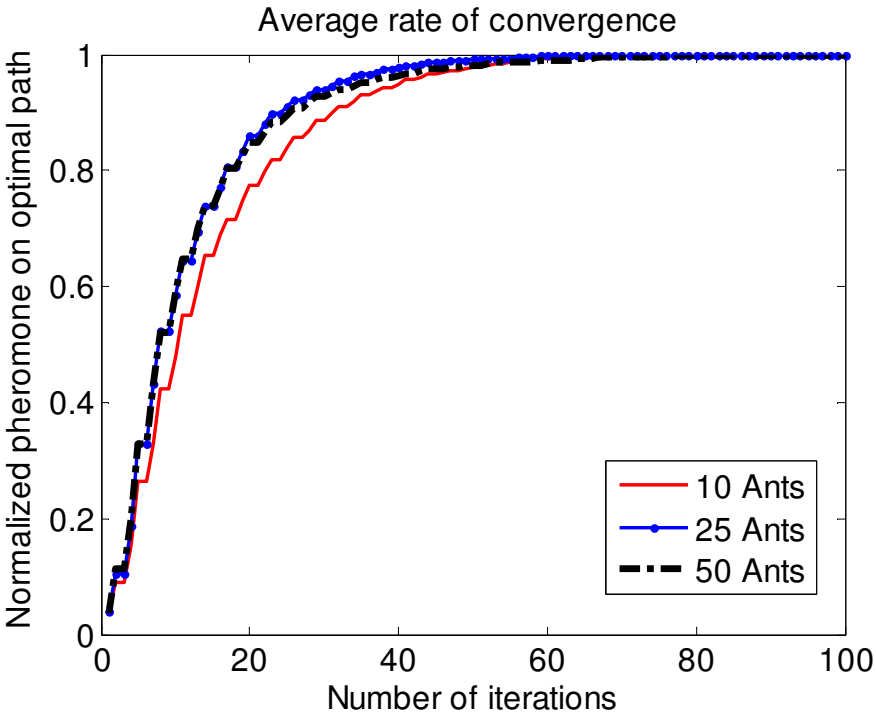

Figure 6-40. Elitist Ant System with local search and heuristics average rate of convergence with $\rho = .4$

6.7 Rank-based Ant System

To achieve the fastest rates of convergence, ants' solutions are ranked and the solutions with the highest cost function are discarded. Once the optimal solution is found pheromone accumulates more rapidly, because the best solutions are weighted heavier and the bad solutions are ignored. The rank-based ant system works the best and requires the fewest ants. As a result, it is used in simulations to test the effectiveness of ACO control.

The evaporation step is used to eliminate pheromone on bad solutions. But, the rank based update gives less weight to less optimal solutions[6]. As a result, pheromone evaporation is slower.

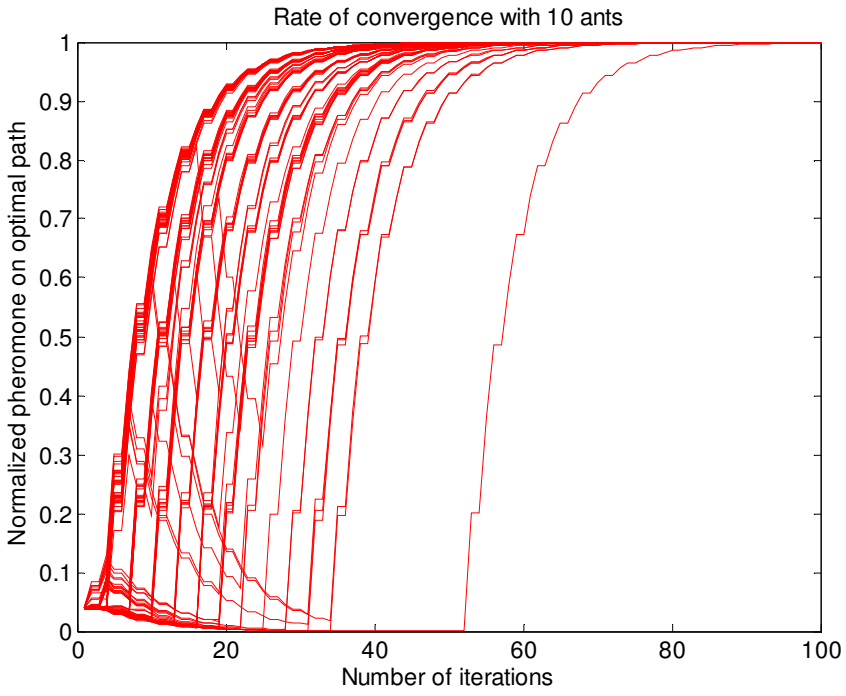

Figure 6-41. Rank-based Ant System with local search and heuristics rate of convergence with 10 ants, $\rho = .2$

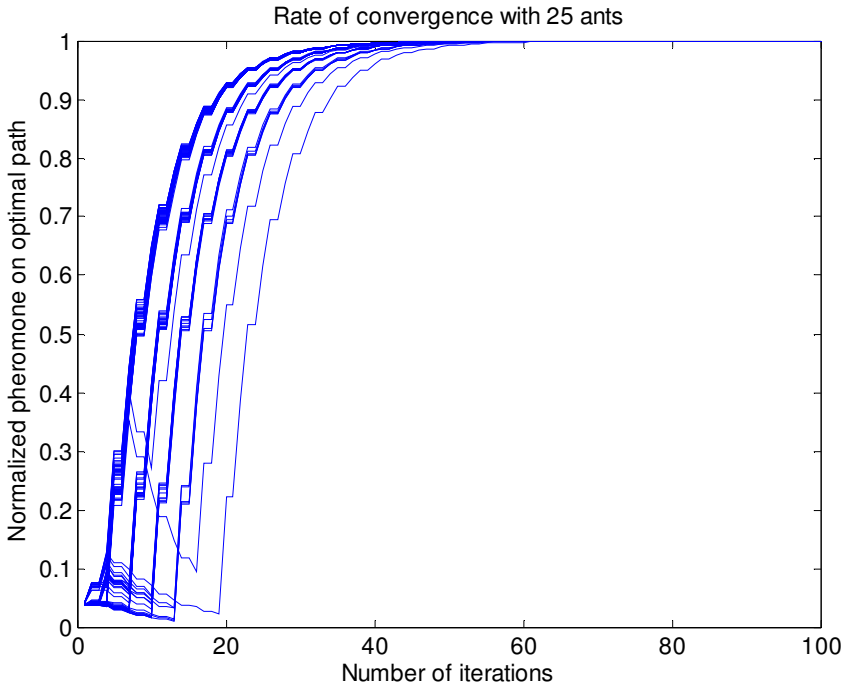

Figure 6-42. Rank-based Ant System with local search and heuristics rate of convergence with 25 ants, $\rho = .2$

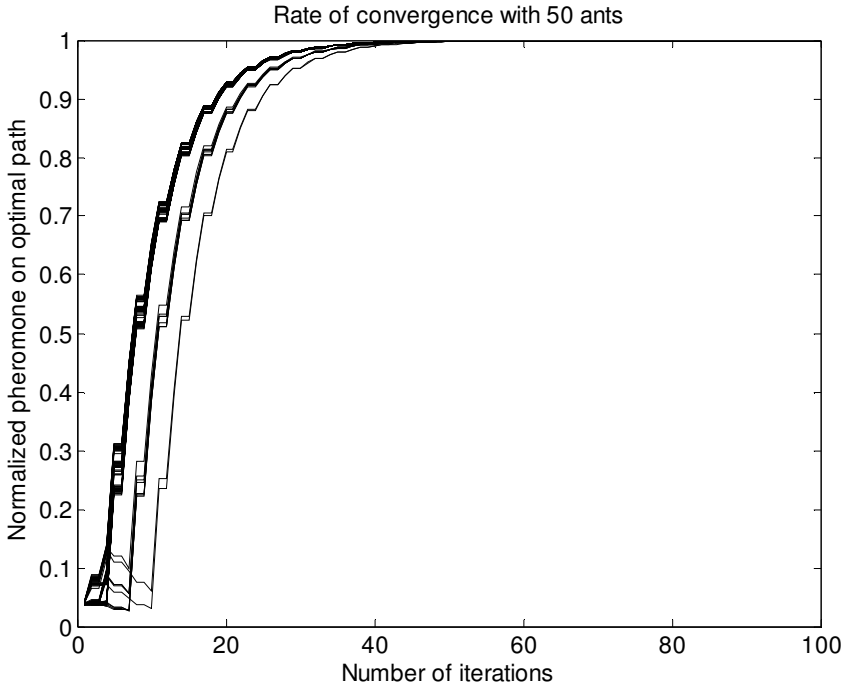

Figure 6-43. Rank-based Ant System with local search and heuristics rate of convergence with 50 ants, $\rho = .2$

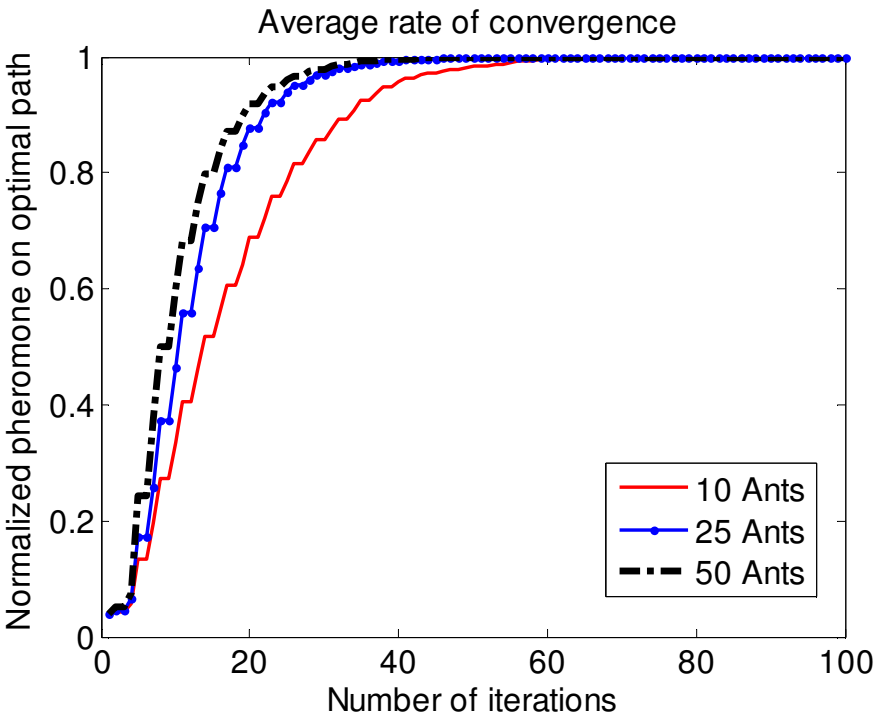

Figure 6-44. Rank-based Ant System with local search and heuristics average rate of convergence with $\rho = .2$

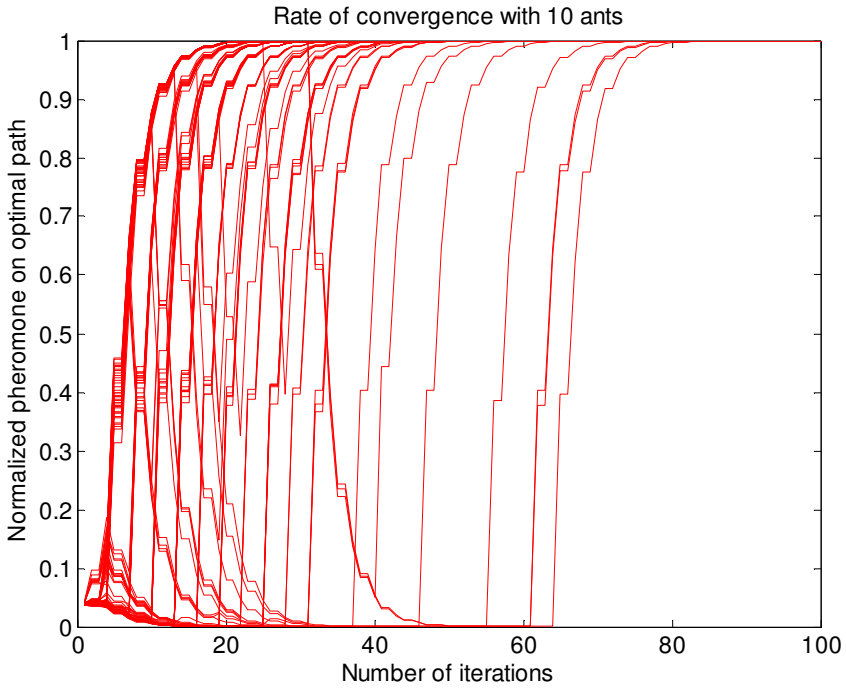

Figure 6-45. Rank-based Ant System with local search and heuristics rate of convergence with 10 ants, $\rho = .4$

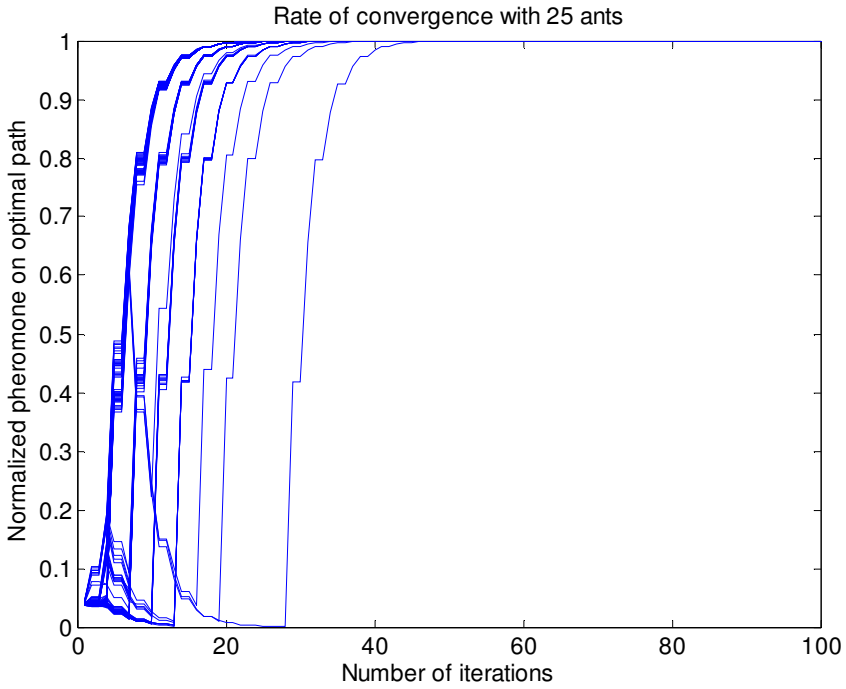

Figure 6-46. Rank-based Ant System with local search and heuristics rate of convergence with 25 ants, $\rho = .4$

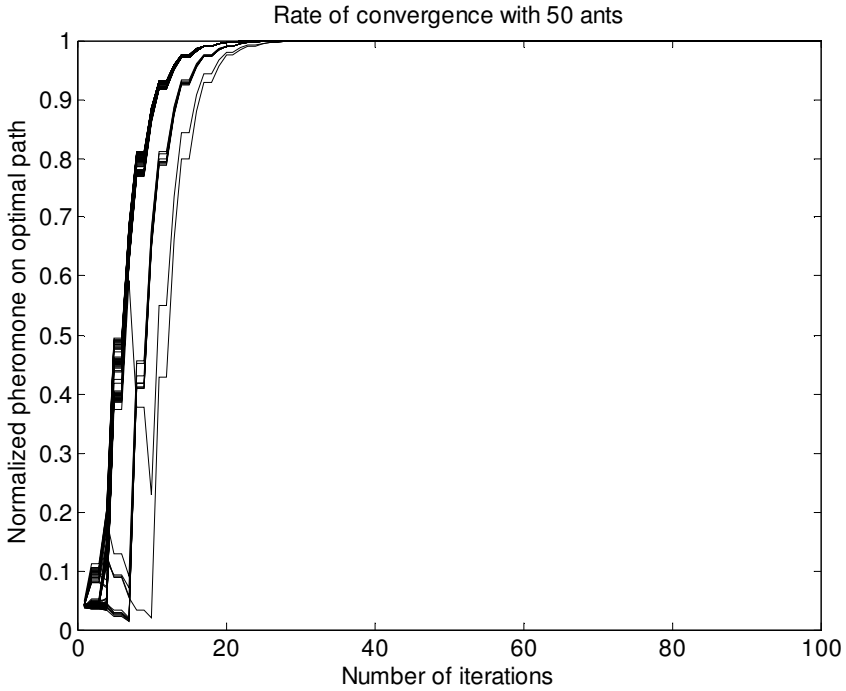

Figure 6-47. Rank-based Ant System with local search and heuristics rate of convergence with 50 ants, $\rho = .4$

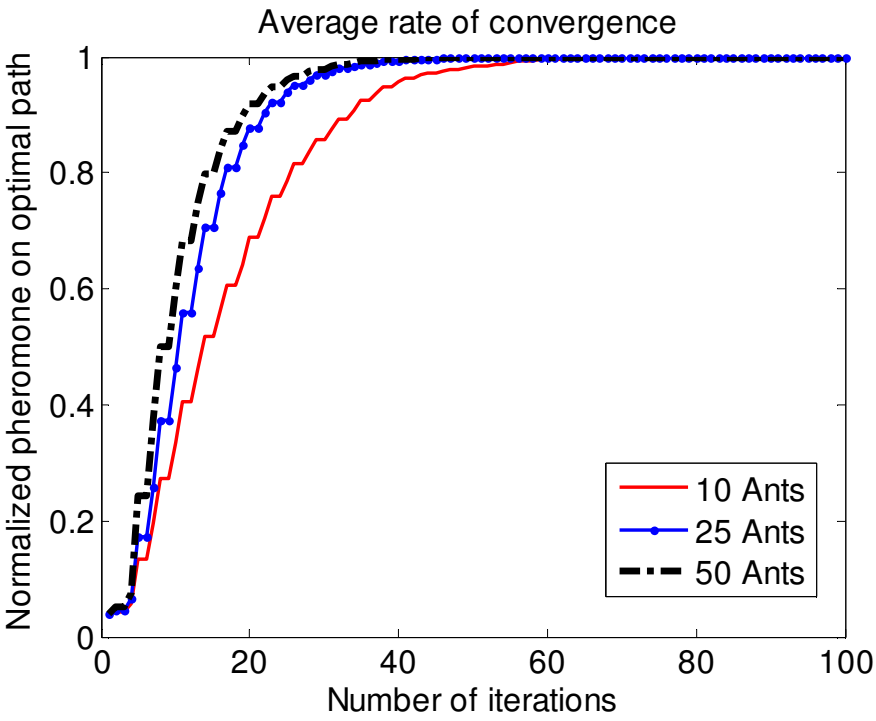

Figure 6-48. Rank-based Ant System with local search and heuristics average rate of convergence with $\rho = .4$

6.8 Pheromone convergence during traffic simulations

The convergence rates in the simple problem when the queue begins empty between the elitist ACO and the rank based method when both use local search and heuristic information are similar. But when running in a traffic simulation, the elitist doesn't converge as fast, and pheromone levels can get stuck, so the solution never converges. Figure 6-49 shows the convergence levels of pheromone during a 20 minute simulation with a traffic volume of eight hundred fifty vehicles per hour per movement. The elitist ant system is used with local search and heuristic weight. Ten ants are used and pheromone evaporation, $\rho = .4$. If the weight of the elitist ant is increased then full convergence occurs more often, but suboptimal solutions can be weighted too heavily,

causing pheromone to concentrate on them. As a result, many iterations of the local search are required to find the optimal solution.

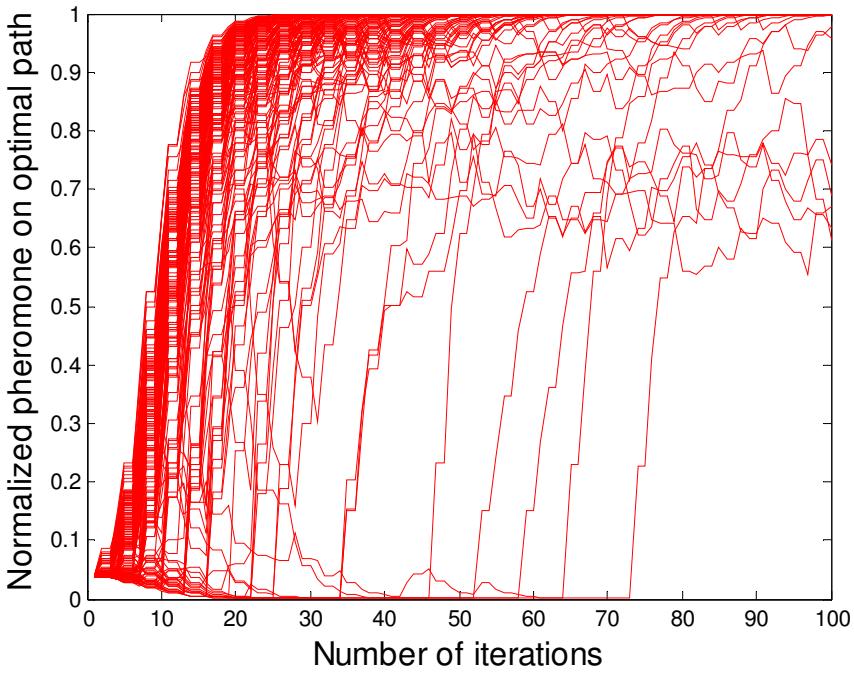

Figure 6-49. Convergence rates during traffic simulation using Elitist ACO with local search

In the rank based method, the same qualitative behavior shown in the figures of this chapter is seen in the convergence during a traffic simulation. Figure 6-50 shows the convergence levels of pheromone during a 20 minute simulation with a traffic volume of eight hundred fifty vehicles per hour per movement. The rank-based ant system is used with local search and heuristic weight. Ten ants are used and pheromone evaporation, $\rho = .2$.

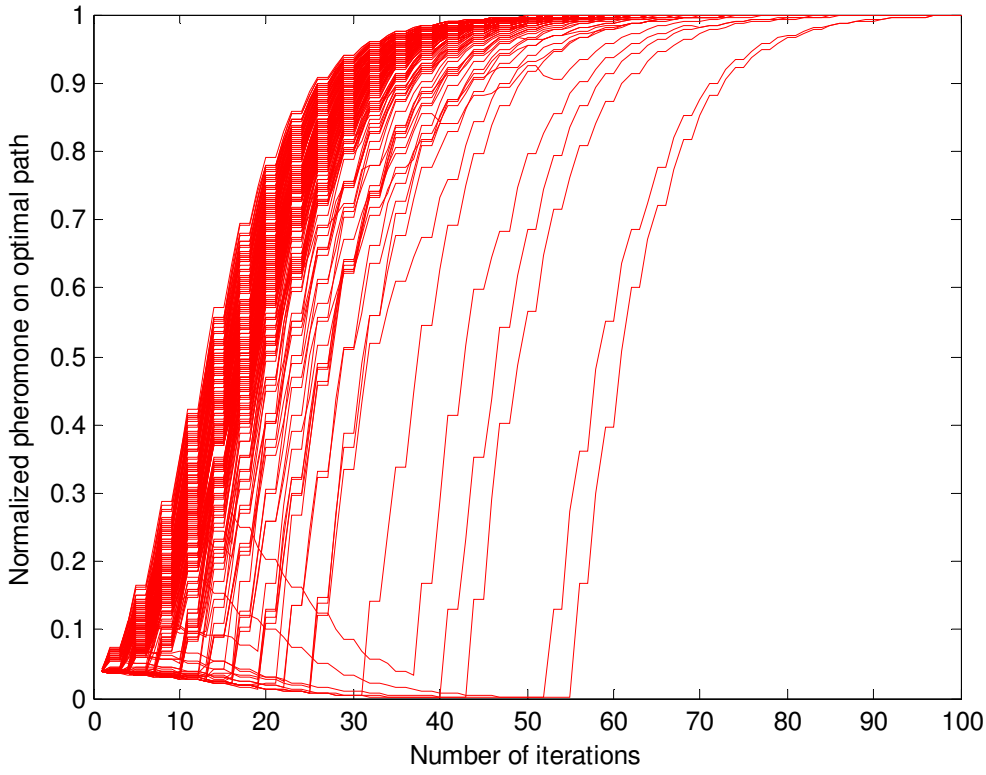

Figure 6-50. Convergence rates during traffic simulation using Rank-based ACO

6.9 Choice of parameters α and β

Proper choice of parameter ACO leads to faster and more accurate convergence of the artificial ant's pheromone. One important parameter choice is the weight α and β, of the pheromone levels and heuristic information respectively, in the ant solution construction step. The heuristic weight function is an exponential, so the weight β and the heuristic scaling c are interchangeable. Observe that $\eta_{ij}^{\beta} = \left(e^{((q^g(t_1)-1)hw-|t_i-t_j|)/c}\right)^{\beta} = e^{((q^g(t_1)-1)hw-|t_i-t_j|)\beta/c}$. For this reason, β is fixed at

one and c is varied. The following figures show the effects of bad parameter choices in the rank-based ant system with local search and heuristic weight.

The values of τ and η are between 0 and 1, so exponentiating them by a parameter less than one increases smaller values more than larger ones. The extra weight to suboptimal solutions reduces the effects of the pheromone evaporation. Ants choose the suboptimal routes more often, so the pheromone does not fully evaporate. Several different paths are selected and updated with pheromone on each iteration, so pheromone does not completely converge to the optimal solution. Figure 6-51 shows pheromone convergence with $\alpha = 1/2$, and Figure 6-52 shows pheromone convergence with $c = 20$.

Conversely, if α is too large then solutions are weighted to heavily and less exploration is done. The solutions chosen early are updated heavily and many iterations of the local search are required to find the optimal solution, if it is found at all. Figure 6-53 shows pheromone convergence with $\alpha = 2$, note that the pheromone often takes many more iterations before it begins to concentrate. If c is too small then the heuristic information is considered to strongly. If the optimal solution is not being weighted heaviest by the heuristic information then and pheromone might not fully converge. The extra weighting of the heuristic information causes the ants to choose this solution more often; since it is close to optimal it receives large updates. The feedback is too large and pheromone never disappears on a suboptimal path. Figure 6-54 shows pheromone convergence with $c = 1$.

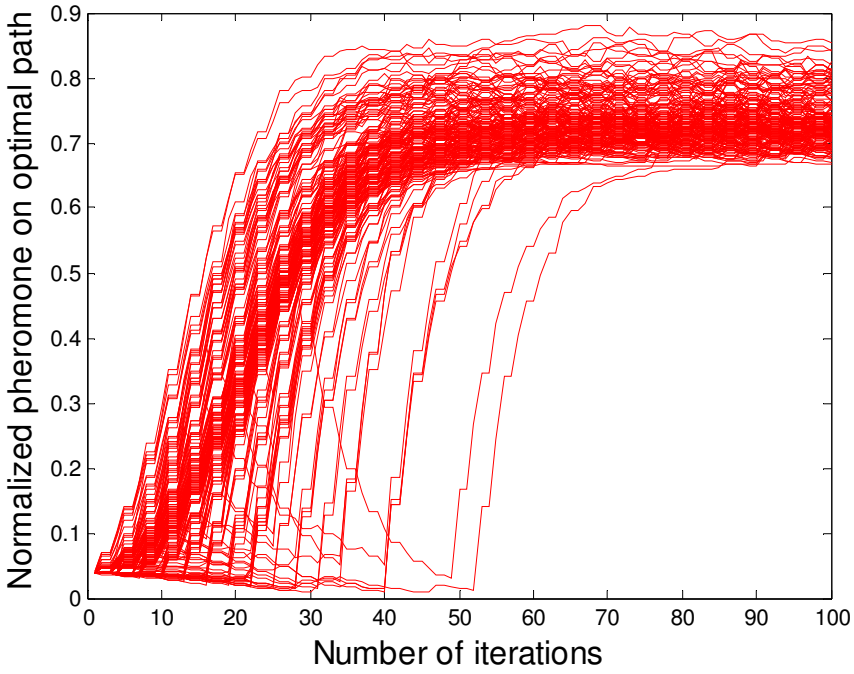

Figure 6-51. Pheromone convergence with $\alpha = 1/2$

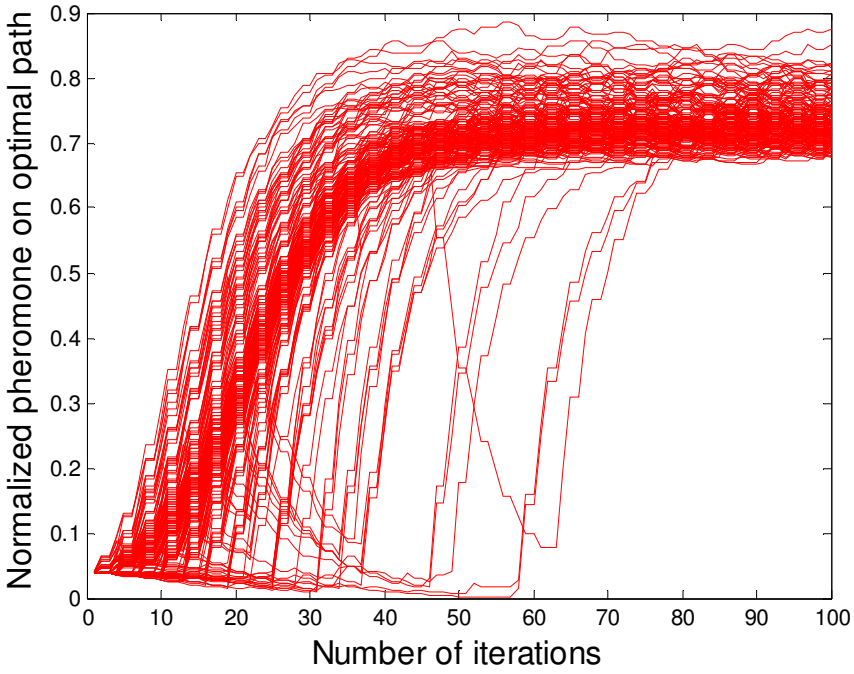

Figure 6-52. Pheromone convergence with $c = 20$

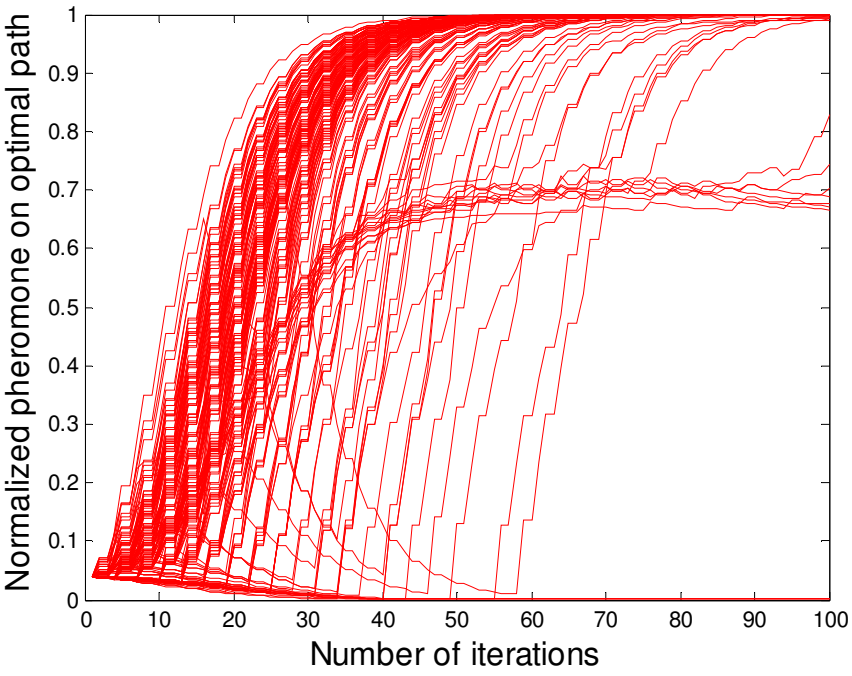

Figure 6-53. Pheromone convergence with $\alpha = 2$

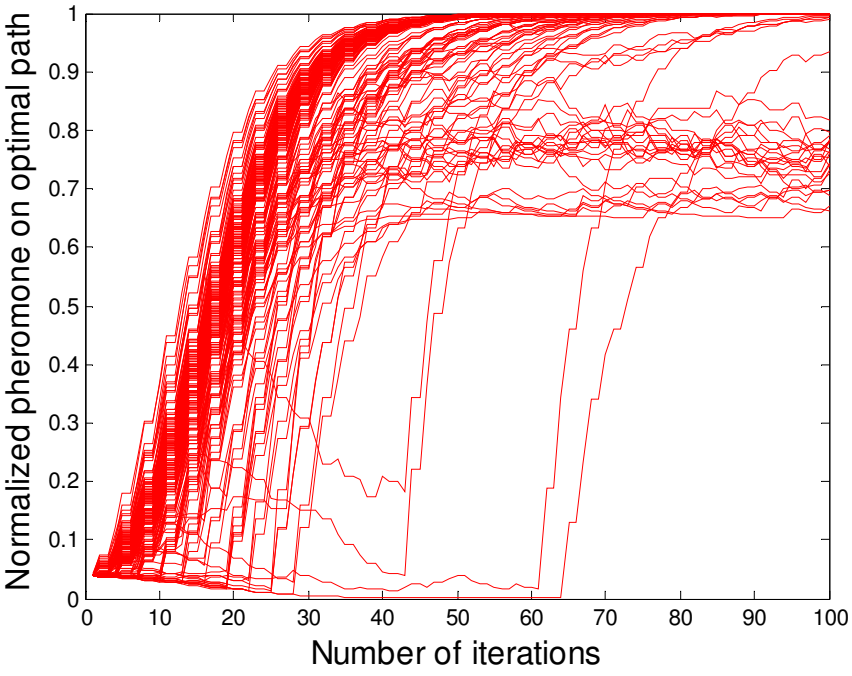

Figure 6-54. Pheromone convergence with $c = 1$

6.10 Average delay

In this section, the control strategy of the ACO is compared with the traditional fully actuated controller, described in Chapter 5. The traffic simulation is run for ten minutes; allowing traffic to reach a steady state and reduce the effects of initial conditions. Then, the delay of each vehicle is recorded for the next ten minutes. A vehicle's delay is defined to be its departure time minus its arrival time. At the end of each trial, the average delay over the second ten minute period is recorded. The average delay is

$$\text{average delay} = \frac{\sum_i dt_i - at_i}{N} \tag{6.1}$$

where at_i and dt_i is the arrival time and departure time of the i^{th} vehicle, respectively. The sum is taken over all vehicles that arrive in the second ten minute interval, and N is the total number of arrivals during this interval. All delays are record in seconds. The ACO algorithm and the Fully Actuated control were run on forty sets of vehicle arrival data. The table of the average delay time in each trial is given in Appendix A. The time resolution of the simulations is .01 seconds.

In low traffic flows the fully actuated controller performs better. When the traffic flow is less than six hundred vehicles per hour per movement, the expected vehicle inter-arrival time is greater than six seconds. The minimum green time is five seconds and with the two second all red time, so the minimum time between a phase transition is seven seconds. The probability of a vehicle arriving during a red signal is small but cannot be ignored. The fully actuated controller is better suited for this situation because it waits for a vehicle on red to arrive before changing.

Once traffic flow is larger than six hundred vehicles per hour per movement, the number of vehicles that arrive per red signal is frequently greater than one. With the higher arrival rates the red movements need to be considered when creating traffic control signals. The fully actuated controller gives too much preference to the green direction. On the other hand, the objective function of the rolling horizon control takes all movements into account.

Figure 6-55 plots the average vehicle delay of each control strategy over all trials. As traffic approaches saturation, the average delay for the fully actuated control increases much faster than in ACO control. Figure 6-56 shows the minimum and maximum average wait times from a trial. As a second performance measure, the average queue lengths from each control strategy are compared. In Figure 6-57 the average queue lengths are shown. They behave similarly to the average delays, with the queue lengths growing much faster in fully actuated control as the arrival rate approaches saturation. Figure 6-58 shows the longest average queue length from a trial.

The ant colony simulations run an order of magnitude faster than real-time, so they can be effectively implemented in real time systems. The rank-based ant system using local search, elitist ant, heuristic weights and ten ants takes eight minutes to run twenty minutes of simulation.

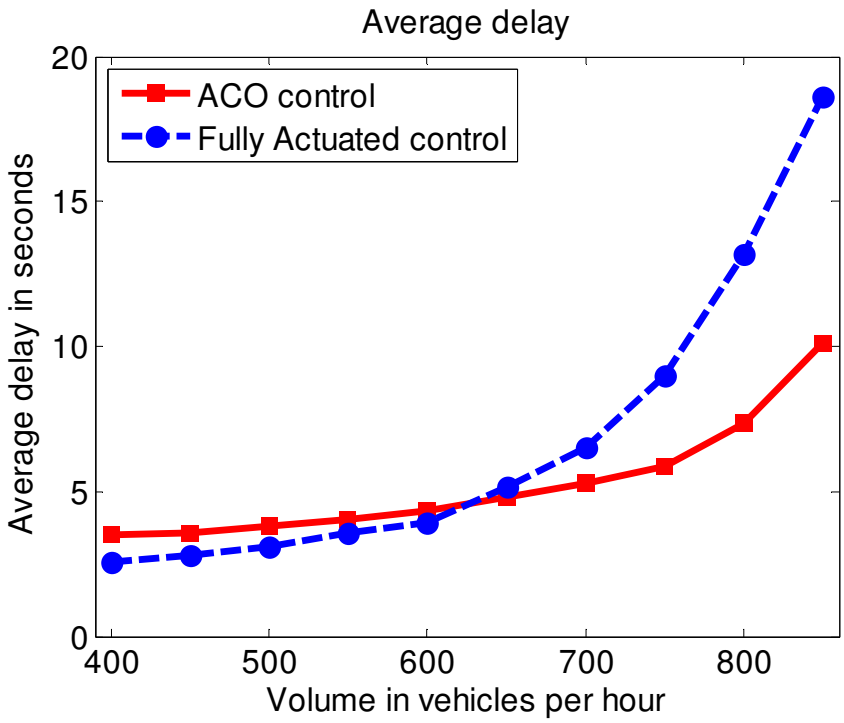

Figure 6-55. Average delay

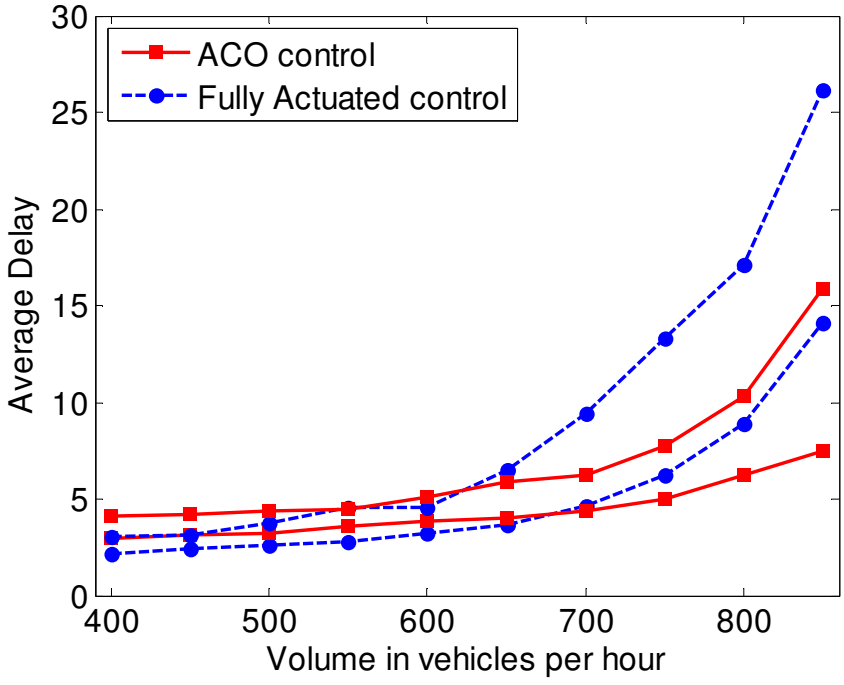

Figure 6-56. Upper and lower bounds on average vehicle delay

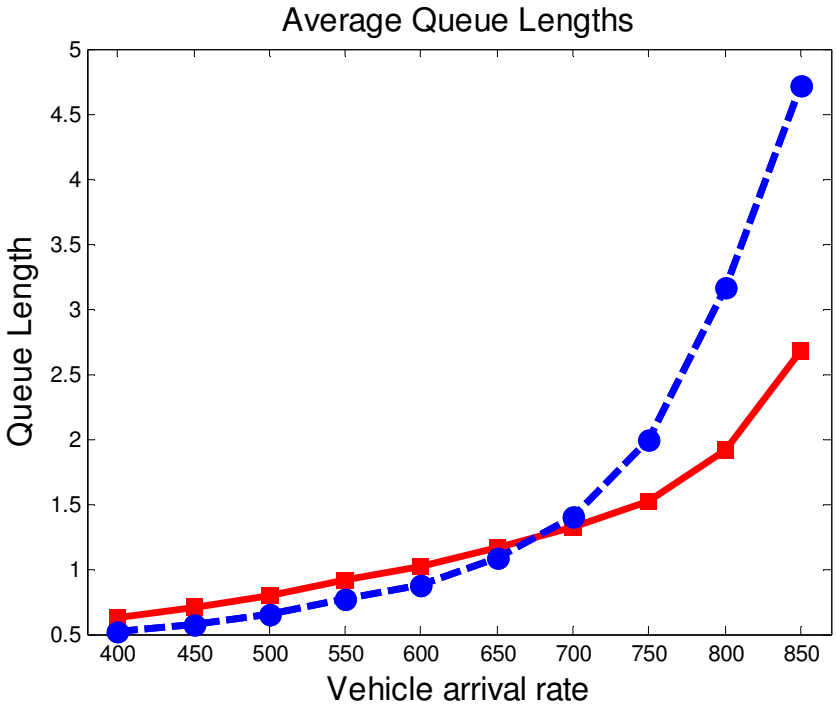

Figure 6-57. Average Queue Length

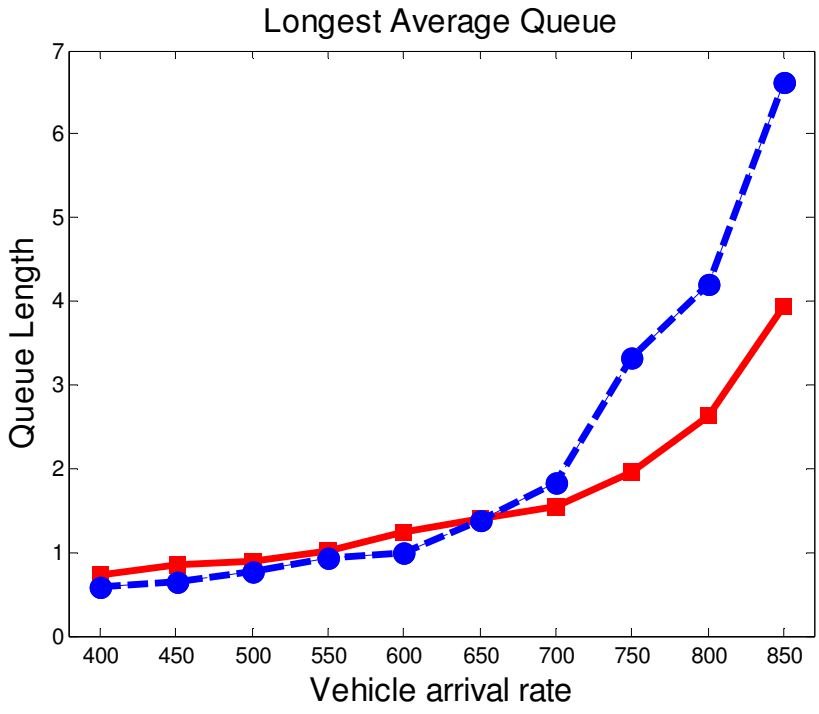

Figure 6-58. Longest Average Queue Length

Chapter 7 Conclusions and Future Works

Effective traffic signal control is the most efficient way to manage traffic flows. Good traffic management reduces user delay, fuel consumption and leads to less network congestion. Because traffic flows are nonlinear and stochastic, there are many difficulties in creating good traffic signal policy. Control strategies must be responsive to fluctuations in traffic flow that can occur on short time scales. Additionally, longer time scales must be considered to reach global optimality.

In this thesis, ant colony optimization is used in the control of traffic signals. ACO has successfully been applied to many combinatorial optimization problems. The probabilistic searches of ant colony algorithms work well with the stochastic nature of traffic flows. Ant colony optimization handles nonlinearities and non-differentiability of objective functions. Additionally, it is successfully applied to the effective rolling horizon control technique.

The use of ant colony optimization in traffic control outperforms traditional fully actuated control in high traffic demand and warrants further evaluation on more complex signal systems. In computer simulations, the ACO runs faster than real time, making it a viable option in real time control. With the addition of heuristic information and rank-based ant updates the optimal solution is found quickly. Once the optimal solution is found, pheromone accumulates quickly.

In addition to evaluating the average delay of vehicles, the convergence rates and accuracy of various ACO methods is evaluated. For the traffic signal problem, the Elitist ant system performs well. The addition of a local search helps pheromone to converge to the optimal solution. Including heuristic information from the queue length increases the

convergence speed. Once the rank based pheromone update is added pheromone accumulates on the optimal solution very quickly once it is found.

The results on a single intersection warrant further research on more complicated systems. Isolated intersections can be made more complex by adding turning lanes. When turning lanes are added the signal control follows an eight phase dual ring sequence [5]. So a full signal cycle also considers green phases for the turning lanes. This control is more complex because not only are the phase lengths chosen but the order of the phases is also determined. The addition of an eight phase signal makes the graph ants transverse larger. At each node an ant decides which phase comes next and how long the phase should be.

The next step is a five intersection network with a central intersection and four surrounding intersections. Utilizing a decentralized control strategy, each intersection has a graph of candidate signal cycles for the ants to traverse. Intersections have internal movements that receive vehicles from other intersections. Traffic from adjacent intersections do not arrive according to a Poisson distribution, but would follow Robertson's platoon flows. Each intersection communicates with other intersections, sending traffic data and arrival rates. The rolling horizon at each intersection is long enough to utilize all available information from neighboring intersections.

References

[1] Transportation research center, "Traffic network study tool: TRANSYT-7F software summary," University of Florida, 1987.

[2] P. B. Hunt, Robertson, D. I., et al., "The SCOOT on-line traffic signal optimization technique," *Traffic engineering and control,* April 1982.

[3] P. Lowrie, "The Sydney coordinated adaptive control systems - principles, methodology, algorithms," *IEE conference publication,* vol. 207, 1982.

[4] W. Wei, Y. Zhang, J. B. Mbede, Z. Zhang, and J. Song, "Traffic signal control using fuzzy logic and MOGA," *Proceedings of the IEEE conference on systems, man, and cybernetics,* 2001.

[5] X.-H. Yu, "Decentralized adaptive signal control for traffic networks," in *Transportation research trends*, P. O. Inweldi, Ed. New York: Nova Science Publishers, 2008.

[6] M. Dorigo and T. Stutzle, *Ant colony optimization*. Cambridge, Massachusetts: MIT Press, 2004.

[7] M. Dorigo and C. Blum, "Ant colony optimization theory: A survey," *Theoretical Computer Science,* vol. 344, pp. 243-278, 2005.

[8] D. Angus, "Ant colony Optimization: From Biological inspiration to an Algorithmic Framework," *Technical Report, CISCP,* 2006.

[9] M. Papageorgiou, C. Diakaki, V. Dinopoulou, A. Kotsialos, and W. Yibing, "Review of road traffic control strategies," *Proceedings of the IEEE,* vol. 91, pp. 2043-2067, 2003.

[10] X.-H. Yu and W. W. Recker, "Stochastic adaptive control model for traffic signal systems," *Transportation Research Part C: Emerging Technologies,* vol. 14, pp. 263-282, 2006.

[11] T. Riedel and U. Brunner, "A control algorithm for traffic lights," in *First IEEE Conference on Control Applications*, 1992, pp. 901-906

[12] J. Y. K. Luk, "Two traffic-responsive area traffic control methods: SCAT and SCOOT," *Traffic engineering and control,* 1984.

[13] N. Gartner, "OPAC: A demand-responsive strategy for traffic signal control," *Transportation research record, no. 906,* 1983.

[14] A. J. Miller, "A computer control system for traffic networks," *Proc. 2nd Int. Symp. Traffic Theory,* pp. 200-220, 1963.

[15] J.-D. Schmöcker, S. Ahuja, and M. G. H. Bell, "Multi-objective signal control of urban junctions - Framework and a London case study," *Transportation Research Part C: Emerging Technologies,* vol. 16, pp. 454-470, 2008.

[16] C. Ledoux, "An urban traffic flow model integrating neural networks," *Transportation Research Part C: Emerging Technologies,* vol. 5, pp. 287-300, 1997.

[17] M. Papageorgiou, A. Messmer, J. Azema, and D. Drewanz, "A neural network approach to freeway network traffic control," *Control Engineering Practice,* vol. 3, pp. 1719-1726, 1995.

[18] C. Quek, M. Pasquier, and B. B. S. Lim, "POP-TRAFFIC: a novel fuzzy neural approach to road traffic analysis and prediction," *Intelligent Transportation Systems, IEEE Transactions on,* vol. 7, pp. 133-146, 2006.
[19] V. S. Borkar, *Topics in controlled Markov chains*: Longman scientific & technical group, 1991.
[20] R. D. Yates, Goodman, D.J., *Probability and Stochastic Processes*, second ed. New York: Wiley, 2005.
[21] R. T. Luttinen, "Statistical Analysis of Vehicle Time Headways." vol. Doctor of Science in Technology: Helsinki University of Technology, 1996.
[22] B. V. Ramanathan, "Actuated signal network simulation program," Institute of Transportation Studies University of California Irvine 1993.
[23] S. M. Ross, *Applied Probability Models with Optimization Applications*. San Francisco: Holden-Day, 1970.
[24] F. M. Dekking, C. Kraaikamp, H. P. Lopuhaa, and L. E. Meester, *A modern introduction to probability and statistics*: Springer, 2005.
[25] J. Vaslin, "Ant Colony Optimization: An adaptive nature inspired algorithm explained, concretely implemented, and applied to routing protocols in wired and wireless networks," India: International Institute of Information Technology, 2006.
[26] M. Dorigo and L. M. Gambardella, "Ant colonies for the traveling salesman problem," *Biosystems,* vol. 43, pp. 73-81, 1997.
[27] B. Bullnheimer, R. F. Hartl, and C. Strauss, "An improved Ant System algorithm for the Vehicle Routing Problem," *Annals of Operations Research,* vol. 89, pp. 319-328, 1999.
[28] R. Montemanni, L. Gambardella, A. Rizzoli, and A. Donati, "Ant Colony System for a Dynamic Vehicle Routing Problem," *Journal of Combinatorial Optimization,* vol. 10, pp. 327-343, 2005.

Appendix A Table of wait times

The following tables show the average vehicle delay in the ACO and Fully-Actuated control. The first row gives the vehicle arrival rate in vehicles per hour per movement. The rest of the columns give the average vehicle delay during each trial.

Table A-1. ACO Control average vehicle delays

400	450	500	550	600	650	700	750	800	850
3.60	3.79	3.72	3.61	4.19	5.13	5.44	5.73	6.81	8.62
3.75	3.55	4.01	3.87	4.53	4.54	5.03	6.77	7.47	10.50
3.61	4.16	3.81	4.24	4.17	4.97	5.15	6.41	7.57	8.96
3.39	3.39	3.91	4.31	4.22	4.60	5.64	6.70	6.60	11.20
3.65	3.82	3.80	4.43	4.38	5.21	5.01	5.89	6.53	8.01
4.05	3.15	3.96	4.00	4.18	4.49	5.25	6.23	8.92	11.09
3.31	3.88	3.68	4.12	4.30	4.57	5.73	5.45	7.18	13.39
3.47	3.48	3.51	4.22	4.07	4.99	5.05	5.57	7.87	9.54
3.06	3.22	3.92	3.60	4.13	4.46	4.40	5.92	8.13	7.46
3.20	3.74	3.76	4.20	4.41	4.57	5.09	5.58	6.84	9.68
3.24	3.63	3.84	3.85	3.99	5.06	5.08	5.21	7.00	12.98
3.48	3.59	4.25	4.03	4.27	4.65	4.77	6.28	7.34	11.02
2.94	3.68	3.54	3.58	5.06	4.96	5.42	6.15	6.49	10.10
3.41	3.41	3.81	4.04	4.29	4.63	5.48	6.95	7.61	8.18
4.08	3.34	3.90	4.06	4.00	4.96	6.27	5.45	6.85	10.43
3.90	3.37	4.35	3.95	3.94	4.92	5.13	4.97	6.70	8.88
3.70	3.59	3.74	4.09	4.60	5.63	4.85	6.10	7.77	10.51
3.73	3.88	3.96	3.71	4.85	5.16	5.80	5.87	8.44	15.89
3.25	3.49	3.53	4.42	4.20	5.88	4.97	5.50	7.82	9.02
3.57	3.71	3.88	3.85	4.52	4.60	4.68	5.90	6.92	10.74
3.77	3.73	3.81	4.27	3.86	4.73	6.28	6.77	6.82	9.65
3.48	3.39	3.70	4.03	4.34	4.78	6.00	5.79	6.99	8.22
3.62	3.70	3.44	3.91	4.02	4.19	4.83	5.38	####	9.24
3.40	3.83	3.84	4.41	4.38	4.60	5.38	5.77	6.97	8.58
3.51	3.48	3.90	4.01	4.38	4.86	4.82	5.72	7.90	10.27
3.50	3.23	3.66	4.09	4.51	4.86	5.40	5.60	6.68	9.18
3.32	3.99	4.10	4.16	4.22	4.25	5.38	6.13	6.83	13.16
3.76	3.60	3.57	4.30	4.85	4.48	4.86	5.24	6.92	12.03

3.82	3.46	3.92	3.90	4.64	4.28	5.89	5.39	7.85	8.58
3.07	3.58	3.83	4.04	4.29	5.06	5.77	5.17	6.79	7.74
3.48	3.41	3.83	3.77	4.35	4.95	5.17	5.57	7.29	9.34
3.81	3.41	3.78	4.01	4.59	4.54	4.82	7.04	6.27	8.57
3.41	3.54	3.92	3.98	4.45	4.57	6.02	5.56	7.80	12.73
3.71	3.51	4.12	4.20	4.18	4.00	5.33	5.51	6.77	9.89
3.51	3.55	3.26	3.82	4.53	4.41	4.98	5.94	7.82	10.77
3.77	3.76	3.73	4.30	4.54	4.92	5.03	5.56	8.21	10.64
3.45	3.51	3.91	4.14	4.03	5.09	5.84	5.77	6.42	9.78
3.43	3.99	3.63	4.06	4.24	4.76	5.78	7.75	8.41	10.10
3.49	3.40	3.76	3.77	4.70	4.94	5.15	6.11	7.07	10.30
3.28	3.29	3.72	4.11	4.04	5.03	4.97	5.24	8.01	10.46

Table A-2. Fully Actuated Control average vehicle delays

400	450	500	550	600	650	700	750	800	850
2.87	2.62	2.82	3.27	3.78	5.75	9	7.36	13.5	17.55
3.08	2.97	3.28	3.74	4.35	4.67	4.68	12.6	11.9	19.02
2.54	3.12	3.43	2.94	3.93	5.41	6.82	10.1	15.8	17.09
2.65	2.85	3.19	3.77	3.43	4.75	8.3	10.8	9.88	19.05
2.48	2.39	3.73	3.93	3.47	5.89	5.73	8.84	10.9	14.09
2.9	2.86	2.93	3.27	3.75	4.27	5.55	13.3	14.3	20.53
2.58	2.73	2.94	3.27	3.82	4.48	7.31	7.64	10.3	22.93
2.83	2.55	2.92	3.85	4.32	4.08	6.03	8.75	10.6	17.02
2.56	2.65	3.52	3.47	3.78	4.63	5.96	7.02	12.3	14.81
2.52	2.9	2.94	3.74	4.28	4.83	5.58	6.89	10.6	18.75
2.55	2.75	2.98	3.48	3.48	4.9	4.75	9.13	13.6	21.13
2.78	2.59	3.04	3.55	4.05	4.89	6.2	10.5	13.6	21.37
2.53	2.76	3.28	3.89	4.06	4.66	6.18	8.81	11.6	16.2
2.28	2.69	3.08	3.74	3.78	6.03	8.11	9.41	13.6	16.77
2.54	2.68	3.07	4	3.77	5.49	7.9	7.83	9.83	20.28
2.93	2.92	3.23	3.22	3.45	5.66	6.62	7.33	11.4	16.06
2.64	2.66	2.96	3.86	4.24	5.21	7.19	13.2	12.8	20.62
2.51	2.94	3.06	3.19	4.02	5.51	6.09	7.27	16.1	23.32
2.75	2.95	2.7	4.52	3.59	6.47	8.37	6.27	14.1	17.57
2.37	2.9	3.12	2.76	3.94	5.29	5.84	7.88	15.3	18.87
2.45	2.96	3.17	3.54	3.24	5.11	6.09	11.6	13.1	17.93
2.49	2.6	3.01	3.94	3.96	4.88	8.1	8.28	13.2	16.84
2.49	2.76	3.08	3.81	3.8	4.12	5.03	7.42	14.6	16.9

2.45	2.73	3.07	3.17	4	4.92	4.66	10.6	14.7	15.2
2.41	2.79	3	3.88	4.16	4.73	8.29	7.79	14.2	19.92
2.62	2.76	3.42	3.75	3.9	6.08	6.35	8.64	13.9	15.8
2.2	3.16	3.37	3.81	4.14	3.68	5.6	8.61	12.9	20.51
2.74	2.82	3.41	3.15	4.54	4.98	5.98	6.4	10.9	26.17
2.46	2.65	3.48	3.52	3.6	5.8	6.97	8.35	17.1	17.72
2.67	3.01	3.14	3.56	3.92	4.59	6.53	7.49	10.6	15.52
2.27	2.98	3.26	4.38	3.85	5.56	7.12	7.29	15.2	18.03
2.13	2.75	3.26	3.01	4.46	6.24	6.59	11.6	13	15.75
2.34	2.59	2.77	3.15	4.5	4.95	9.45	9.13	16.2	23.24
2.63	2.79	3.34	3.58	4.16	4.17	7.07	9.66	8.89	18.35
2.27	3	3.02	3.05	3.76	5.3	5.63	11	14.6	18.64
2.55	3.02	2.92	3.25	4.02	5.29	6.12	9.66	16.6	19.55
2.52	2.53	2.98	3.52	3.63	5.48	6.77	8.87	12.9	18.33
2.49	2.74	3.51	3.53	3.62	5.87	5.26	11.7	15.2	18.26
2.49	2.87	2.84	3.26	4.33	5.48	5.58	8.56	11.7	19.89
2.41	3.13	2.58	3.5	4.18	5.8	6.26	6.97	16.5	19.55

Appendix B Matlab Code

%%start.m

```
global volume headway redtime;
volume=800;                     %Vehicles per hour
headway=2;                      %Spacing between cars on internal
                                %movements
redtime =2;                     %All red time for intersections

NumCars=fix(volume*.5);         %Number of cars generated in each
                                    %direction
gencar()     %Generate vehicles
```

%%gencar.m

```
%outputs NumCars number of cars with shifted exponential distribution

%first generate spacing between cars
arrival = headway - (3600/volume -headway)*log(rand(4,NumCars));

%add spacing to get arrival times
for j = 2:NumCars
     arrival(1:4,j)= arrival(1:4,j-1)+arrival(1:4,j);
end
```

%%inctime.m

```
%Traffic flow is simulated and control signals generated
%Time is incremented to the next signal choice, once the optimal signal
is
%found
%ACO parameters
iterations=75;          %Number of iterations before choosing a switch
                              %time
numants=10;             %number of ants used
evaporation =.8;        %pheromone evaporation rate
local=1;                %1 if local search is to be used, 0 otherwise
locstep = 5;            %How often local search is performed
localneighbor=4;        %Size of local search neighborhood
heur=1;                 %1 if heuristic information is to be used, 0
                              %otherwise
elitist =1;             %1 if elitist ant is to be used, 0 otherwise
rank =1;                %1 if rank-based update is to be used, 0
                              %otherwise
N=10;                           %number of ants used in rank-based
totalsig = zeros(1,400);        %array containing signal switch times
totalqueue = zeros(4,400);      %array containing queue on each
                                    %movement at switch times
```

```
await=zeros(4,NumCars);          %array containing delay of each
                                     %vehicle

t=0;                             %start time
initial = zeros(4,1);            %number of cars in each direction at
                                     %start of each phase
totalsig(1) = t;                 %totalsig is vector of switching times
totalqueue(:,1)=initial;         %Array of queue length at each change
                                  % in each direction
totalwait=zeros(4,1);            %Array of vehicle wait times
green = [1,0,1,0]';              %1 if green on a movement otherwise 0
next=ones(4,1);                  %Index of next arrival
l = volume/3600;                 %lambda, vehicle arrival rate
inc=1;                           %Number of signal transitions
currentwait=zeros(4,1);          %The current wait of vehicles waiting
                                  % at the queue
cycle = zeros(4,1);


while t<min(arrival(:,size(arrival,2)))-200  %Loops until stopping time

    greendir= mod(green(1)+1,2)+1;   %1 if movement 1 is green,
                                         %otherwise 2
    reddir= mod(greendir,2)+1;       %1 if movement 1 is red ,
                                         %otherwise 2


    ACO()                            %run ACO to find optimal
                                         %signal

    ccc = [t1,t2,t3];
    sigset(1) = t;
    sigset(2) = sigset(1)+gsetting(1);
    sigset(3) = sigset(2)+gsetting(2);



    movecars()      %The chosen path is evaluated and vehicle's wait
                        time is computed



    t = sigset(2);      %Advance time to next signal change

    green=~green;                    %switch green to other direction
    totalsig(inc) =  t;              %keep track of signal
    totalqueue(:,inc)=initial;       %keep track of queue lengths
    inc=inc+1;


end
```

%%ACO.m

```
%Pheromone levels are initialized, then ants construct new solutions on
%each iteration

prob = cat(3,cat(1,ones(1,26),zeros(25,26)),ones(26,26,1));
prob2=prob;             %prob2 used with heuristic

%prob is a 3-d array consisting of 2, 26 by 26 matrices, dimension. The
%matrix in the first slot is for the next time change and
%the matrix in the second slot is for the time change after that. The
%ij-th slot of the 2nd matrix gives the probability of an ant moving
%from time t+i to time t+i+j
%It is updated after each iteration

weight = 1:26;
weight = exp(-abs(weight -  (max(green.*initial)*2 - 2)+6)/5);
%Weight is used for heuristic information

gminwait=10000;

for iteration = 1:iterations       %based on ants


        updatepher()        %Ants create solution and find optimal path


end
```

%%updatepher.m

```
% At each iteration ants create and evaluate candidate solutions, here
% they deposit pheromone, based on ACO parameters, and pheromone is
% evaporated.

minwait=10000;
if local == 1 && mod(iteration,locstep )==0    %Do local search on 5th
                                                %iteration

        locmin(1)=max(1,gsetting(1)-6-localneighbor);
        locmax(1)=min(26,gsetting(1)-6+localneighbor);
        locmin(2)=max(1,gsetting(2)-6-localneighbor);
        locmax(2)=min(26,gsetting(2)-6+localneighbor);


locprob = {ones(1,locmax(1)-locmin(1)+1),ones(locmax(1)...
locmin(1)+1,locmax(2)-locmin(2)+1)};


optcen()
        antwait =waittime;              %waittime of each ant
        aset = setting;                 %signal setting of each ant
```

```
        for a = 2:numants                %Number of ants
            optcen()                    %calc waittime for each ant
            antwait = [antwait ;waittime];
            aset = [aset; setting];
        end


else
        optcen2()
        antwait =waittime;               %wait time of each ant
        aset = setting;                  %signal setting of each ant
        for a = 2:numants                %Number of ants
            optcen2()                    %calc wait time for each ant
            antwait = [antwait ;waittime];
            aset = [aset; setting];
        end
end

%Ranking

sorted = sortrows([antwait,aset]);
aset = sorted(:,2:3);

if minwait < gminwait        %Pick best solution so far
    gminwait = minwait;
    goptsig = optsig;
    gsetting = [goptsig(2)-goptsig(1),goptsig(3)-goptsig(2)] ;
end

if ~(mod(iteration, localneighbor )==0 && local ==1)  %If not using
                  %local search or not local search step

    prob = evaporation*prob;             %Evaporation of pheromone

    if rank == 1

    for r = 1:size(aset,1)
        for s = 1:size(aset,2)    %Add weight to best solution of last
                                        %group

            if s ==1
                prob(1,aset(r,s)-4-2,s) = prob(1, aset(r,s)-4-2, s)...
+(max(0,(N-r))/(antwait(r)));
            else
                prob(aset(r,s-1)-4-2,aset(r,s)-4-2,s)=...
prob(aset(r,s-1)-4-2,aset(r,s)-4-2,s)+(max(0,(N-r))/(antwait(r)));
            end
        end
    end


    else
        for r = 1:size(aset,1)
            for s = 1:size(aset,2)    %Deposit pheromone on paths
```

```
                if s ==1
                    prob(1,aset(r,s)-4-2,s) = ...
prob(1, aset(r,s)-4-2, s) +(1/antwait(r));
                else
                    prob(aset(r,s-1)-4-2,aset(r,s)-4-2,s)=...
prob(aset(r,s-1)-4-2,aset(r,s)-4-2,s)+(1/antwait(r));
                end
            end
        end
    end


    if elitist ==1                  %add elitist ant's pheromone
                                    %deposit
        prob(1,gsetting(1)-4-2,1) = prob(1,gsetting(1)-4-2,1) +...
(10)/ gminwait ;
        prob(gsetting(1)-4-2,gsetting(2)-4-2,2) = ...
prob(gsetting(1)-4-2,gsetting(2)-4-2,2) +10/ gminwait ;
    end

    if heur==1                      %include heuristics
    prob2 = cat(3,cat(1,prob(1,:,1).*weight,zeros(25,26)),prob(:,:,2));
    else
        prob2=prob;
    end

end
```

optcen.m

```
%Each ant creates a candidate solution randomly based on pheromone
%deposits and then the expected delay is computed to evaluate the
%solution
%This is used on local search step


global volume;


   %Randomly pick next solution based on pheromone
   sumprob1 = sum(locprob{1},2);

   sumprob2 = sum(locprob{2},2) ;


   sumprob = {sumprob1,sumprob2};

s=1;j=1;offset=0;
setting=zeros(1,2);
signal=zeros(2,1);

while s < 3
signal(s)=rand*sumprob{s}(j) ;   %Choose signal switch times based on
```

```
                                %pheromone

    i =1;
        while signal(s) > sum(locprob{s}(j+offset,1:i)) %pick the
                                                        %signal
            i = i+1;
        end

        j=i-locmin(s)+1;
        j=1;

        setting(s)=i+3+locmin(s)+redtime;

    s = s+1;
end
                %sigset is the next two switch times
                %setting is length of next two phases



sigset = [t];

t1= initial;

for i = 1:size(setting,2)
   sigset =  cat(2, sigset, sigset(i)+setting(i));
end

%t1 is vehicles in queue initially
%t2 is expected number of vehicles in queue at next signal change
%t3 is expected number of vehicles in queue at second signal change

%On green movements
t2 = green.*max(0 , t1 - floor((setting(1)-redtime)/headway +1)+...
    volume/3600* min(headway*(t1-1)/(1-
(volume/3600)*headway),setting(1)));
%On red movements
t2 = t2 + ~green.*(t1+volume/3600*setting(1));


t3 = ~green.*max(0 , t2 - floor((setting(2)-redtime)/headway +1)+...
    volume/3600* min(headway*(t2-1)/(1-
(volume/3600)*headway),setting(2)));
t3 = t3 + green.*(t2+volume/3600*setting(2));

compwait()                      %Compute wait time of signal choice


if waittime < minwait
    minwait = waittime;
    optsig = sigset;
    optsetting = setting;
end
```

optcen2.m

```
%Each ant creates a candidate solution randomly based on pheromone
%deposits
%and then the expected delay is computed to evaluate the solution
%This is used when not using local search

sumprob = sum(prob2,2);   %Randomly pick next solution based on
                                %pheromone
s=1;j=1; setting=zeros(1,2);
signal=zeros(2,1);
while s < 3
signal(s)=rand*sumprob(j,1,s) ;   %Choose signal switch times based on
                                  %pheromone
    i = 1;
    sp = prob2(j,1:i,s);

        while signal(s) > sp %pick the signal
        i = i+1;
        sp = sp + prob2(j,i,s);
        end

        j=i;
        setting(s)=i+4+redtime;
    s = s+1;
end
                %sigset is the next two switch times
                %setting is length of next two phases
sigset = t;

t1= initial;

for i = 1:size(setting,2)
   sigset =  [ sigset, sigset(i)+setting(i)];
end

%On green movements
t2 = green.*max(0 , t1 - floor((setting(1)-redtime)/headway +1)+...
    volume/3600* min(headway*(t1-1)/(1-
(volume/3600)*headway),setting(1)));
%On red movements
t2 = t2 + ~green.*(t1+volume/3600*setting(1));

t3 = ~green.*max(0 , t2 - floor((setting(2)-redtime)/headway +1)+...
    volume/3600* min(headway*(t2-1)/(1-
(volume/3600)*headway),setting(2)));
t3 = t3 + green.*(t2+volume/3600*setting(2));

compwait()              %Compute expected wait time of signal
if waittime < minwait
    minwait = waittime;
    optsig = sigset;
    optsetting = setting;
end
```

%%Compwait.m

```
%Computes the expected wait time of a signal

cycle = zeros(4,1);

%computes lower bound on extra wait time
M = 30/headway+1-30*volume/3600;
q= t3 +(~green)*7*volume/3600;
for k =1:4
    while t3(k) > M - (~green(k))*7*volume/3600 + cycle(k)*...
(M - 7*volume/3600) &&cycle(k) <5
        cycle(k) = cycle(k) +1;
        q(k) = t3(k) -(M - (~green(k))*7*volume/3600 + (cycle(k)-1)*...
(M - 7*volume/3600));
        if cycle(k) == 5 && iteration ==70
            t
        end
    end
end

%compute initial vehicles released, new arrivals released and not
%released
initrel = green.* min(1 +  fix((setting(1)-redtime)/headway),t1)+...
~green.* min(1 +  fix((setting(2)-redtime)/headway),t2);
newrel = green.*max(0,min(1+fix((setting(1)-redtime)/headway)-t1,...
    l * (t1-1)*headway/(1- l * headway)))+...
~green.*max(0,min(1+fix((setting(2)-redtime)/headway)-t2,...
    l * (t2-1)*headway/(1- l * headway)));

notrel = green.*(t1 - initrel)+...
    ~green.*(t2-initrel);


%compute wait time of each group of vehicles
inwait = initrel.*(initrel-1)*headway/2;
newwait = green.*newrel*headway.*(t1+1)/2+...
    ~green.*newrel*headway.*(t2+1)/2;
notrwait = green.*notrel*setting(1)+...
    ~green.*notrel*setting(2);
nnrwait = green.*max(0,l* (setting(1)-redtime- newrel*headway).^2/2.*...
    sign(((t1-1)*headway)/(1-l * headway)-setting(1)+redtime))+...
    ~green.*max(0,l* (setting(2)-redtime- newrel*headway).^2/2.*...
    sign(((t2-1)*headway)/(1-l * headway)-setting(2)+redtime));

new = green.*max(0,l * q*headway/(1- l * headway));
wait3 = (~green).*t3*(5+2)+...
cycle*(30+5+2)^2*l/2+M*(M-1)/2.*cycle+... %wait time from uncleared
                                          %cycles
    max(0,q.*(q-1)*headway/2) +...       %cars at beginning of final
                                          %cycle
    new*headway.*(q+1)/2;         %cars
```

```
redwait = (green).*(t2*setting(2) +...
    (volume/3600).*(setting(2).^2)/2 )+...
(~green).*(t1*setting(1) +...
    (volume/3600).*(setting(1).^2)/2 );

%Total expected wait time
wait = inwait+newwait+notrwait+nnrwait+wait3+redwait+currentwait;

%Average expected wait time
waittime =(sum(wait))/((sigset(3)-sigset(1))*volume/900+sum(initial));
```

%%Movecars.m

```
%Move cars according to the optimal control

setting = sigset(2)-sigset(1);
queue =initial;

for k =greendir:2:greendir+2        %clear green cars on green
                                          %movements
    m=next(k); n=0;
    for j=0:initial(k)-1            %loop through vehicles in queue to
                                    %determine if relased or not

        if headway* j <=setting(1)-redtime   %Vehicle relased
            await(k,m+j)=await(k,m+j)+headway*(j);
            n=n+1;
        end

    end

    m=m+n;

    next(k)=m;               %next is now the index of the next vehicle
                              %to be released

    initial(k)=0;

    while arrival(k,m) <sigset(2)-redtime     %new arrivals while
                                         %signal is still green



        if arrival(k,m)<sigset(1)+(queue(k)-1)*headway  %if arrives
%before traffic has been released

           if queue(k)*headway <= setting(1)-redtime     %if initial
%queue will be released on current signal

               await(k,m)=sigset(1)+ headway*queue(k) - arrival(k,m);
               next(k)=next(k)+1;
               queue(k)=queue(k)+1;
```

```
            else
              %If queue is too long then calculation begins on next (red)
                %phase

               initial(k)=initial(k)+1;
               queue(k)=queue(k)+1;
            end

        else             %if vehicle arrives when queue is empty then
                        %no wait
            await(k,m)=0;
            next(k)=m+1;
        end
       m=m+1;
    end

    if arrival(k,next(k)+initial(k))<sigset(2) && ...
      arrival(k,next(k)+initial(k))>sigset(2)-2
        initial(k) = initial(k)+1;
    end
    currentwait(k) = ...
 sum(max(0,(-arrival(k,next(k):next(k)+initial(k))+sigset(2))));
end


for k=reddir:2:reddir+2          %Loop over red directions

    m=next(k);

    initial(k)=0;                %reset initial



    while arrival(k,m) <sigset(2)
        await(k,m)= sigset(2) - arrival(k,m);       %add wait time to
%vehicles in queue
        initial(k)=initial(k)+1;                    %Count vehicles in
%queue
        m=m+1;
    end


currentwait(k) = sum(await(k,next(k):next(k)+max(0,(initial(k)-1))));
%Update current wait
end
```

%%Fullact.m

```
%Fully actuated control

t=0;                        %start time
totalsig = t;                       %totalsig is vector of switching times

totalwait=zeros(4,1);               %Arrary of vehcile wait times
green = [1,0,1,0]'; next=ones(4,1);
    greendir= mod(green(1)+1,2)+1;
    reddir= mod(greendir,2)+1;
initial = zeros(4,1);               %number of cars in each direction at
                                        %start of optimization
totalqueue=initial;                     %Array of quenelenth at each change
                                        %in each direction
inc =1;
chg=zeros(4,1);
currentwait=zeros(4,1);
nextarrival = ones(4,1);
changegreen=0;
first=1;
fulltime=0;

while t<min(arrival(:,size(arrival,2)))-100

    sigset = [t];
    nextchangefullact()

t=t +redtime;

    if endflag ==1
        for k=reddir:2:reddir+2
            m=nextarrival(k);

            while arrival(k,m) <t
            m=m+1;
            end
        end
    end

goptsig =[sigset,t,t+7];
sigset = [sigset, goptsig(2),goptsig(3)];
movecars()

green=~green;
        greendir= mod(green(1)+1,2)+1;
        reddir= mod(greendir,2)+1;
        totalsig = [totalsig, t];
        totalqueue=[totalqueue,initial];
        first =0;
        [totalsig;totalqueue];

end
```

%%nextchangefullact.m

```
%Inputs are current quene
%arrival times
%green
%Determines next time fully actuated controller changes the signal

changegreen=0;  %Is 1 if conditions for changing signal are met
endflag =0;     %Is 1 if signal should be changed
time =0;        %Length of signal

for k =greendir:2:greendir+2 %clear green cars
    time(k) = max(0,(initial(k)-1))*headway;

end

if max(time) >30                              %if time to release cars in
                               %queue is longer than max green time
     mintime = [time(greendir),time(greendir+2)];
         [maxtime,argmax]=max(mintime);
         [mintime,argmin]=min(mintime);


        nextarrival(greendir+2*(argmax-1))=...
nextarrival(greendir+2*(argmax-1))+16;
        while arrival(greendir+2*(argmin-1),...
nextarrival(greendir+2*(argmin-1))) < t+30
            nextarrival(greendir+2*(argmin-1))=...
nextarrival(greendir+2*(argmin-1))+1;
        end


    time=30;
    endflag=1;
    t=t+30;

else
    nextarrival = nextarrival+initial.*green;    %All cars in queue are
                                                      %released
end


while endflag ~= 1
    endclear = [0,0];           %Is 1 if there are no more cars
                                    %arriving
    if max(time) ~=0            %If there are cars in queue
        while and(endclear(1),endclear(2)) ~=1

            if arrival(greendir,nextarrival(greendir))<...
t+time(greendir)+2 %If car arrives before queue is released
                time(greendir) = time(greendir)+2;
                nextarrival(greendir)=nextarrival(greendir)+1;
            else
```

```
                endclear(1)=1;
            end

            if arrival(greendir+2, nextarrival(greendir+2))< ...
t+time(greendir+2)+2 %If car arrives before queue is released
                time(greendir+2) = time(greendir+2)+2;
                nextarrival(greendir+2)=nextarrival(greendir+2)+1;

            else
                endclear(2)=1;
            end

            if max(time)>29                %If cars keep arriving until
                                                %the max green time
                endflag=1;
                endclear(1)=1;
                endclear(2)=1;
                t=t+30;
            end

        end

            mintime = [time(greendir),time(greendir+2)]; %Which green
                                        %signal requires less green time
            [mintime,argmin]=min(mintime);
        %Let cars pass on shorter green signal if other direction is
            %still green

        while arrival(greendir+2*(argmin-1),...
nextarrival(greendir+2*(argmin-1)))< t+max(time)*(1-endflag)
            nextarrival(greendir+2*(argmin-1))=...
nextarrival(greendir+2*(argmin-1))+1;
        end


    else   %If  queue is empty
       time = max(time);         %Make time a scalar

        if arrival(greendir,nextarrival(greendir))< t+2 %If car arrives
                                            %before minimum green if up

            time = time+2;
            nextarrival(greendir)=nextarrival(greendir)+1;
            if arrival(greendir+2,nextarrival(greendir+2))< t+2
                nextarrival(greendir+2)=nextarrival(greendir+2)+1;

            end
        elseif arrival(greendir+2, nextarrival(greendir+2))< t+2
            time = time+2;
            nextarrival(greendir+2)=nextarrival(greendir+2)+1;

        else
            endflag=1;
        end
```

```
        if fulltime+ time >28
            endflag =1;
            t = t +time;
        end
        %end
    end
    fulltime=max(max(time),1)+fulltime; %Check if reached minimum green
                                                %time
    if fulltime <= 5
        endflag=0;
    end

    if endflag ~=1

    t=t+max(max(time),1)   ;

    time =0;
    end

    if endflag ==1
        time =max(time);
        if arrival(reddir,nextarrival(reddir)) <t + time ...
||arrival(reddir+2,nextarrival(reddir+2)) <t + time||fulltime+ time >28

             changegreen=1;
            fulltime =0;
        else


            endflag =0;
            t= t+1;

        end
    end

end
```